

T.C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

**TCP TIKANIKLIK KONTROL ALGORİTMASININ
ANALİZİ VE İYİLEŞTİRİLMESİ**

AHMET KEMAL AYGİN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2008

T.C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

**TCP TIKANIKLIK KONTROL ALGORİTMASININ
ANALİZİ VE İYİLEŞTİRİLMESİ**

Ahmet Kemal AYGİN

YÜKSEK LİSANS TEZİ

DANIŞMANI: Yrd. Doç. Dr. Hacı Ali MANTAR

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2008



**GEBZE YÜKSEK
TEKNOLOJİ ENSTİTÜSÜ**

YÜKSEK LİSANS/DOKTORA JÜRİ ONAY FORMU

G.Y.T.E. Mühendislik ve Fen Bilimleri/Sosyal Bilimler Enstitüsü Yönetim Kurulu'nun .../.../2008 tarih ve 2008/... sayılı kararıyla oluşturulan jüri tarafından .../.../2008 tarihinde tez savunma sınavı yapılan Ahmet Kemal AYGİN tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (TEZ DANIŞMANI) : Yrd. Doç. Dr. Hacı Ali MANTAR

ÜYE : Doç. Dr. İbrahim SOĞUKPINAR

ÜYE : Yrd. Doç. Dr. Mehmet Ali ARSLAN

ONAY

G.Y.T.E. MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ Yönetim Kurulu'nun __/__/__ tarih ve __/__/__ sayılı kararı.

İMZA/MÜHÜR

ÖZET

TEZİN BAŞLIĞI : TCP TIKANIKLIK KONTROL ALGORİTMASININ ANALİZİ VE İYİLEŞTİRİLMESİ

YAZAR ADI :Ahmet Kemal AYGİN

İnternette en çok kullanılan taşıma protokolü TCP'dir. Bu protokol ile dosya transferi, elektronik posta gönderme gibi birçok işlemi gerçekleştirmekteyiz. Bu işlemlerdeki güvenilirlik ve hız, birtakım algoritmalarla sağlanmaktadır. TCP protokolü; yavaş başlangıç, tıkanıklık önleme, hızlı yeniden iletim ve hızlı geri dönüş algoritmalarını temel alarak birçok sürüm oluşturmuştur. Bu sürümler, TCP Tahoe, TCP Reno gibi farklı isimler almışlardır.

Standart TCP zaman içerisinde ortam ve şartlara göre kendini yenilemek zorunda kalmıştır. Bunun nedeni standart TCP'deki algoritmaların artan bant genişliği ve büyüyen mesafeler sebebiyle yetersiz kalmasıdır. Ayrıca TCP'nin veri akışını kontrol etmek için kullandığı algoritmalar da zaman içerisinde yeni çıkan uygulamaların değişik özellikleri ve Internet'te artan trafik yükünden dolayı bir çok yeni problemler oluşmuştur.

Bu çalışmada, *TCP Tahoe* ve *TCP Reno* protokollerini iyileştirerek modifiye edilmiş bir *TCP Reno* protokolünün gerçekleştirilmiştir. Bu çalışmada öncelikli olarak *TCP Tahoe* ve *TCP Reno* algoritmaları incelenecek daha sonra geliştirilen modifiye *TCP Reno* algoritması bir OPNET simülasyonu ile gerçekleştirilecektir. TCP'de yoğunlaştığımız asıl konu ise **Congestion Control** (Tıkanıklık kontrolü)'dür. Çünkü, TCP algoritmaları tıkanıklık varsayımları üzerine gelişmiştir.

SUMMARY

**THESIS TITLE : THE ANALYSIS AND MODIFICATION OF TCP
CONGESTION CONTROL ALGORITHM**

AUTHOR : Ahmet Kemal AYGİN

The most used transport protocol in the Internet is TCP. This protocol is used for file transfer, e-mail, telnet, etc. The reliability and the speed of this protocol are provided by some algorithms including slow start, congestion avoidance, fast retransmission and fast recovery. TCP Tahoe and TCP Reno are two well-known version of TCP.

Standard TCP is forced to modify itself for the newer conditions and environment in the time period. However, the standard TCP was not enough for the increasing bandwidth and growing distances. Also, the algorithms that TCP used for flow control has been substantially changed with the attributes of the newer applications and the increasing load of traffic in the Internet.

The purpose of this work is to implement a modified TCP Reno protocol by modifying TCP Tahoe and TCP Reno. First, TCP Tahoe and TCP Reno algorithms will be observed then modified TCP Reno algorithm will be implemented on an OPNET simulation. The main subject which we will focus on TCP is the congestion control algorithm. Because, TCP algorithms are developed on the congestion assumptions.

TEŞEKKÜR

Bu çalışmanın tüm aşamalarında engin bilgi birikimini, deneyimini ve ilgisini hiçbir zaman esirgemeyen ve bana destek olan danışman hocam Yrd. Doç. Dr. Hacı Ali Mantar, edindiği tecrübeyi benimle paylaşan ve pek çok noktada her türlü desteği veren Ömer EROĞLU'na ve son olarak beni bu günlere getiren aileme sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER DİZİNİ

1. GİRİŞ	1
2.1.TCP TAHOE	5
<i>2.1.1.Yavaş başlangıç ve tıkanıklık önleme algoritmaları</i>	<i>5</i>
<i>2.1.3. Hızlı yeniden iletim</i>	<i>8</i>
2.2.TCP RENO	9
<i>2.2.1.Hızlı geri dönüş</i>	<i>10</i>
3. MODİFİYE TCP RENO	12
4.DENEYSEL SONUÇLARI ANALİZ VE KARŞILAŞTIRMA	16
5.SONUÇ VE ÖNERİLER	21
KAYNAKLAR DİZİNİ	22
ÖZGEÇMİŞ	24

KISALTMALAR VE TÜRKÇE KARŞILIKLAR

TCP	:	Transmission Control Protocol
CWND	:	Congestion Windows
SSTRESH	:	Slow Start Threshold
SegSize	:	Segment Size
ACK	:	Acknowledgment
SMSS	:	Sender Maximum Segment Size
BSD	:	Berkeley Software Distribution
RTT	:	Round Trip Time
RTO	:	Retransmission Time Out
AIMD	:	Additive Increase Multiplicative Decrease
FTP	:	File Transfer Protocol
HTTP	:	Hyper Text Transfer Protocol

Slow start	:	Yavaş Başlangıç
Congestion avoidance	:	Tıkanıklık önleme
Fast retransmission	:	Hızlı yeniden İletim
Fast recovery	:	Hızlı geri dönüş
Segment	:	Bölüt
Transmission window	:	Geçiş penceresi
Advertised window	:	İlan penceresi
Congestion window	:	Tıkanıklık penceresi
Bottleneck	:	Darboğaz
Buffer overflow	:	Tampon bellek taşması
Duplike	:	Çift
Timeout	:	Zamanaşımı
Additive Increase	:	Ek artış
Binary Search Increase	:	İkili Arama Artışı
Segsize	:	Bölüt büyüklüğü

ŞEKİLLER DİZİNİ

Sekil:

Sayfa:

1- Yavaş başlangıç algoritmasının çalışması	6
2- Modifiye Reno Algoritmasının TCP Reno ve TCP Tahoe ile Karşılaştırması (2 MSS değeri için)	12
3- Modifiye Reno Algoritmasının TCP Reno ve TCP Tahoe ile Karşılaştırması (2 MSS ve 2 ACK değeri için)	13
4- Modifiye Reno TCP parametreleri	14
5- Genel ağ haritasının görünümü	16
6- Konya Alt ağı	17
7- Ankara Alt ağı	17
8-Tıkanıklık penceresi boyutunun zaman aralığındaki artış grafiği	17
9- Tahoe ve Reno sürümlerinde cwnd'nin zaman aralığında değişimi	18
10- Modifiye Reno TCP, TCP Tahoe ve TCP Reno için cwnd büyüklükleri değişimi	19

TABLÖLAR DİZİNİ**Tablo :****Sayfa:**

1- Performans Karşılaştırması

19

1. GİRİŞ

TCP, güvenilir bağlantı temelli, uçtan uca bir protokoldür. Alıcının paketleri doğru bir şekilde aldığını belirten bir el sıkışma mekanizmasını içinde barındırır. TCP günümüzde en çok kullanılan protokollerden biridir. Ancak, kullandığımız ağlarda, ağ protokolü istenildiği kadar güvenilir olmamaktadır. Çünkü, bazı nedenlerle paket kayıpları yaşanmaktadır. Örneğin, ağda bir hata meydana gelmesi, ağda tıkanıklık oluşması ve yönlendiricilerin paketleri düşürmesi gibi sebepler söylenebilir. Burada TCP'nin fonksiyonları önem kazanmaktadır. TCP bir paket kaybı oluşması durumunda buna tepki verir ve oluşan hatayı düzeltmeye çalışır. TCP bir paket yolladığında arkasından bir zamanlayıcıyı da beraberinde başlatır. Eğer bir *ACK* zaman aşımı periyodu içerisinde dönmemiş ise o zaman o bölüti yeniden gönderir [1].

Şu an kullanılan iletişim protokolleri yeni uygulama alanlarında ve yeni ağ teknolojilerinde kullanıldığı zaman bir takım kısıtlamalarla karşılaşmaktadır. Örneğin, multimedya uygulamaları tıkanıklık kontrolüne ihtiyaç duyarlar ancak gerekli bir şekilde düzenleme ve güvenilir bir dağıtım da isterler. Bunu ne standart TCP ne de standart UDP sağlamaz. Bu nedenle performansı artırıcı yeni tasarımların oluşturulmasına ihtiyaç duyulur. İşte bu noktadan yola çıkarak problemler incelenip yeni sürümler için fikirler ortaya çıkmaktadır. En büyük problem olan paket kayıplarının olduğu ortamlarda, TCP performansını geliştirmeye yönelik çalışmalar yapılmıştır. Standart TCP, düşük bant genişliği ve düşük gecikme durumlarında işe yaramıştır ancak günümüzdeki şartlarda yeterli olması mümkün değildir [2].

Son yıllarda TCP'nin yüksek hızlı ve uzun mesafeli ağlarda kullanımı, ağ araştırmaları gruplarını, TCP'yi geliştirmeye sevk etmiştir. Çünkü, TCP, internette en çok kullanılan protokoldür ve bant genişliği gecikme çarpanı her geçen gün artmaktadır. Bant genişliği gecikme çarpanı, bant genişliği ile gecikme süresinin çarpımından oluşur. Buradan elde edilen rakam gönderilen ancak alınmayan yani ortamda bulunan veri boyutudur. Bazen “veri bağlantısının kapasitesinin RTT ile çarpımıdır” diye de söylenir. Bu çarpanı kontrol eden AIMD (Additive Increase Multiplicative Decrease) mekanizması, gönderme oranının kararlı hale gelmesi açısından, uçtan uca gerekli olan bant genişliğini algılamakta çok yavaş kalmaktadır.

Bu nedenle yüksek hızlı ve uzun mesafeli ağlar için farklı sürümler geliştirilmiştir. Bu sürümlerde yüksek bant genişliğinden dolayı oluşan gecikme ve zaman kayıpları en aza indirilmiştir [3].

Kablosuz bağlantılarda ile ilgili yapılan çalışmalarda, kablosuz ağlardaki paket kayıpları, çoğunlukla kablosuz kanaldaki bozulmalardan kaynaklanan bit hatalarından, ikinci olarak hareketliliğe bağlı olarak kanalı yakalayamamaktan ve son olarak da sık olmasa da tıkanıklıktan dolayı oluşmaktadır. Daha önce de belirtildiği gibi TCP paket kayıplarını tıkanıklık durumunda algılamaktadır. Bu nedenle bit hataları ve kanalı yakalayamama gibi durumlarda TCP yeterli olamamakta ve TCP'ye bağlı çalışan uygulamalarda sıkıntılar yaşanmaktadır.

Ayrıca kablosuz ağlarda yaşanan diğer bazı problemler ise, düşük sinyal nedeniyle veri iletişimini sağlayamamak ve uzun süreli gürültü oluşmasıdır. Bu durumlarda oluşan bir paket kaybı, TCP tarafından bir tıkanıklık olarak algılanmakta ve hemen yavaş başlangıç algoritması başlatılmaktadır.

Ortamdaki engellere bağlı olarak RTT süresi çok uzun olabilmekte ve zamanaşımı mekanizmaları düzgün çalışmamaktadır. Bu nedenle kablosuz ağlarda kullanılmak üzere Snoop TCP gibi kablosuz ağlarda kullanılan TCP çözümleri üretilmiştir. Ancak bu çözümler tıkanıklık üzerine çözümler değil daha çok hareketlilik halinde hücre yakalamak ve bit hatalarını gidermek amacıyla yapılmış çalışmalardır [4].

Yapılan çalışmalarda belirtildiği gibi TCP'de esas problem, oluşan bir darboğaz durumunda uygulanması gereken algoritmalarıdır. **TCP Tahoe**'deki problem, bir paket kaybını algılamak için geçen süre tam bir zaman aşımı süresidir ve birçok gerçeğe bu büyük bir zamanaşımı süresi gerektirir. Ayrıca acil ACK göndermediğinden, her defasında bir zamanaşımı beklenir ve hat boşaltılır. Bu durum yüksek bant genişlikli ürün bağlantılarında zaman açısından yüksek bir maliyet oluşturur. Bu maliyetin anlamı bir penceredeki kaybolan paketlerin her birisi için tekrar yeniden gönderim gerçekleşir ve çok fazla bekleme gerektirir. Ayrıca hat boşuna meşgul edilmiş olur.

TCP Reno paket kaybı az olduğu durumlarda TCP üzerinde mükemmel çalışmaktadır. Ancak bir penceredeki paket kaybı birden fazla olursa o zaman Reno çok iyi performans gösterememektedir. Eğer birden fazla paket kaybı varsa, ilk kaybolan paket için çift ACK'lar alınır. Sonra kaybolan ikinci paket için bilgi ise ancak bir RTT sonra, yeniden iletilen ilk paketin ACK'sından sonra alınır. Yani bu durumda bir penceredeki iki kayıptan dolayı **cwnd** iki kez yarıya indirilmiş ve düşük değerden başlanmasına neden olmuştur. Bu sebeple hızlı yeniden iletim ve hızlı geri dönüş algoritmaları daha geç başlatılmış olmaktadır. Bu da tıkanıklığın giderilmesi ve iletimin tamamlanması için zaman kaybına neden olur.

Bu çalışmada öncelikle tıkanıklık kontrolü incelenecek daha sonra bu algoritmaların meydana getirdiği TCP versiyonları ele alınacaktır. Bu nedenle yavaş başlangıç, tıkanıklık önleme ve hızlı yeniden iletim algoritmaları TCP Tahoe bölümü içerisinde incelenmiş, hızlı geri dönüş bölümü TCP Reno'yu beraberinde getirdiği için bu sürümde ele alınmıştır. Son olarak da TCP Tahoe ve TCP Reno protokollerinde yaşanan problemlere bir çözüm önerisi sunulmuş ve modifiye bir TCP Reno protokolü oluşturulmuş ve simüle edilmiştir. Bu TCP Reno protokolü yeni bir sürüm değil sadece var olan sürüm üzerinde iyileştirme yapılmış bir protokoldür.

2. Tıkanıklık kontrolü ve ilgili çalışmalar

Tıkanıklık oluşması büyük bir veri akışının olduğu hattan verinin gelmesi ve küçük bir veri akışının olduğu hattan da çıkışın yapılması durumunda meydana gelir. Tıkanıklık ayrıca bir yönlendiriciye çıkış kapasitesinden fazla giriş verisinin gelmesinden dolayı da oluşur. Bir göndericide iki tür paket kaybı oluşur: İlki bir zamanaşımı oluşmasından, ikincisi ise çift *ACK*'ların alınmasıyla.

Bununla beraber, bir hasardan dolayı oluşan paket kaybı oranı çok küçük olarak hesaplanmıştır. Burada hasardan kasıt, ortamdaki kablolarda oluşacak kesme, engel koyma ve aşınma gibi somut hasarlar veya elektromanyetik veya doğal bir titreşimden dolayı oluşabilecek geçici hasarlardır. Ancak bu nedenlerle paket kaybı yaşanması çok düşüktür. Ancak, daha önce de tarif edilen nedenlerle birlikte bir paket kaybının görülmesi, kaynak ile hedef arasında bir tıkanıklık olduğunu işaret etmektedir.

İnternette kullanılan TCP'deki AIMD algoritması paket kaybını bir ağ tıkanıklığı olarak algılamakta ve paket gönderim oranını azaltarak bir reaksiyon göstermektedir. TCP'nin uygun çalışma ortamı sağlanarak bu kayıp oranını azaltılabileceği ve yeni sürümlerle bunu düzeltebileceği görülmüştür [5].

TCP, gönderdiği verinin *ACK*'sını doğru bir şekilde alırsa o zaman gönderim hızını artırır [6]. Diğer yandan, doğru bir *ACK* almadığında yani çift *ACK* aldığı anda ise, bir tıkanıklık algoritması çalıştırması gereklidir. Ancak bu çalıştırılacak algoritmanın doğru seçilmesi asıl problemdir. Bununla ilgili yapılan bir çalışmada, tıkanıklık penceresi büyük olduğu durumlarda, *RTT*'nin büyüklüğünü, yapılacak ek artışın belirlenmesi ve tıkanıklık penceresi küçük olduğunda da ikili arama artışının belirlenmesi gerektiği düşünülmüştür [7].

Kirov, yaptığı çalışmada tıkanıklık kontrolünün yapılabilmesi için kesin olmayan bazı giriş parametrelerinin tahmin edilebilmesi gerektiğini belirtmektedir. Bunlar; ağı kullanan kullanıcı ve uygulama sayısı, ağ kapasitesi, tıkanıklık noktaları gibi parametrelerdir. Hızla büyüyen bant genişlikleri ve çok farklı yazılım uygulamaları TCP'nin akış kontrol mekanizmalarına dikkati daha da artırmıştır. Ağ

kullanımını en uygun şekilde yönetmek amacıyla araştırmacılar, TCP akış kontrol penceresi yönetimine yoğunlaşmışlardır [8].

Jacobson, yaptığı çalışmada, yavaş başlangıç algoritmasını bir dengeleme unsuru olarak belirtmiş, RTT'nin ise bu dengeyi korumak için bir araç olduğunu belirtmiştir. Yolun benimsenmesini ise tıkanıklık önleme olarak ifade etmiştir. Ayrıca hızlı geri gönderimin çok önemli olduğunu ve bu konuda çalışılması gerektiğini belirtmiştir [9].

Tıkanıklık kontrolündeki asıl ilke, göndericinin geçiş penceresinin, hedef alıcıda ve orta sınıf sunucularda, tampon bellek taşması oluşmadan ayarlanmasıdır. TCP bunun için tıkanıklık penceresi adlı bir değişken kullanır. Burada istenen, bir TCP bağlantısında oluşan bir darboğaz durumunda, tıkanıklık penceresinin ayarlanması ve geçiş penceresinin, ilan penceresine ve tıkanıklık penceresine göre seçilmesidir [10]. Bölüm 2.1'de bu algoritmaların kullanıldığı TCP Tahoe incelenmektedir.

2.1.TCP Tahoe

TCP paketlerin bütünlüğünün korunması ilkesine göre çalışır. Buna göre eğer mümkün bir bant genişliğinde bir bağlantı sağlanmış ise burada gönderilen paketin alındı mesajı gelmeden yeni bir paket gönderilmez. Bu sayede TCP, alıcı tarafından paketler alındıkça zamanlayıcılarını buna göre ayarlayacaktır. Ayrıca TCP ağ kapasitesini yansıtan bir tıkanıklık penceresini de kullanır. Bölüm 2.1.1'de TCP Tahoe'de yer alan algoritmaları incelenmiştir.

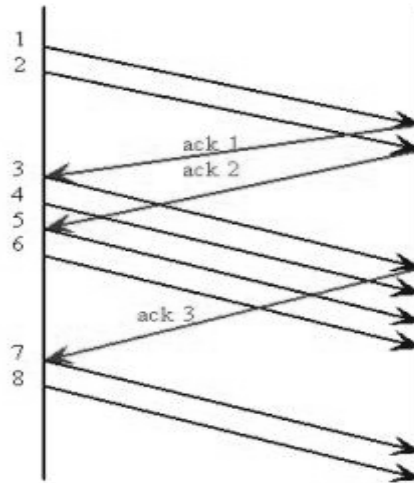
2.1.1.Yavaş başlangıç ve tıkanıklık önleme algoritmaları

TCP sürümlerinin gelişim sürecinde öncelikli olarak bu iki algoritma yer almaktadır. Buna göre bir veri iletiminde işlemler şu şekilde oluşur; TCP paket geçişleri gelen *ACK*'lara göre zamanlanır. Buna göre bağlantı ilk başladığında veri göndermek ve veri almak için *ACK*'lar gerekli olmaktadır.

İşte bu durumda **TCP Tahoe**, bir paket kaybından sonra bir bağlantının başlaması veya yeniden başlatılması için yavaş başlangıç algoritmasını başlatır. Bu

algoritmayla başlamasının sebebi, ağı zorlayan bir başlangıç patlaması yapılması ve bağlantının hiç başlatılamaması gibi bir durumun engellenmek istenmesidir. Yavaş başlangıç **cwnd**'yi 1 olarak ayarlar ve alınan her *ACK* ile **cwnd**'yi artırır. Bu artış eksponansiyel olarak bir kayıp yaşanana kadar devam eder. Bir tıkanıklık durumunda ise gönderim hızı yarıya indirilir ve bu değer **ssthresh** değeri olarak saklanır, daha sonra **cwnd** 1 olarak ayarlanır ve yeniden başlanır. **cwnd** değeri **ssthresh** değerine eşit olana kadar devam edilir. Bu süreç **yavaş başlangıç** olarak adlandırılır [11].

Bir ağda veri iletimine başlamadan önce TCP, ağı derinlemesine bir incelemeye alır ve uygun olmayan veri parçalarının ağda yayılmasına engel olur. Yavaş başlangıç algoritması göndericinin TCP paketine başka bir pencere daha ekler. Bunun adı tıkanıklık penceresidir, kısaca **cwnd** denilir. Başka bir ağdaki istemciyle yeni bir bağlantı kurulmak istendiğinde tıkanıklık penceresi bir bölüte indirilir. Bu bölüt, genelde 536 byte veya 512 byte olarak başlatılır.



Şekil 1 Yavaş başlangıç algoritmasının çalışması

Şekil 1’de görüldüğü gibi, gönderici bir bölütü göndererek başlar ve onun *ACK*’sını bekler. Bu *ACK* alındığında ise, bu tıkanıklık penceresi 1 den 2 ye çıkartılır ve bu iki bölüt birlikte gönderilir. Bu iki bölüt karşılandığında yani *ACK*’sı döndüğünde, tıkanıklık penceresi 4 e çıkarılır. Bu katlanan bir artış sağlar ancak bu bazen katlanan bir artış olmayabilir. Çünkü alıcı bazen *ACK*’ları geciktirebilir. Bunu

da iki bölüt aldığında bir *ACK* göndererek yapar. Gönderici tıkanıklık penceresinin veya ilan edilen pencere büyüklüğünün minimum değeri kadar iletim yapabilir. Tıkanıklık penceresi büyüklüğü gönderici tarafından yapılan akış kontrolü ile düzenlenir. İlan edilen pencere büyüklüğü ise alıcı tarafından düzenlenir.

Aynı zamanda internet bant genişliği kapasitesi aşılabılır ve ortadaki bulunan yönlendiriciler tarafından paketler düşürülebilir. Bu durum alıcıya tıkanıklık penceresinin çok büyük olduğunun sinyalinı verir.

Tıkanıklık önleme kaybolan paketlerle ilgili kullanılabilecek bir yöntemdir. *TCP Tahoe* tıkanıklık önleme için “Additive Increase Multiplicative Decrease” kullanır. Burada bir paket kaybı tıkanıklık işareti olarak alınır ve geçerli pencere boyutunun yarısını bir eşik değeri olarak alınır ve kaydedilir. **cwnd** 1 olarak ayarlanır ve o eşik değerine ulaşana kadar yavaş başlangıç sürdürülür. Daha sonra bir paket kaybı oluşana kadar doğrusal olarak artar. Bant genişliği seviyesine kadar bu artışı sürdürür.

Ayrıca tıkanıklık önleme ve yavaş başlangıç algoritmaları, birbirinden bağımsız algoritmalarıdır, ancak ikisi birlikte gerçekleştirirler. Ne zaman bir tıkanıklık meydana gelse TCP iletim hızını yavaşlatmalı ve yavaş başlangıç algoritmasını başlatmalıdır [12,13,14].

Tıkanıklık önleme ve yavaş başlangıç algoritmaları her bağlantı için iki tane değişkene ihtiyaç duyar:

- Bir tıkanıklık penceresi (**cwnd**)
- Bir yavaş başlangıç eşik değeri (**ssthresh**)

Bu birleşen algoritma aşağıdaki gibi çalışmaktadır:

- a) Yapılan bu bağlantıda, **cwnd** değeri 1 bölüt ve ssthresh değeri 65535 byte olarak başlatılmıştır. **Cwnd** nin başlangıç değeri pratikte **2*SMSS** byte değerine eşit veya küçük olmalıdır ve 2 bölütten büyük olmaması öngörülür. **SMSS** göndericinin iletebileceği en büyük bölüt büyüklüğüdür. **Cwnd** ‘nin

başlangıç değeri keyfi olarak yüksek tutulabilir ama tıkanıklığa bağlı olarak düşürülmelidir.

- b) TCP çıkış rutini, **cwnd**'nin minimum değeri veya alıcının ilan edilen pencere büyüklüğü değerinden daha fazla ileti göndermez.
- c) Tıkanıklık oluştuğunda geçerli pencere değerinin yarısı, **ssthresh** değerinde saklanır. Ek olarak, eğer tıkanıklık bir zamanaşımı süresinde belirlenirse, **cwnd** 1 bölüt olarak ayarlanır ve **ssthresh** değerine gelinceye kadar bu süreç devam eder.
- d) Eğer yeni bir veri diğer tarafta karşılanırsa o zaman **cwnd** artırılır. **Cwnd** 'nin artma şekli, TCP'nin yavaş başlangıç veya tıkanıklık önleme gerçekleştiriyor olmasına bağlı olarak gerçekleşir. Eğer **cwnd**, **ssthresh**'e eşit veya az ise, TCP yavaş başlangıç modundadır, diğer durumda tıkanıklık önleme algoritması uygulanmaktadır. Yavaş başlangıç, TCP tıkanıklık başladığında bulunduğu değerin yarısına geldiği zamana kadar devam eder, daha sonra tıkanıklık önleme devreye girer. Bu kaydedilen pencere büyüklüğünün değerine göre yapılır.

Daha önce de belirttiğimiz gibi yavaş başlangıç tıkanıklık penceresini katlayarak artırmaktadır. Tıkanıklık önleme diğer bir taraftan tıkanıklık penceresinin(**cwnd**) her *ACK* alındığında $\text{segsz} * \text{segsz} / \text{cwnd}$ şeklinde artmasına zorlamaktadır. Burada **segsz**, segment büyüklüğü ve **cwnd** de byte ların büyüklüğünü ifade eder.

Bu sonuçlar **cwnd**'nin doğrusal bir artış yapmasına ve yavaş başlangıç'ında katlanarak büyümesine neden olur. **Cwnd**'deki bu artış, bir RTT'de alınan her *ACK* değeri kadar artışla değil her ortalama dönüş süresinde **cwnd**'nin 1 bölüt artışına neden olur. Bölüm 4'te bu algoritmalarla ilgili olarak bir OPNET benzetimi yapılmıştır ve sonuçlar Bölüm 4'te verilmiştir.

2.1.3. Hızlı yeniden iletim

Hızlı yeniden iletim tıkanıklık önleme algoritmasının bir modifikasyonudur. TCP göndericisi hızlı yeniden iletim algoritmasını gelen çift *ACK*'ları algılama ve kayıpları tamir etmek için kullanacaktır. Hızlı yeniden iletim algoritması 3 tane çift

ACK gelmesi durumunda bunu bir segmentin kaybolduğunu işareti olarak algılar (gelen diğer paketlerden hariç 4 tanımlı *ACK*'nın gelmesi). 3 tane çift *ACK*'nın alınmasından sonra, TCP kaybolan bölütü tekrar iletmek üzere hatta yollar ve burada herhangi bir zamanlayıcının bitmesini beklemez. Bu algoritma(Hızlı yeniden iletim) ilk defa 4.3BSD Tahoe sürümünde ortaya çıkmıştır [15,16].

TCP Tahoe sürümünde ortaya çıkan problem, bir paket kaybını algılayabilmek için tam bir zamanaşımı periyodu geçmelidir ve bu nedenle birçok gerçekleştirme çok uzun zaman almaktadır. Ayrıca, hızlı *ACK*'lar gönderemediği ve “**go back n**” yaklaşımı kullandığı için her defasında, paket kaybını anlamak için bir zamanaşımı beklemesi ve hattın boşaltılması gerekmektedir. Bu durum, yüksek bant genişliğinde büyük bir maliyet oluşturmaktadır.

2.2.TCP Reno

TCP Reno, TCP Tahoe'nin bazı özelliklerini, örneğin yavaş başlangıç ve hızlı yeniden iletim algoritmasının zaman aşımı süresinin uzun olması gibi, muhafaza etmektedir. Bununla birlikte bazı özellikleri de beraberinde getirmektedir. Kayıp paketler daha erken fark edilmekte ve hat her defasında boşaltılmamaktadır [17,18].

TCP Reno, bir bölüt alındığında acil bir *ACK* göndermektedir. Ne zaman bir çift *ACK* alınsa yeni bir dizi beklenmekte veya ağda geciken ve bozulan olan bölütler bulunmakta veya paket kaybedilmiş bir paket olduğu düşünülmektedir. Eğer birden fazla çift *ACK* alınıyorsa, o zaman yeterli zamanın geçtiği ve ancak uzun bir yol alındığında alıcıya ulaşacağı anlaşılır. Paketin kaybolması yüksek bir olasılıktır.

Bu durumda hızlı yeniden iletim algoritması devreye girer. Ardarda 3 çift *ACK* alındığında bu paket kaybına işarettir ve zamanaşımı beklenmeden bir geri gönderim başlamalıdır. Bir paket kaybı durumunda Reno'da yapılan diğer bir değişiklik ise **cwnd**'nin 1 olarak ayarlanmasıdır. Böylece hat boşaltılır. Bundan sonra hızlı geri dönüş devreye girer [19].

2.2.1.Hızlı geri dönüş

Tıkanıklık önleme algoritması yavaş başlangıç olmadan hızlı yeniden iletim algoritmasının gönderdiği kaybolan segmentin ne olduğunu göstermesiyle başlatılır. Bu algoritma çok aşırı olmayan bir tıkanıklık durumunda yüksek çıkışta veri iletimini özellikle yüksek pencere büyüklükleri için sağlamaktadır.

Bu durumda, çift *ACK*'ların alınmasıyla yavaş başlangıç'ın başlatılamaması TCP'ye birden fazla paketin kaybolduğu bilgisini verir. Alıcı çift bir *ACK*'yı ancak diğer bir segment alındığında gönderebilir ki bu segment ağdan çıkmış ve alıcının buffer'ına girmiş demektir. Diğer bir deyişle, her iki uç arasında bir veri iletimi sağlanmaktadır ancak TCP akışı yavaş başlangıç algoritmasını çalıştırarak aniden düşürmek istememektedir. Hızlı geri dönüş algoritması da bu durumlar için ilk defa 4.3BSD Reno sürümünde ortaya çıkmıştır.

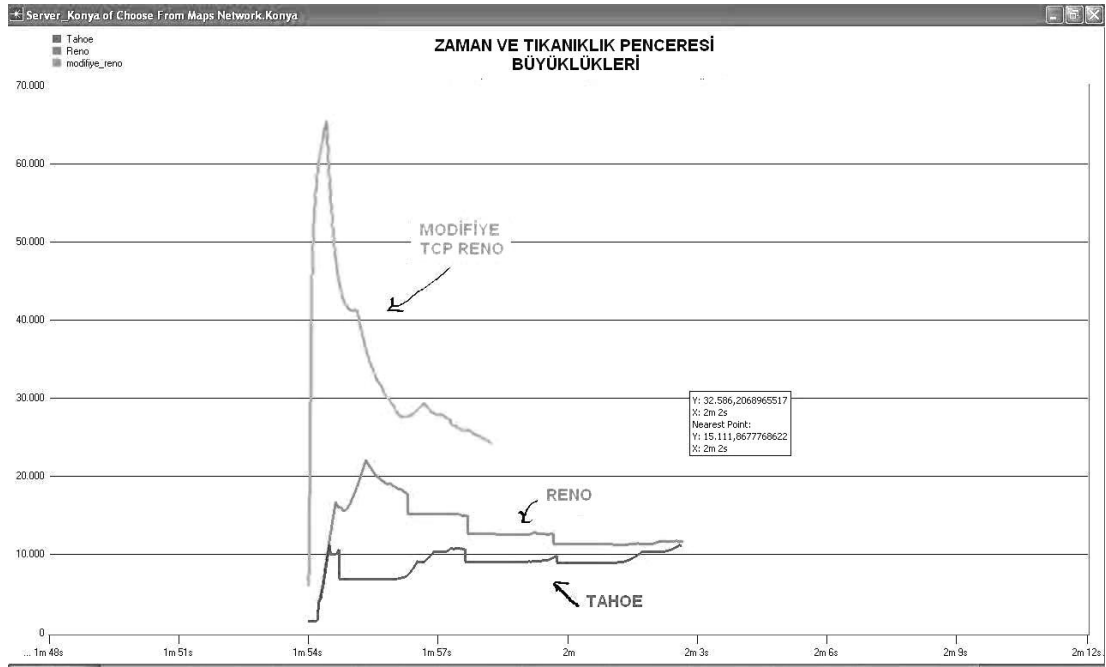
Hızlı yeniden iletim ve hızlı geri dönüş algoritmaları aşağıdaki durumlarda gerçekleşirler:

- 1- Üçüncü bir çift *ACK* alındığında, **ssthresh** değerini **cwnd** değerinin yarısına ayarlanır ancak bu değer iki bölütten az olamaz. Kayıp bölüt tekrar iletilir ve **cwnd**, **ssthresh** artı üç kat bölüt büyüklüğü değerine ayarlanır. Bu tıkanıklık penceresinin ağdan çıkarak alıcının ön bellek hafızasına artarak gitmesini sağlar.
- 2- Her zaman bir diğer çift *ACK* alındığında, **cwnd** değeri bölüt büyüklüğü kadar artırılır. Bu tıkanıklık penceresi değerini ağdan ayrılan her bölütte biraz daha şişirir.
- 3- **Cwnd** ve ilan edilen pencere büyüklüğü değeri izin verdiği müddetçe paket iletimine devam edilir.
- 4- Yeni bir veriyi karşılayan bir *ACK* alındığında, **cwnd** ve **ssthresh** set edilir. Bu *ACK* ile tekrar iletimden sonraki bir RTT süresinde 1. maddedeki adımları tekrar eder. Ayrıca, bu *ACK* ilk çift *ACK* ile ilk kayıp paket arasındaki bölütlerinde karşılanmasını sağlar. Bu durumda tıkanıklık önleme devreye girer ki burada TCP paket kaybolduğundaki değerinin yarı değerine iner.

Bölüm 4’te hızlı yeniden iletim ve hızlı geri dönüş gerçeklemeleri benzetim şartlarında uygulanmıştır.

kaybı olduğu düşünülürse, o zaman gereksiz yere tıkanıklık algoritması başlatılmış olmaktadır. Bu durumda da yine zaman kaybı yaşanacaktır.

Bu düşünceden yola çıkarak yapılan değişiklik, hızlı yeniden iletim algoritmasının daha erken start olması için gerekli çift ACK sayısı 3'den ikiye düşürülmesidir. Bu değişikliği yaptığımız benzetim üzerinde uyguladığımızda dezavantaj olarak nitelediğimiz tıkanıklık algoritmasının gereksiz yere başlatılması durumundan fazla etkilenilmemiştir ve gelişme kaydedilmiştir. Bu değişiklik neticesinde, şekilde görüldüğü gibi TCP Reno ile Modifiye TCP Reno algoritması daha çok ayrışması sağlandı.



Şekil 3- Modifiye Reno Algoritmasının TCP Reno ve TCP Tahoe ile Karşılaştırması (2 MSS ve 2 ACK değerleri için)

Eğer MSS değeri daha yüksek bir değerden başlatılırsa o zaman eksponansiyel artışın daha hızlı olacağı düşünülmektedir. Buna göre diğer yapılan değişiklikte, eşik değeri olarak en yüksek bölüt büyüklüğü, 1 MSS'den 4 MSS'ye yükseltildi. Bu artışın avantajı, her paket kaybı olduğunda 1 MSS yerine 4 MSS değerinden eşik değeri başlatılarak eksponansiyel artışın daha hızlı olması sağlamasıdır. Dezavantajı ise eğer paket kaybı birden fazla istemcide meydana gelirse o zaman yönlendiricide yükün artacağı düşünülmektedir.

Bunu daha açmak istersek sistemde paket kayıplarının çok yüksek olduğu durumda tüm istemcilerin paketleri sırayla eşik değeri olan 4 MSS değerinden başlatılırsa, o zaman bu yine darboğaz oluşumunu engelleyemeyecek daha fazla zaman kaybına neden olabilecektir. Ancak yaptığımız benzetimde böyle bir durumla karşılaşılmamıştır.

Bu gelişmeler dikkate alınarak hızlı yeniden iletim için gerekli çift ACK sayısının en uygun seviyesi 2 ACK olarak belirlendi. Yavaş başlangıç algoritması başlangıç değeri ise 4 MSS seçildi.

Modifiye TCP Reno'nun kullanılmasıyla hem hızlı yeniden iletim hem de hızlı geri dönüş algoritmaları birlikte kullanabilmekte ve istenilen performans için modifikasyonlar rahatça yapılabilmektedir. Burada yapılan modifikasyonları özetlenirse:

- 1- Hızlı yeniden iletim algoritmasının daha erken başlamasını sağlamak için gerekli çift ACK'ların sayısını 3'ten 2 ye düşürülmüştür. Bu sayede gecikme durumunda bekleme süresini kısaltılmış ve hızlı yeniden iletim algoritması erken başlayarak tıkanıklık kısa sürede giderilmiştir.
- 2- Yavaş başlangıç algoritmasının başlama noktası 1 MSS(Maximum Segment Size) değil 4 MSS olarak belirlendi. Bu sayede hem hızlı geri dönüş algoritmasının çalışması kolaylaşmış oldu hem de **cwnd**'nin tüm hattı kapatması engellendi.

Attribute	Value
Slow-Start Initial Count (MSS)	4
Fast Retransmit	Enabled
Duplicate ACK Threshold	2
Fast Recovery	Reno
Window Scaling	Disabled
Selective ACK (SACK)	Disabled
ECN Capability	Disabled
Segment Send Threshold	Byte Boundary
Active Connection Threshold	Unlimited
Nagle Algorithm	Disabled
Karn's Algorithm	Enabled
Timestamp	Disabled
Initial Sequence Number	Auto Compute

Şekil 4 - Modifiye Reno TCP parametreleri

Burada, yavaş başlangıç algoritması başlama eşiği (MSS) değeri 4'e yükseltilmiş, çift ACK eşik değeri 2'ye düşürülmüştür. Buna göre, meydana gelen performans artışı, aşağıdaki tablo 1'de belirtilmiştir.

Burada kullanılan ağda kullanılan alt ağ sayısı 2 den 7'ye çıkarılmış ve paket gönderim oranı ve performansta çok büyük bir fark görülmemiştir. Ayrıca trafikte meydana gelen artış belli bir seviyeye kadar artmış sonra dengelenip sabitlenmiştir. Gönderilen paketlerin ACK sayısı, artan alt ağ miktarı kadar bir artış göstermiştir. Yeniden gönderim miktarı da alt ağ miktarına paralel bir artış içerisine girmiştir.

Genel olarak bakıldığında yapılan çalışmada yapılan değişikliklerle performans artışı sağlanması amaçlanmıştır. Modifiye TCP Reno algoritmasında düşünülen değişiklikler 2 ACK ve 4 MSS değerleri simüle ettiğimiz ağda iyi sonuç vermiştir.

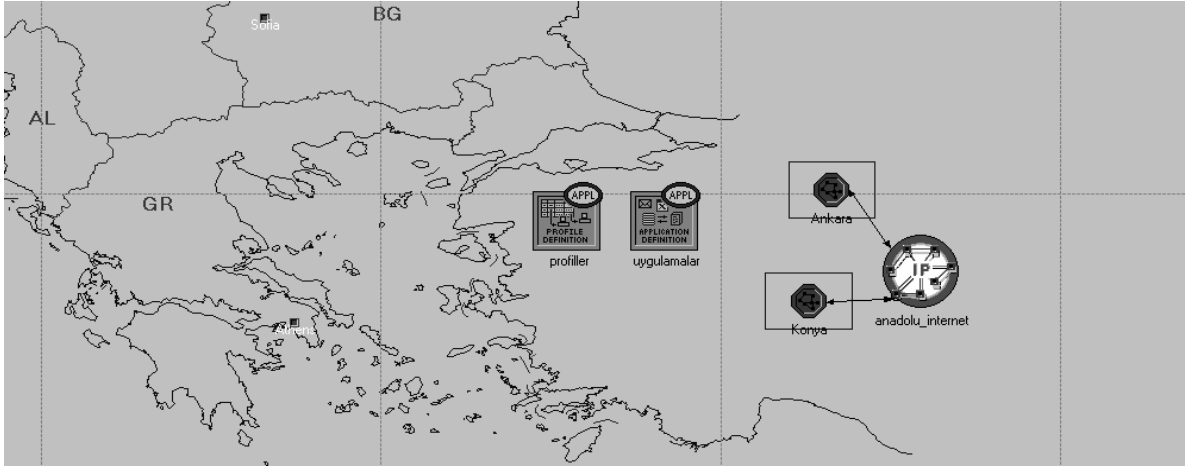
4.Deneysel sonuçları analiz ve karşılaştırma

Bu bölümde önce yavaş başlangıç ve tıkanıklık önleme benzetimi ile hızlı yeniden iletim ve hızlı geri dönüş gerçeklemeleri sonuçlarıyla birlikte anlatılmıştır. Daha sonra modifiye Reno TCP algoritmasının benzetim sonuçları analiz edilmiş ve karşılaştırılmıştır.

Yavaş başlangıç ve tıkanıklık önleme benzetiminde, ağ üzerinde yapılan konfigürasyon ile TCP bir uçtan uca iletişim protokolü olarak çalıştırıldı. Bir sunucu Konya’da, bir sunucu da Ankara’da yer almaktadır. Burada tıkanıklık penceresi büyüklüğü farklı mekanizmalarla incelenmiştir.

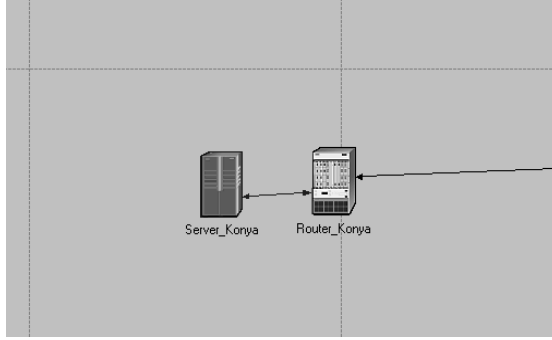
Burada iki tane alt ağ oluşturuldu. Bunlar Konya ve Ankara alt ağlarıdır. Bu iki alt ağ bir sunucu ve bir yönlendiriciden oluşmaktadır. Bu sunucuların her biri yönlendiricilere saniyede 10 megabit tabanlı bant genişliği sağlayan UTP kabloyla bağlıdır. Yönlendiriciler ise internete saniyede 100 megabit tabanlı UTP kabloyla bağlıdır.

Şekil 5,6 ve 7’de bu bağlantı tipleri ayrıntılı olarak gösterilmektedir.



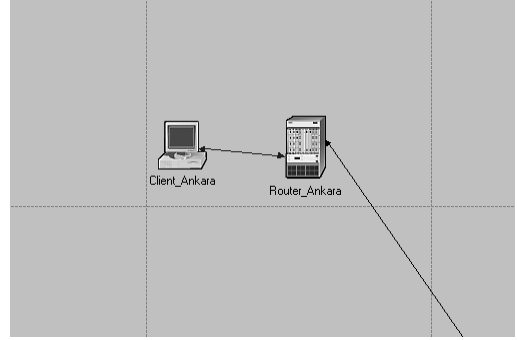
Şekil 5- Genel ağ haritasının görünümü

OPNET’te oluşturulan bu şekilde Ankara ve Konya adlı iki tane alt ağ oluşturulmuş ve bu ağlar internet üzerinden birbirine bağlanmıştır. Uygulama olarak dosya transfer protokolü uygulaması çalıştırılmaktadır.



Şekil 6- Konya Alt ağı

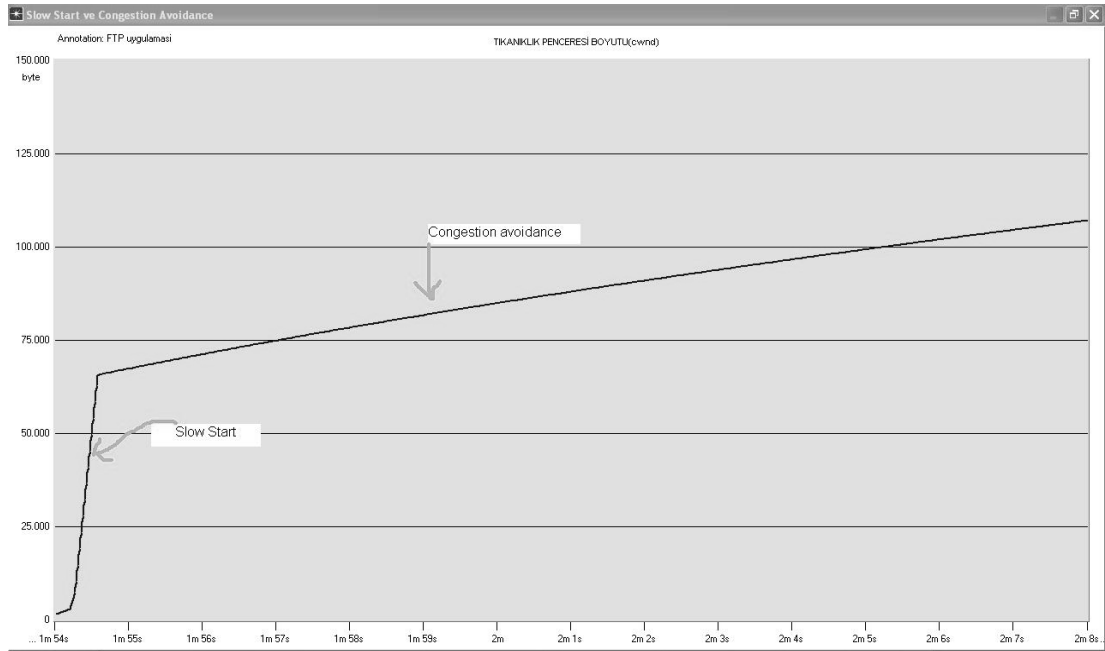
Bir sunucu ve bir yönlendiriciden oluşur.



Şekil 7- Ankara Alt ağı

Bir istemci ve bir yönlendiriciden oluşur.

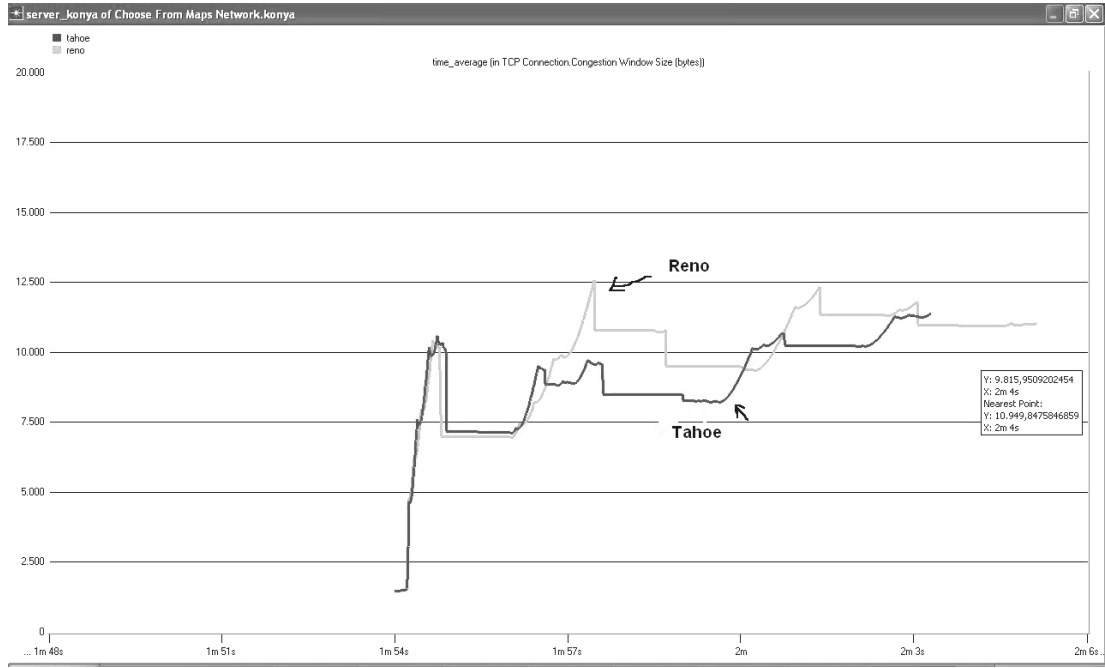
Bu gerçeplemede yapılan bir FTP uygulaması hem Konya alt ağında hem de Ankara alt ağında çalıştırılmış ve bu uygulama internet üzerinden veri alışverişinde bulunmuştur. Bu gerçeplemede yapılan 100000 paketlik veri aktarımı, 10 dakikada gerçeplemiş ve ağına buna göre gerçepleşen grafiğı Şekil 8'deki gibi oluşmuştur.



Şekil 8- Tıkanıklık penceresi boyutunun zaman aralığındaki artış grafiğı (Yavaş başlangıç ve tıkanıklık önleme)

Şekil 8'te görüldüğü gibi, X koordinatı zaman aralığını Y koordinatı ise cwnd boyutunu göstermektedir. Beklendiğı gibi, yavaş başlangıç algoritmasında belli bir eşik değerine kadar yüksek bir artış gözlenmiş, daha sonra tıkanıklık önleme algoritması başlayarak lineer bir artış seyretmiştir.

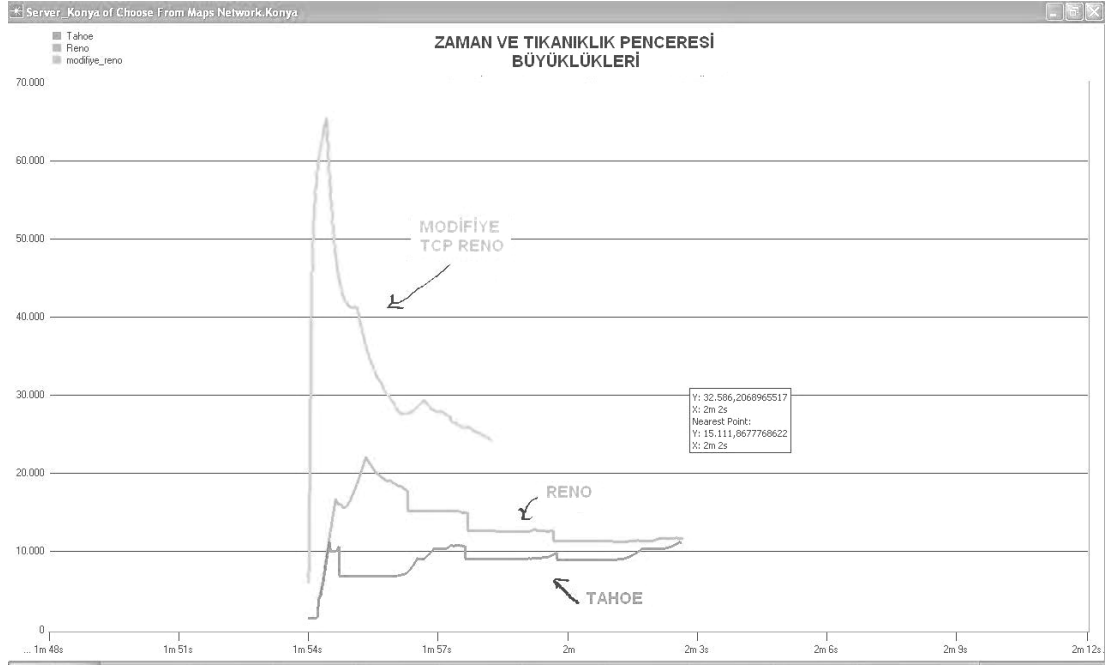
İkinci gereklemede, hızlı geri dnüş ve hızlı yeniden iletim algoritmalarının davranışlarını incelemek için TCP'nin Tahoe ve Reno versiyonlarını karşılaştırdık. Tahoe versiyonunda hızlı yeniden iletim algoritmasını, TCP Reno versiyonunda ise hızlı geri dnüş algoritmasını ekledik.



Şekil 9- Tahoe ve Reno sürümlerinde cwnd'nin zaman aralığında değişimi

Şekil 9'da görüldüğü gibi TCP Reno'nun, TCP Tahoe'den az bir farkla daha iyi performans gösterdiği ancak geç başlangıç yapıp geç sonlandığı gözlenmektedir. Bu çalışmada Tahoe sürümündeki paket kayıp oranı %0.5 olarak set edilmiştir. Bir tıkanıklık bir zamanaşımı tarafından gösterilirse o zaman **cwnd** bir bölüt olarak gösterilir. Kısacası, yavaş başlangıç algoritması çalıştırılır. TCP Reno sürümünde ise TCP Tahoe'ya benzemekle birlikte paket iletim oranı artmıştır. Burada yavaş başlangıç algoritması değil hızlı geri dnüş algoritması çalıştırılır.

Son olarak, bölüm 3'te önerilen değişiklikler uygulandı ve benzetim çalışmaları kaydedilip karşılaştırıldı. Şekil 10'da verilen grafikte Modifiye TCP Reno ve TCP Reno algoritmalarının tıkanıklık penceresi boyutlarının zamana göre değişim grafiğı verilmektedir. Modifiye TCP Reno çok daha yüksek **cwnd** ile iletişim sağlamaktadır. Grafikten aynı zamanda Hızlı yeniden iletim yapmalarına karşılık daha yüksek veri akışı yapıldığı anlaşılmaktadır.



Şekil 10 – Modifiye RenoTCP, TCP Tahoe ve TCP Reno için cwnd büyüklükleri değişimi

Şekilde modifiye TCP Reno algoritmasının diğer TCP versiyonlarına göre boyut açısından daha fazla veri aktarımı sağladığı ve süre açısından daha hızlı bir şekilde veri aktardığı gözlenmektedir. Cwnd büyüklüğü 65000 byte seviyesine ulaşmış ve işlemin bitme süresi TCP Reno ve TCP Tahoe'ye göre büyük ölçüde fark etmiştir.

Ayrıca yeniden gönderim sayısında azalma kaydedilmiş ve karşılanan ACK sayısı daha iyi bir rakama yükselmiştir. Verilen performans karşılaştırması tablosunda kaydedilen gelişme daha iyi anlaşılmaktadır.

	Karşılanan ACK sayısı	Tıkanıklık Süresi	Cwnd büyüklüğü	Yeniden gönderim sayısı
Normal TCP Reno	798	6s	64000	32
Modifiye TCP Reno	854	4,5s	22000	25

Tablo 1- Performans Karşılaştırması

Tablo 1'e göre, modifiye TCP Reno algoritmasında, tıkanıklığın giderilme süresi ve yeniden gönderimlerin azalması sebebiyle görülen performans artışı %20 civarındadır.

Ayrıca bu analizde modifiye TCP Reno algoritmasının daha önce hızlı yeniden iletim algoritmasına girerek akışı hızlandığı görülmektedir. Burada daha büyük MSS değeri kullanarak hızlı geri dönüş ve hızlı yeniden iletim algoritmalarının daha erken başlangıç yapmaları sağlanmıştır. Üçüncü çift ACK beklenilmeden tıkanıklık olduğu varsayıp tıkanıklık algoritmasının başlatılması ile de yolda meydana gelen kayıplar için çok daha az süre beklenilmiştir.

Bu durumda, çalışma yaptığımız ağ üzerindeki benzetimde değişikliklerin iyi sonuç verdiği gözlenmiştir. Ancak bu durum farklı ağlarda aynı neticeyi vereceği anlamına gelmemektedir.

5.Sonuç ve Öneriler

Bu çalışmada, tıkanıklık kontrol mekanizmasını **TCP Reno**'nun modifiye edilmesiyle daha kullanışlı bir hale getirdik. Üretilen senaryolar, TCP akış kontrol algoritmaları arasında karşılaştırmalar yapılmasını ve modifiye bir TCP algoritması üretilmesini sağlamıştır.

Bu çalışma ile farklı TCP algoritmalarının tamamının akışı hızlandırmaya yönelik olduğu ve **cwnd** büyüklüğünün hatta bağlı olarak hangi seviyede iyi sonuç vereceği simüle edildi. Sonuç olarak **TCP Reno** algoritmasının ağ yükünün fazla olmadığı durumlarda en iyi çözüm olduğu ve bu algoritmanın modifiye edilerek iyileştirilmesi ile daha iyi sonuçlar vermesi sağlanmıştır. Gerçeklenen senaryodaki ağda, modifiye TCP Reno algoritmasının normal TCP Reno Algoritmasından daha hızlı çalıştığı gözlemlendi. Sonuç olarak, yeni bir TCP algoritması değil, TCP Reno algoritmasında iyileştirmeler yapılmış oldu.

Bu tez kapsamında, iki TCP sürümü incelenmiş ve bu sürümler iyileştirilerek uygun gerçeklemeler ve analizler yapılmıştır. Bu çalışma baz alınarak ileride yapılabilecek araştırma önerileri şu şekilde sıralanabilir:

- TCP Tahoe ve TCP Reno sürümlerine ek olarak TCP New Reno ve TCP SACK sürümleri de bu çalışmalara dahil edilip TCP'nin tüm ağlarda daha verimli bir şekilde kullanılması sağlanabilir.
- TCP'de meydana gelen tıkanıklıklar için yeni çözüm önerileri ve gelişmeler baz alınarak yeni sürümler ortaya atılabilir.

KAYNAKLAR DİZİNİ

- [1] Kurose J.F,Ross K.W., “Computer Networking”, Pearson Publishing, Third Edition, 2005
- [2] He Eric, “A Survey Of Protocols Other Than “Standart” TCP”, Global Grid Forum, University Of Illinois, 2005
- [3] Mascolo Saverio,Vacirca Francesco, “Issues in Performance Evaluation of New TCP Stacks in High Speed Networks” , INFOCOM Department, Roma ,Italy,2005
- [4] Jacobson V., “Modified TCP Congestion Control And Avoidance Algorithms”, Technical Report,1990
- [5]“TCP Libra: balancing Flows over Heterogeneous propagation Scenarios”, Submission to SigMetrics ,Italy,2006
- [6] Omueti M,Trajkovic L, “OPNET Model of TCP with Adaptive Delay and Loss Response for BroadBand GEO Satellite Networks” ,Simon Fraser University, Vancouver, Canada, 2007
- [7] Xu L.,Harfoush K., Rhee I.,“Binary Increase Congestion Control(BIC) For Fast Long Distance Networks”,N.Carolina State University, 2004
- [8] Kirov George ,“A Simulation Analysis Of The TCP Algorithms”, Bulgarian Academy Of Sciences, Research Paper,2002
- [9] Jacobson V, Karels M.J.,”Congestion Avoidance And Control”, SIGCOMM Symposium On Communication Architecture Protocols,November, 1988
- [10] Kirov George, “A Simulation Study Analysis Of The Tcp Control Algorithms”, International Conference On Computer Systems And Technologies, Compsys Tech,2005
- [11] Khan J, Punnam P., Zaghal R., ”Event Based Extensible Interactive Transparent Networking:Performance Study With Fast TCP Principles”, 21st International Conference On Advanced Networking And Applications ,2007

- [12] Peterson L.L.,Davie B.S., “Computer Networks:A systems Approach”, Morgan Kaufmann Publishing, Third edition, 2003
- [13] Tanenbaum A.S., “Computer Networks”, Prentice Hall Publishing, Transport Layer, Third edition, 2003.
- [14] Floyd S., “HighSpeed TCP for Large Congestion Windows”, RFC 3649, Experimental,2003.
- [15] Hassan M.,Jain R. ,“High Performance TCP/IP Networking”, Pearson Education International,2004
- [16] Stevens W., ,“Tcp Slow Start,congestion Avoidance,Fast Retransmit and Fast Recovery Algorithms”, RFC:2001 , Networking Group, January 1997
- [17] Corral G., Zaballos A., “Simulation Based Study Of TCP Flow Control Mechanisms Using Opnet Modeler”,White paper, 2005.
- [18] Jin C., Wei D., Low H., Bunn J., Choe D., Doyle J., Buhrmaster G., Cottrell L., Paganini F., Newmann H., Feng W., “Fast TCP:From Theory To Experiments” , IEEE Network Journal ,January- February 2005
- [19] Svensson T., Popescu A., “Development of Labaratory Exercises based on OPNET Modeler”, Master Thesis, Blekinge Institute of Technology, 2003.
- [20] Akhtar M.N., Barry M.A.O., Al-Raweshidy H.S., “Modified Tahoe TCP for Wireless Networks Using OPNET Simulator”, Department of Electronic Engineering, Kent University,2003.

ÖZGEÇMİŞ

Ahmet Kemal AYGİN, 1981 yılında Mersin'in Erdemli İlçesinde doğdu. İlkokulu Konya Barbaros Hayrettin Paşa İlkokulunda, Ortaokul ve Liseyi Konya Selçuklu Anadolu Lisesinde tamamladı. 2000 yılında kazandığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümünü 2004 yılında bitirdi. Yine aynı yıl Gebze Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Anabilim dalında Yüksek Lisans Eğitimine başladı. Çalışma hayatına 2005 yılında kısa bir süreliğine Güney Afrika Cumhuriyetinde başlayan yazar, daha sonra Türkiye'ye dönüp Sosyal Güvenlik Kurumunda Bilgisayar Mühendisi olarak göreve başlamıştır. Halen SGK'da Ağ yöneticisi olarak görevine devam eden yazar, Geniş Alan Ağları ve Protokoller üzerinde çalışmalar yapmaktadır.