İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

TELSİZ ORTAMLARDAN GEÇEN TCP BAĞLANTILARI İÇİN YENİ BİR BAŞARIM ARTIRMA TEKNİĞİ

YÜKSEK LİSANS TEZİ

Müh. İlkin Ulaş BALKANAY

Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Programı : BİLGİSAYAR MÜHENDİSLİĞİ

Tez Danışmanı: Prof. Dr. A. Emre HARMANCI

Mayıs 2005

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

TELSİZ ORTAMLARDAN GEÇEN TCP BAĞLANTILARI İÇİN YENİ BİR BAŞARIM ARTIRMA TEKNİĞİ

YÜKSEK LİSANS TEZİ

Müh. İlkin Ulaş BALKANAY
(504021534)

Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Programı : BİLGİSAYAR MÜHENDİSLİĞİ

Tez Danışmanı: Prof. Dr. A. Emre HARMANCI

Mayıs 2005

ÖNSÖZ

Bu çalışmanın hazırlanmasında yardımlarını ve değerli görüşlerini esirgemeyen tez danışmanım Sayın Prof. Dr. A. Emre Harmancı'ya, çalışma hakkındaki değerli yorumlarını bizimle paylaşan Doçent Dr. Albay Erdal Çayırcı'ya ve her durumda beni destekleyen aileme teşekkürlerimi sunarım.

Mayıs 2005 **İlkin Ulaş BALKANAY**

İÇİNDEKİLER

KISALTMALAR

TCP	Transmission Control Protocol
OSI	Open System Interconnection
ITCP	Indirect TCP
ACK	Acknowledgment
DUPACK	Duplicate Acknowledgment
CWND	Congestion Window
AWND	Advertised Window
RTT	Round Trip Time
RTO	Retransmission Timeout
ZWA	Zero Window Advertisement
ZWP	Zero Window Probe
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
GEO	Geostationary Satellite
SACK	Selective Acknowledgment
IPSec	Internet Protocol Security
KA-TCP	Kayıp Ayırt Edici TCP

TABLO LİSTESİ

ŞEKİL LİSTESİ

TELSİZ ORTAMLARDAN GEÇEN TCP BAĞLANTILARI İÇİN YENİ BİR BAŞARIM ARTIRMA TEKNİĞİ

ÖZET

Paket bozulma olasılığı oldukça küçük olan telli ağlar için tasarlanmış olan TCP (Transmission Control Protocol) tıkanıklık denetimi algoritmaları, telsiz ortamlarda kullanıldığında ciddi başarım kayıpları ile karşılaşılır.

İlk geliştirilen TCP tıkanıklık denetimi algoritmaları (TCP Tahoe ve TCP Reno'da kullanılan) segman kaybını tıkanıklık habercisi olarak kabul edip veri iletim debisini kısarak tıkanıklığa karşı önlem almaya çalışırlar. Fakat günümüzde telli ve telsiz ağların beraber kullanıldığı bir haberleşme ağında, segman kayıplarının asıl sebebi tıkanıklık olmayabilir. Hata oranı çok yüksek olan telsiz linkler üzerinde bozulmaya uğrayıp kaybolmuş TCP segmanları için tıkanıklık denetimi algoritmarını çalıştırmak, istenmeyen başarım kayıplarına yol açar.

TCP, kaybolan TCP segmanlarının kaybolma sebeplerini ayırt edebilirse doğru müdahaleyi yapabilir. Tıkanıklık yüzünden yönlendirici kuyruklarında düşürülmüş TCP segmanları için tıkanıklık denetimi algoritmalarını çalıştırır. Telsiz linkler üzerinde bozularak kaybolmuş segmalar için tıkanıklık kontrolü algoritmalarını çalıştırmadan kaybolan segmanı yeniden göndermeyi dener. Böylece gereksiz yere iletim debisini kısılmamış olur.

Bu çalışmada, hedefe ulaşamayan TCP segmanlarının, tıkanıklık yüzünden mi yoksa telsiz link üzerindeki bozucular yüzünden mi kaybolduğunu anlamaya yarayan bir yöntemden bahsediliyor. Segmanların neden kaybolduğunu anladıktan sonra TCP alındı işleme ve veri segmanı işleme algoritmalarında yapılan değişikliklerle veri iletim başarımındaki artış benzetim yoluyla gösteriliyor.

A NEW PERFORMANCE IMPROVEMENT TECHNIQUE FOR TCP CONNECTIONS ON WIRELESS LINKS

SUMMARY

TCP congestion control algorithms are first designed for wired networks where packet losses are mainly due to network congestion. But for networks with wired and wireless lossy links, traditional congestion control algorithms may cause performance degradations.

Traditional TCP congestion control algorithms rely on packet loss as an indicator of network congestion and try to slow down the data sending rate when a TCP segment is lost somewhere in the network.

TCP is unable to determine if a segment loss is due to congestion or corruption of the packet due to errors in the network. Wireless networks are characterized by high bit error rates and periodic disconnectivity. As a result TCP performs poorly in wireless environments as it interprets packet corruption as congestion in the network. Executing congestion control algorithms for corrupted segments on wireless links, results in low throughputs for TCP transfers.

If TCP is able to differentiate between congestion losses and wireless losses, congestion control algorithms will be triggered for segments that are dropped on congested links and retransmission algorithms will be triggered without reducing sending rate for segments that are lost on wireless lossy links.

In this study an explicit loss differentiation mechanism is explained and a new extension of TCP, Loss Discriminating TCP, is introduced. The performance gain in throughput is shown with various simulations.

1. GİRİŞ

Bu tez çalışmasında TCP'nin tıkanıklık denetimi algoritmaları incelenmiş ve telsiz linklerde, TCP tıkanıklık denetimi algoritmalarının yaşadığı problemleri çözmeyi amaçlayan yeni bir yöntem geliştirilmiştir.

Kolaylık olması amacıyla yazının devamında, TCP varlıkları (TCP yazılımı) için ve TCP protokolü (kurallar bütünü) için "TCP" sözcüğü kullanılacaktır. TCP sözcüğünün hangi anlamda kullanıldığı yazı içerisinden anlaşılabilir olacaktır.

TCP yani **T**ransmission **C**ontrol **P**rotocol yedi katmanlı OSI referans modelinin, aktarım katmanında yer alır. TCP iki hostun birbirleriyle güvenilir ve bağlantılı haberleşmesini sağlar.

Bağlantılı haberleşme tarafların (kaynak ve hedef) iletişime geçmeden önce aralarında bir oturum açmasıdır. Oturumun açılması sırasında bilgisayarlar kendi iletişim parametrelerini birbirlerine iletirler ve bu parametreleri dikkate alarak iletişimde bulunurlar.

Güvenilir haberleşmeden kasıt bilginin karşı tarafa doğru bir şekilde bozulmadan gittiğinden emin olmaktır. Bu güvenilirlik, bilginin alındığına dair karşı taraftan gelen bir onay mesajı ile sağlanır. Eğer bilgi gönderildikten belli süre sonra bu mesaj gelmezse segman yeniden gönderilir.

4. katmanda (aktarım katmanı) yer alan TCP'nin temel görevleri aşağıda listeleniyor:

- Bir üst katmandan gelen verinin uygun uzunlukta parçalara bölünmesi (segmalara bölünmesi),
- Her bir parçaya alıcı kısımda aynı biçimde sıraya koyulabilmesi amacıyla sıra numarası verilmesi (sekans numarası verilmesi),
- Kaybolan veya bozuk gelen parçaların tekrarlanması (retransmission),
- Uygulamalar arasında yönlendirme yapılması (multiplexing),
- Güvenilir veri dağıtımın sağlanması,

- Hostlarda veri taşmasının önlenmesi (flow control).

TCP kendisine atanmış olan bu görevleri yapabilmek amacıyla aktarım katmanında veri parçalarının önüne başlık bilgisi ekler. Başlık bilgisi ve veri parçası birlikte TCP segmanı olarak anılır. Her segmana sıra numarası verilir. Hedef segmanlar kendine ulaştıkça bunları tampon belleğine yerleştirir. Gelen segmaların sırasında bir bozulma yoksa hedefte çalışan TCP gönderilen en son segman için bir onay mesajını kaynağa yollar.

Yukarıda saydığımız TCP görevlerinden en karmaşığı ve en önemlisi tıkanıklık denetimidir. Bu yüzden birçok araştırmacı, TCP tıkanıklık denetimi algoritmaları üzerinde çalışmalar yapmıştır. Telsiz linklerin de haberleşme ağlarına katılmasıyla TCP tıkanıklık denetimi algoritmalarının telsiz ortamlara uyarlanması son zamanlarda , üzerinde çok çalışılan konular arasına girmiştir.

2. TCP TIKANIKLIK DENETİMİ YÖNTEMLERİ

1980'li yılların sonlarında Van Jacobson, TCP'nin tıkanıklık kontrolü algoritmalarını tasarlarken paket kaybını tıkanıklık sinyali olarak kabul etmişti [1]. Paket kaybının sebebinin tıkanıklık olacağı varsayımı o zamanların telli haberleşme ağları için geçerli bir varsayımdı.

Haberleşme ağlarına telsiz linklerin de dahil olmasıyla birlikte, paket kayıplarının tıkanıklık yüzünden olduğu varsayımı geçerliliğini yitirmiş oldu. Paket bozulma oranları telli linklere göre daha fazla olan telsiz linkler üzerinde çalışan TCP'de mevcut tıkanıklık kontrolü algoritmaları çalıştırıldığında ciddi başarım kayıpları meydana gelir. Bunun sebebi kaybolan her segman için tıkanıklık kontrolü algoritmalarının tetiklenmesidir. Telsiz link üzerinde bozulmaya uğramış ve kaybolmuş bir segman için TCP tıkanıklık kontrolü algoritmalarını çalıştırırsa debisini gereksiz yere kısar. TCP'nin debisindeki bu azalma TCP veri iletim başarımında düşüşe sebep olacaktır.

Telli ve telsiz linklerden oluşan bir haberleşme ağında, TCP veri iletimi başarımını artırmak için segman kaybının gerçek sebebini ayırt edebilmek çok önemlidir. TCP kaybolan segmanların kaybolma nedenlerini öğrenebilirse:

- Tıkanıklık yüzünden kaybolmuş segmanlar için tıkanıklık kontrolü algoritmalarını çalıştırır.
- Telsiz linkler üzerinde bozulmaya uğrayarak kaybolmuş segmanlar için tıkanıklık kontrolü algoritmalarını çalıştırmaz, debisini kısmadan kaybolan segmanları yeniden aktarmaya çalışır.

Segmanların kaybolma sebeplerini ayırt etmeye yarayan yöntemler iki grupta toplanabilir.

1. Uçtan uca çalışan yöntemler
2. Ara düğümlerden yardım alan yöntemler.

Uçtan uca çalışan yöntemler paket kaybının sebebini ayırt edebilmek için bazı varsayımlarda bulunurlar.

1. Bütün segman kayıpları tıkanıklık yüzündendir varsayımı:

   Bu varsayım hata oranı çok küçük olan telli linkler için uygundur. Bu varsayımı yapan yöntemler fazladan tıkanıklık yaratmamak için paket kayıplarının bozulma yüzünden olduğu durumlarda bile veri iletim debilerini kısarlar ve bu durum başarımı olumsuz yönde etkiler. İnternet erişimi olan bilgisayarların hemen hemen hepsinde kullanılan TCP bu varsayıma dayanarak calışmaktadır.

2. Bütün segmanlar link üzerinde bozulmaya uğrayıp kaybolmuştur. Tıkanıklık yoktur varsayımı:

   Eğer tıkanıklık yaşanma olasılığı yok denecek kadar az ise kaybolan bir segman için segman bozularak kaybolmuştur diyebiliriz. Network kaynaklarının yeterli olduğu şartlarda çalışan yöntemler ya da kaynak rezervasyonu ile çalışan yöntemler bu varsayımı kullanabilir.

3. Segmanlar tıkanıklık yüzünden ya da bozulma yüzünden kaybolmuştur varsayımı:

   Bu varsayımı yapan yöntemler TCP'nin klasik tıkanıklık kontrolü algoritmalarını kullanmazlar. Segmanların kaybolma sebeblerini ayırt edebilmek için segmanların hedefe varışları arasındaki zaman farklılıkları [6], TCP bağlantının geçtiği yol üzerindeki tahmini kapasite [9], [7] gibi değerlere bakarlar.

Ara düğümlerin yardımıyla segman kayıplarını ayırt etmeye yarayan yöntemlere örnek olarak Snoop protokolünü [8] ve TCP bağlantısını ikiye bölerek çalışan Indirect-TCP protokolünü [10] verebiliriz.

2.1 İlk Nesil TCP Tıkanıklık Denetimi Yöntemleri

İlk nesil tıkanıklık denetimi yöntemleri telsiz linkleri hiç hesaba katmamışlardır. Bu yöntemler segman kaybını tıkanıklık sinyali olarak algılayıp gerekli önlemleri almaya çalışırlar.

2.1.1 TCP Tahoe

Van Jacobson tarafından 1980'li yıllarda tasarlanan TCP Tahoe [1], kayıp segmanları tıkanıklık habercisi olarak kabul eder. O yıllarda telsiz haberleşme ağları günümüzdeki kadar yaygın olmadığı için, TCP'nin tıkanıklık denetimi algoritmaları geliştirilirken TCP segmanlarının telsiz linklerde kaybolma olasılığı yok sayılmıştır.

TCP Tahoe'da tanımlanan tıkanıklık kontrolü algoritmalarının amacı olası bir tıkanıklık durumunda kaynağın debisini azaltmasını sağlamaktır.

TCP Tahoe, tıkanıklık denetimini Slow Start, Congestion Avoidence ve Fast Retransmission algoritmalarını kullanarak yapar.

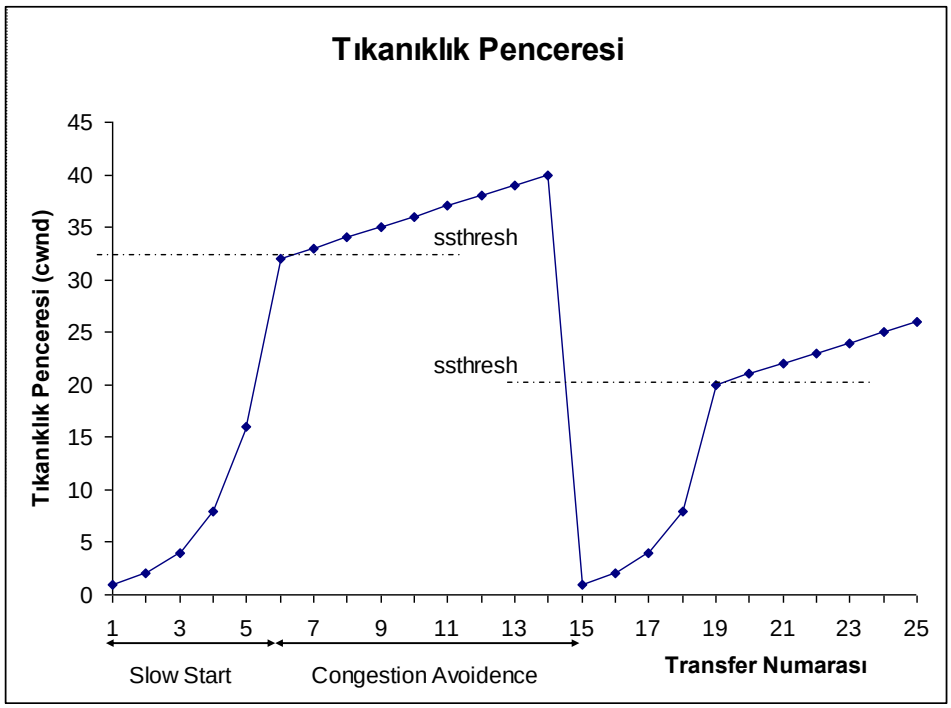
- **Slow Start :**

Slow Start algoritması iki değişkenden yararlanır. cwnd (tıkanıklık penceresi) ve ssthresh (slow start eşik değeri). Kaynak alındı (ACK) almadan min(cwnd ,awdn) kadar segman iletimi yapabilir. awnd değeri TCP'de akış kontrolü yapmak için kullanılır. awnd (advertised window), hedef tarafından kaynağa bildirilir ve hedefin alındı göndermeden en fazla kaç tane segmanı tampon alanında tutabileceğini gösterir. Böylece hızlı veri iletimi yapabilen bir kaynak hedefte çalışan ve veriyi yavaş işleyen bir TCP birimini boğmamış olur.

TCP bağlantısının başında cwnd değeri 1'dir. Kaynak aldığı her alındı için cwnd değerini bir artırır. Bu artış Şekil 1'de görüldüğü gibi üssel bir artıştır.

- **Congestion Avoidence :**

cwnd değeri ssthresh'e ulaştıktan sonra, cwnd değeri doğrusal olarak artmaya devam eder. Kaynak aldığı her alındı için cwnd değerini 1/cwnd kadar artırır.



Şekil 2.1: Tıkanıklık Penceresi

- **Fast Retransmit :**

Kaynak, bir TCP segmanının kaybolduğunu iki farklı şekilde anlar:

1. Belirli bir süre içerisinde kaynak alındı alamaz ise kaynakta segmanlar için tutulan yeniden iletim sayaçları zaman aşımına uğrar. (RTO-retransmission timeout.) Bu durum segmanın tıkanıklık yüzünden kaybolduğu anlamına gelir.

2. Aynı segman için N tane alındı alınırsa bu durum o segmanın tıkanıklık yüzünden kaybolduğu anlamına gelir. N genellikle 3 seçilir.

Kaynak, bir TCP segmanının kaybolduğunu anlarsa:

1. ssthresh değerini cwnd değerinin yarısı yapar. Kaybolan segmanı yeniden gönderir.

2. cwnd değerini 1 yapar böylece tekrar Slow Start algoritmasını çalıştırdığında iletim debisi başlangıçtaki değerini almış olur.

3. Yeniden iletim sayacının zaman aşımı süresini uzatır. Bu değer genellikle iki katına çıkarılır. Böylece yeniden iletilen segmanın hedefe ulaşması için daha fazla zaman tanınmış olur.

2.1.2 TCP Reno

TCP Tahoe, 3 adet tekrarlanan alındının ardından Slow Start algoritmasını çalıştırır. Slow Start, kaynağın debisini en az olacak şekilde kısar (cwnd = 1). TCP Reno ise 3 DUPACK ardından Slow Start algoritmasını çalıştırmak yerine Fast Retransmit ve Fast Recovery algoritmalarını çalıştırır [2].

Belirli bir süre boyunca (RTO), kaynak gönderdiği segmanlar için alındı alamazsa bu durum ağır bir tıkanıklığın habercisi olabilir. Fakat kaynağın aynı segman için birden fazla tekrar eden alındı (DUPACKs) alması ağır bir tıkanıklık durumu ile karşılaşıldığı anlamına gelmez. Kaynağın alındı alması segmanların hedefe ulaştığı anlamına gelir. Yani haberleşme kanalındaki segmanlar kanaldan çıkıp hedefe varmıştır.TCP Reno, 3 tane DUPACK aldığı zaman TCP Tahoe gibi cwnd değerini 1'e indirmez, fast retransmit ve fast recovery algoritmalarını çalıştırır.

Fast Recovery algoritması şu sekilde çalışır:

- ssthresh, cwnd'un yarısı yapılır (2 segmandan küçük olmayacak şekilde). Kaybolan segman yeniden transfer edilir. $cwnd = ssthresh + 3$ yapılır. Böylece cwnd hedefe ulaşan TCP segmanı sayısı kadar büyütülmüş olur.

- Kaynak, aldığı her tekrar eden alındı için tıkanıklık penceresini (cwnd) bir segman boyu büyütür. Tekrar eden alındı alınması haberleşme kanalından bir segmanın daha çıktığını bildirir. Eğer tıkanıklık penceresi izin verirse yeni bir segman gönderilebilir

- Kaynak yeni veriyi onaylayan bir alındı alırsa, bu alındı yeniden transfer edilen segmanı da kapsar. cwnd değeri ssthresh yapılır. Tıkanıklık penceresinin boyu TCP Tahoe'da olduğu gibi 1 değil, segmanın kaybolduğunun anlaşıldığı andaki değerin yarısı olur. Böylece TCP Reno, debisini paketin kaybolduğu andakinin yarısı yapmış olur.

2.1.3 TCP New Reno

Aynı tıkanıklık penceresi içerisinden birden fazla TCP segmanı kaybolursa TCP Reno, Fast Recovery algoritmasından erken çıkar. TCP Reno, Fast Recovery'den çıktığında kaynak, kaybolan ve yeniden gönderilen segmanların tümü için henüz alındı almamıştır. TCP New Reno , Fast Recovery algoritmasında yaptığı küçük bir değişiklik ile bu problemi çözmeye çalışır [17].

Aynı pencere içinden birden fazla TCP segmanının kaybolduğu durumlarda, TCP New Reno, Fast Recovery aşamasında yeni bir alındı alınca hemen Fast Recovery'yi bitirmez. Fast Recovery'den çıkmak için o zamana kadar gönderilen en büyük sekans numaralı segmanı da kapsayan bir alındının gelmesini bekler. TCP Reno aldığı ilk yeni alındıda Fast Recovery'den çıkıyordu ve aynı pencere içinde birden fazla segman kaybı oldugu durumlarda Fast Retransmit ve Fast Recovery algoritmalarını birden fazla çalıştırarak debisini sürekli yarıya indiriyordu.

TCP New Reno, bir tıkanıklık penceresi içerisinden birden fazla segmanın kaybolduğu durumlarda TCP Reno üzerine yapılan bir iyileştirmedir.

2.2 Telsiz Ağlarda TCP Tıkanıklık Denetimi Yöntemleri

Telsiz linklerin en önemli özelliği yüksek hata oranlarıdır. Telli linklere göre, telsiz linkler üzerinde paketlerin bozulma olasılığı daha fazladır.

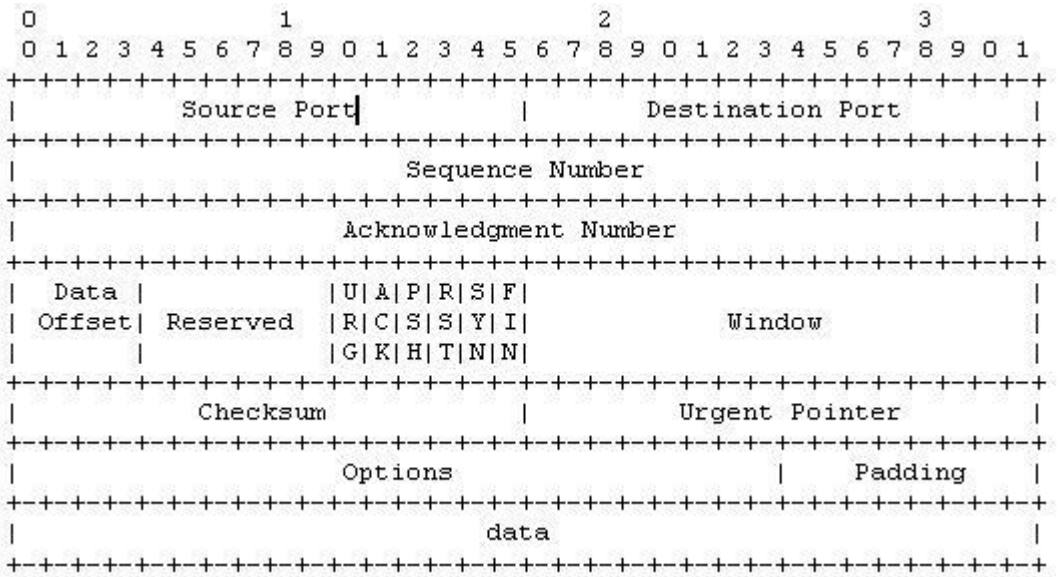
Telsiz ortamda veri iletimini zorlaştıran etkiler çeşitlidir. Ortamdaki gürültü, kırılmalar (refraction), multipath bu etkilerden bazılarıdır [18]. Telli linklerden farklılık gösteren telsiz linkler için bölüm 2.1'de anlatılan tıkanıklık denetimi algoritmalarını kullanamayız. Telsiz ortamlarda da çalışması beklenen tıkanıklık denetimi algoritmalarının, TCP segmanlarının telsiz ortamlarda da kaybolabileceği düşünülerek tasarlanması gerekmektedir.

Bu bölümde telsiz ortamlar için tasarlanmış TCP tıkanıklık denetimi yöntemlerinden önemli olan bir kaç tanesi anlatılıyor.

2.2.1 Freeze TCP

Şekil 2-2 'de TCP başlığının formatı görülmektedir. TCP başlığında bulunan "Window" alanı hedefte çalışan TCP tarafından doldurulur. Alındı segmanları (ACK'ler) içerisinde taşınır ve hedefin ne kadar sekizli istediğini kaynağa bildirir. TCP akış denetimi yaparken TCP başlığındaki bu alanı kullanır. Hedef tarafında yeterli tampon alan kalmazsa "Window" alanına 0 yazılarak kaynağa alındı gönderilebilir. Kaynağa 0'lık bir "Window" gönderilmesi olayına "Zero Window Advertisement" denir. Bu durumda:

- Kaynak, hedef tekrar penceresini açana kadar veri gönderemez.
- Kaynak, TCP bağlantısını açık tutmak zorundadır.
- Kaynak, yeniden iletim sayaçlarının hepsini durdurur.



Şekil 2.2: TCP Başlık Formatı

[3]'te anlatılan yöntem TCP'nin yukarıda anlatılan özelliğini kullanarak, telsiz ağlarda TCP'nin veri iletim başarımını artırmayı amaçlıyor.

Freeze-TCP tekniğini gerçekleştirmek için sadece hedefte çalışan TCP'de değişiklik yapmak yeterli oluyor. TCP bağlantısının geçtiği düğümlerde ya da kaynakta bir değişiklik yapmaya gerek yok. Freeze-TCP hedef düğümün, sinyal gücünü dinlemesine ve sinyal gücündeki değişmelere bakarak olası bir bağlantı kopukluğunu önceden sezmesine dayanır. Sinyal gücünün zayıfladığı durumda hedef kaynağa "Window" alanı 0 olan bir alındı gönderir (Zero Window Advertisement).

Freeze-TCP tekniği sadece iletişim bağında kopukluk olduğu durumlarda işe yarar. Telsiz ağlarda sinyal gücünün yeterli seviyede olduğu durumlarda ve veri iletiminin devam ettiği durumlarda bu teknik TCP'nin veri iletimi başarımına bir katkı sağlamaz.

Freeze-TCP uygulanırken hedefin bağlantı kopukluk anını önceden sezebildiği varsayılıyor. Kopma gerçekleşmeden T süre önce kaynağa ZWA (Zero Window Advertisement) gönderiliyor. Freeze-TCP tekniğinin doğru çalışabilmesi için T süresini iyi seçmek gerekir. T süresi çok büyük seçilirse kaynağa erkenden ZWA gönderilmiş olur ve kaynağın debisini erken kapatmasına sebep olur. T süresi çok küçük seçilirse hedefin gönderdiği ZWA'nın kaynağa ulaşamadan kaybolma ihtimali vardır. Bu durumda kaynak bağlantı kopukluğu yüzünden kaybolan segmanlar için RTO (Retransmit timeout) ile tetiklenene kadar bekler sonra da Slow Start yaparak veri iletimine kaldığı yerden devam eder. Yapılan tesler T için en uygun değerin RTT (Round Trip Time) olduğunu gösteriyor.

2.2.2 TCP Peach

TCP-Peach uydu haberleşmesinde kullanılmak üzere geliştirilmiş bir protokoldür [7]. Uydu haberleşmesinin belirgin özelliği hata yüzdesinin yüksekliği ve gecikme zamanının yüksekliğidir. TCP-Peach uydu haberleşmesindeki bu zorlukları geliştirdiği yeni tıkanıklık denetimi algoritmaları ile çözmeyi amaçlamaktadır.

TCP'de kullanılan Slow Start algoritması uydu haberleşmesinde problemlidir. Slow Start evresinde olan bir bağlantının B bit oranına (bit rate) ulaşabilmesi için gereken süre aşağıdaki denklem 2.1'de gösterilmiştir.

$$t_{SlowStart} = RTT.(1+\log_2 B.RTT / l) \qquad \textbf{(2.1)}$$

t zamanı RTT ile doğru orantılı olduğu için RTT ne kadar büyük olursa TCP'nin istenen debiye ulaşması için geçecek süre de o kadar uzun olur. Uydu

haberleşmesinin temel problemi RTT değerlerinin yüksek olmasıdır. Aşağıdaki tabloda çeşitli örnekler mevcut. Örneğin GEO tipinde bir uydu için 10 Mb/saniye debiye ulaşmak için geçen Slow Start süresi 2.32 saniyedir.

Tablo 2.1: LEO, MEO ve GEO uyduları için Slow Start süreleri

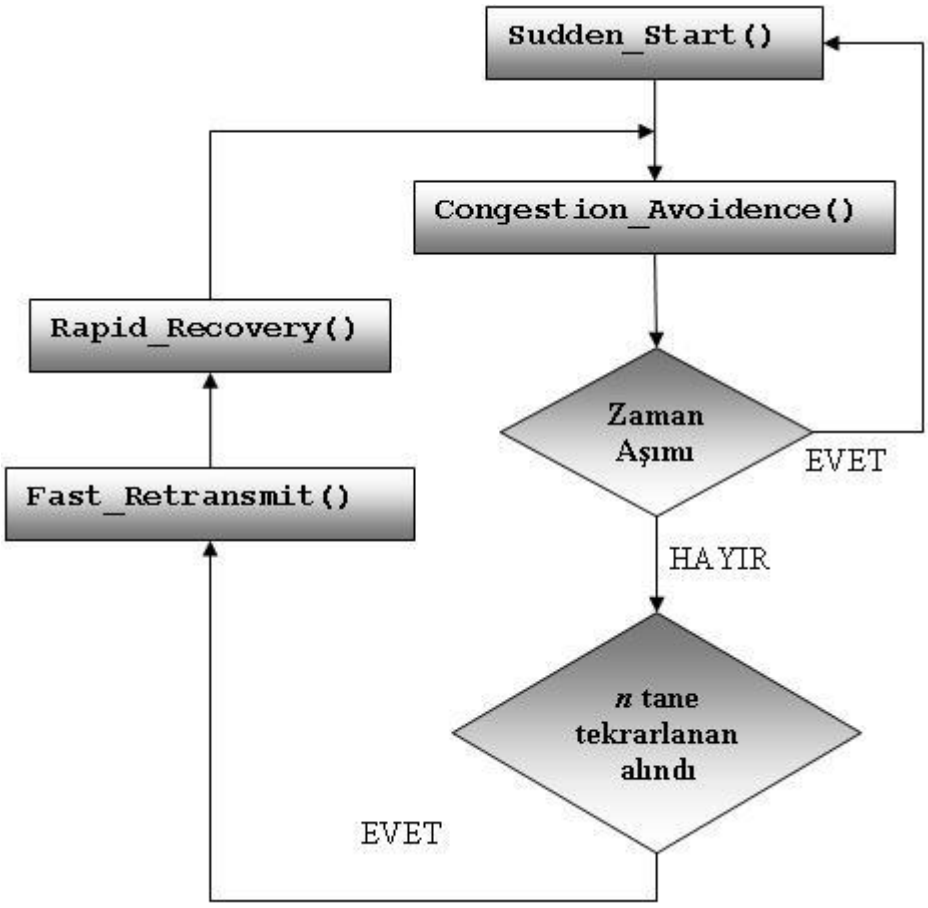
Uydu Tipi	RTT	t_slow_start	t_slow_start	t_slow_start
LEO	50 msec	0.18 sec	0.35 sec	0.55 sec
MEO	250 msec	1.49 sec	2.32 sec	3.31 sec
GEO	550 msec	3.91 sec	5.37 sec	7.91 sec

TCP-Peach tıkanıklık denetimi algoritmalarını yürütürken Dummy Segment adı verilen TCP segmanlarından yararlanmaktadır. Dummy Segment'ler kaynak tarafından üretilen düşük öncelikli segmanlarıdır. En son gönderilen TCP segmanının kopyası olarak üretilirler. Kaynak iletişim ağının kaynaklarının yeterli olup olmadığını Dummy Segment'ler ile anlamaya çalışır. Düşük öncelikli oldukları için gerektiğinde yönlendiricilerde ilk çöpe atılacak segmanlar Dummy Segment'lerdir. Dummy Segment'ler için geri gönderilen alındılar da hedef tarafından işaretlenir. Bu alındılar da düşük öncelikli segmanlar olarak iletişim ağında taşınır. Dummy Segment'lerin kullanımı kısaca şöyledir: Kaynak gönderdiği Dummy Segment'lar için alındı alınca kullanılmayan ağ kaynaklarının olduğunu anlar ve veri iletim debisini artırır.

TCP-Peach Sudden Start, Congestion Avoidence, Fast Retransmit ve Rapid Recovery algoritmalarından oluşur.

TCP-Peach Sudden Start:

TCP-Peach kaynağı veri iletimine Slow Start yerine Sudden Start algoritması ile başlar. Bir RTT süresi içinde 1 tane veri tasşıyan TCP segmanı ve (rwnd-1) tane de Dummy Segment gönderir. (rwnd, hedefin bildirdiği boş tampon alan büyüklüğüdür.) TCP-Peach bağlantının ileri aşamalarında Sudden Start sırasında gönderdiği Dummy Segment'leri veri iletim debisini artırmaya ya da kısmaya karar vermek için kullanacak. Şekil 2.4'te TCP-Peach kaynağının kullandığı Sudden Start algoritması görülüyor.



Şekil 2.3: TCP-Peach Tıkanıklık Denetimi Algoritmaları

```
Sudden_Start()
        cwnd  = 1;
        τ   = RTT / rwnd;
        send(Data_Segment);
        for (i=0 to rwnd-1)
                send(Dummy_Segment);
        end;
end;
```

Şekil 2.4: TCP-Peach Sudden Start

t=0 anında bir TCP bağlantısının kurulduğunu varsayalım.

- $0 \leq t < RTT$

Bu arada kaynak 1 tane veri taşıyan TCP segmanını ve rwnd-1 tane Dummy Segment'i alındısını almadan gönderir. Kaynak toplamda rwnd tane segman göndermiş olur. Bu segmanların gönderim zamanlarını 0-RTT zaman aralığına eşit olarak yayar.

- $RTT \leq t < 2 * RTT$

0-RTT zaman aralığında gönderilen segmanların alındıları gelmeye başlar. Bu aşamada TCP-Peach algoritmalarında kullanmak üzere yeni bir değişken tanımlanıyor: **wsdn.** Kaynak alındı aldığı zaman wsdn'nin o andaki değerine göre yeni segman gönderip göndermemeye karar verir

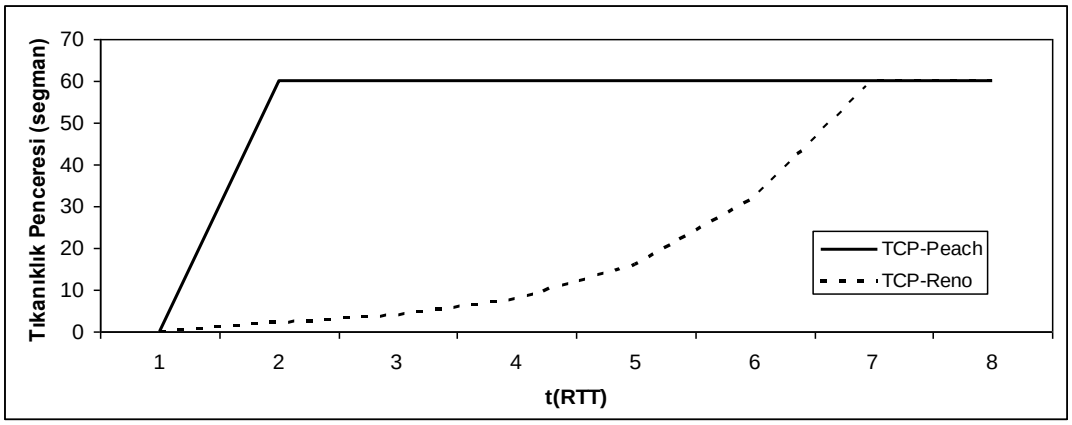
- wsdn sıfır ise, kaynak alındı aldığı zamam cwnd (tıkanıklık penceresi) değerini bir artırır.
- Eğer wsdn sıfırdan farklı bir değerde ise, wsdn bir azaltılır, cwnd değeri aynı kalır.

wsdn değişkeninin kullanılma amacı, tıkanıklığın yaşandığı durumlarda TCP-Peach kullanan bir kaynağın TCP-Reno gibi davranmasını sağlamaktır. Bağlantının başlangıcında wsdn'nin değeri sıfır yapılır.

(RTT ≤ t < 2 * RTT) zaman aralığında wsdn değeri sıfır olduğu için Dummy Segment'ler için gelen her alındıda cwnd değeri bir artırılıyor. Böylece kaynak bu zaman aralığında alındı almayla aynı hızda data segman gönderebiliyor. Bu aşamada kaynak Dummy Segment'ler için alındı almazsa, Dummy Segment'lerin hedefe ulaşamadığı ve dolayısıyla iletişim ağında bir tıkanıklık olduğunu anlar.

- t ~ 2 * RTT

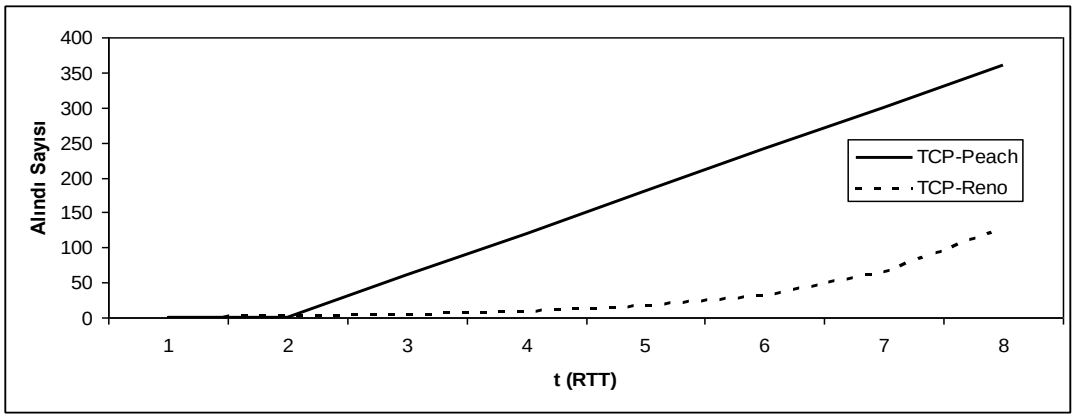
En son gönderilen Dummy Segment için alındının bu zamanda gelmesi beklenir. Kaynak Congestion Avoidence algoritmasını çalıştırır.



Şekil 2.5: Bağlantı başlangıcında TCP-Peach ve TCP Reno tıkanıklık penceresi karşılaştırması

Şekil 2.5'te TCP-Peach ve TCP Reno'da tıkanıklık penceresinin, TCP bağlantısının başlangıcında, zaman içinde aldığı değerler karşılaştırılıyor. Şekilde zaman birimi

olarak RTT kullanılmış. Sudden Start sayesinde TCP-Peach'de tıkanıklık penceresinin TCP Reno'ya göre daha hızlı açıldığı şekilde görülmektedir.



Şekil 2.6: Bağlantı başlangıcında TCP-Peach ve TCP Reno iletilen segman sayısı karşılaştırması

Şekil 2.6'da alındısı alınmış TCP segmanlarının sayısının zamana göre grafiği görülmektedir. Şekilden de anlaşılacağı gibi TCP-Peach Sudden Start sayesinde bağlantı başlangıcında TCP-Reno'dan daha fazla TCP segmanı iletimi yapar.

TCP-Peach Rapid Recovery:

Rapid Recovery algoritması, Fast Recovery [2] algoritmasının yerine geçer. Telsiz linklerde meydana gelen bozulmalardan dolayı, TCP veri iletim başarımında meydana gelen düşüşü engellemek için tasarlanmıştır.

TCP-Peach kaynağı, N tane tekrarlanan alındı alırsa önce Fast Retransmit [2] algoritması ile kaybolan segmanı yeniden gönderir, daha sonra yeniden gönderilen TCP segmanı için alındı gelene kadar Rapid Recovery algoritmasını çalıştırır. Rapid Recovery algoritması ortalama bir RTT sürer ve sonra kaynak Congestion Avoidance aşamasına geçer.

Rapid Recovery algoritması, ilk başta, TCP segmanı tıkanıklık yüzünden kaybolmuştur varsayımı yaparak tıkanıklık penceresi değerini TCP-Reno'da [2] olduğu gibi yarıya indirir. Tıkanıklık penceresinin değerine $cwnd_0$ dersek, tıkanıklık penceresi $cwnd_0/2$ yapılır. Bunun dışında, iletişim ağındaki kaynakların (kullanılabilir bant genişliği) durumunu öğrenebilmek için de bir miktar Dummy Segment gönderilmesi gerekmektedir. Rapid Recovery aşamasında ne kadar Dummy Segment gönderilmesi gerektiğine şöyle karar verilir: Gönderilen Dummy Segment'ler için alındılar, yeniden gönderilen TCP segmanı için gelen alındıdan sonra gelecektir. (Yani kaynak Congestion Avoidence algorimasını çalıştırırken.)

TCP segmanı tıkanıklık yüzünden kaybolmuş ise tıkanan yönlendiriciler bir RTT içerisinde yaklaşık olarak $cwnd_0$ tane segmana hizmet verebilirler. Bu durumda iletişim ağı $cwnd_0/2$ tane normal TCP segmanı, $cwnd_0/2$ tane de Dummy Segment taşıyabilir. TCP-Peach kaynağı Congestion Avoidence aşamasındayken, gönderdiği Dummy Segment'lerden $cwnd_0/2$ tanesi için gelen alındılarda, tıkanıklık penceresini açmamalıdır. Bunu sağlamak için Sudden Start açıklanırken tanıtılan **wsdn** değişkenin değeri $cwnd_0/2$ yapılır. Kaynak Dummy Segment'ler için $cwnd_0/2$ tane alındı aldıktan sonra Dummy Segment için gelen her alındıda tıkanıklık penceresi değerini bir artırır. Bu bilgiler ışığında TCP-Peach Rapid Recovery aşamasında göndereceği Dummy Segment sayısını $cwnd_0$ olarak belirler. Sonuç olarak, gönderilen Dummy Segment'lerin hepsi için alındı gelirse yani iletişim ağında bir tıkanıklık durumu yoksa, tıkanıklık penceresinin değeri TCP segman kaybının farkedildiği andaki değerine eşit olacaktır. Tıkanıklık durumlarında gönderilen Dummy Segment'ler için alındılar gelemeyecektir. Böylece TCP-Peach kaynağı tıkanıklık penceresini açamayacaktır. TCP segmanının tıkanıklık yüzünden kaybolduğu durumlarda TCP-Peach, TCP-Reno gibi çalışmaktadır.

Yeniden transfer edilen TCP segmanı da kaybolabilir. t_{Retr} 'a kabolan segmanın yeniden gönderildiği zaman dersek, (t_{Retr} + RTO) zamanı sonunda yeniden gönderilen segman için alındı gelmez ise Rapid Recovery'ye son verilir ve kaynak Sudden Start algoritmasını çalıştırır.

TCP-Peach, iletişim ağı kaynaklarının kullanılabilirliğini ölçmek için Dummy Segment'lerden faydalanmış Sudden Start ve Rapid Recovery algoritmaları ile telsiz ortamlarda TCP veri iletim başarımında meydana gelen problemleri çözmüştür.

2.2.3 TCP Westwood

TPC Westwood tekniğinin amacı telsiz ortamda, TCP'nin veri iletim başarımını artırmaktır. TCP Westwood uçtan uca çalışan bir tekniktir. TCP bağlantısının geçtiği yol üzerindeki ara düğümlerin yardımına ihtiyaç duymaz. TCP Reno üzerine geliştirilen TCP Westwood, sadece TCP kaynağında değişiklik yapılmasını gerektirir. TCP alıcısında herhangi bir değişiklik yapmaya gerek yoktur [9].

TCP Westwood (TCPW) kaynağı, alındı (ACK) alma sıklığını sürekli gözlemler ve bu gözlemler sayesinde bağlantının taşıdığı TCP segmanı miktarını (bant genişliğini) tahmin etmeye çalışır. Kaynak TCP segmanı kaybı sinyali aldığı zaman (örneğin 3 tane tekrarlanan alındı ya da yeniden iletim sayacının zaman aşımına uğraması), tıkanıklık penceresini ve Slow Start eşik değerini yeniden seçmek için bağlantının kullandığı tahmini bant genişiliği değerine bakar. Tahmin edilen bant genişliği

değerinin tıkanıklık penceresi ve Slow Start eşik değerinin ayarlanmasında kullanılması gereksiz yere TCP kaynağının iletim debisinin kısılmasını engeller. TCP Westwood tekniğinde paket kaybı sinyalinden sonra yapılan kaynaktaki debi ayarlamasına Faster Recovery deniyor. TCP segmanlarının telsiz linklerde kaybolduğu durumlarda Faster Recovery sayesinde TCP Westwood kaynağı, TCP Reno kaynağı gibi debisini yarıya ya da 1 segman/RTT'ye indirmez.

Kaynağa gelen alındı, bir miktar verinin hedefe ulaştığı anlamını taşır. Eğer iletim sırasında hiç segman kaybı olmazsa gönderilen verinin zamana bölünmesi ile bağlantının kullandığı bant genişliği bulunabilir. TCP Westwood'u tasarlayan araştırmacılar TCP bağlantısı boyunca, TCP segmanlarının eşit büyüklükte olduğunu varsayıyorlar. Bunun sebebini şöyle açıklayabiliriz: Kaynak tekrarlanan alındılardan alırsa, bir miktar TCP segmanının hedefe ulaştığını anlar fakat hedefe ulaşan TCP segmanlarından hangilerinin bu tekrarlanan alındıların gönderilmesini tetiklediğini bilemez. Hedefe gönderilen TCP segmanları farklı boyutlarda olursa tekrarlanan alındı geldiğinde kaynak hedefe ulaşan verinin miktarını doğru hesaplayamaz.

Alındı tekrarlanan bir alındı ise bant genişliği hesabı, toplam gönderilen segman sayısına bir segman eklenerek yapılır. Alındı yeni segmanlar için gönderilmiş ise iki durum vardır. Hedef ya kümülatif bir alındı göndermiştir ya da geciktirilmiş bir alındı göndermiştir. Hedefin kaynağa geciktirilmiş alındı (delayed ACK) göndermesi durumunda TCP'nin kullandığı bant genişliğini hesaplamak biraz daha karmaşıktır. Kaynak alındı aldığı zaman, Şekil 2.7'te görülen algoritma ile bant genişliği hesabına ekleyeceği segman sayısını hesaplar.

```
PROCEDURE AckedCount

cumul_ack = current_ack_seqno - last_ack_seqno;
if (cumul_ack = 0)
      //Dup. ACK alındı
      accounted_for=accounted_for + 1;
      cumul_ack=1;
endif

if (cumul_ack > 1)
      //delayed ACK ya da Cumulative ACK.
      if (accounted_for >= cumul_ack)
            accounted_for = accounted_for - cumul_ack;
            cumul_ack = 1;
      elseif (accounted_for < cumul_ack)
            cumul_ack = cumul_ack - accounted_for;
            accounted_for = 0;
      endif
endif

last_ack_seqno = current_ack_seq_no;
acked = cumul_ack;
return (acked);,

END PROCEDURE
```

Şekil 2.7: TCP Westwood AckedCount Algoritması

TCP Westwood, kullandığı bant genişliğini hesapladıktan sonra bu değeri TCP segmanlarının kaybolduğu durumlarda cwnd ve ssthresh'i yeniden hesaplamada kullanıyor. Kaynak TCP segmanlarının kaybolduğunu iki şekilde anlayabilir.

- Kaynak 3 tane tekrarlanan alındı almıştır
- Yeniden iletim sayaçları zaman aşımına uğramıştır.

```
if (3 tekrarlanan alındı)
      ssthresh = (BWE * RTTMin) / seg_size;
      if (cwnd > ssthresh)
            // congestion avoidence
            cwnd = ssthresh;
      endif
endif
```

Şekil 2.8: TCPW 3 tekrarlanan alındı alınması durumu

Kaynağın 3 tane tekrarlanan alındı aldığı durumda cwnd ve ssthresh değerlerini nasıl güncellediği Şekil 2.8'te görülmektedir. BWE (bandwidth estimation) hesaplanan tahmini bant genişliği değerini tutar. RTTMin, bağlantı süresince gözlemlenmiş en küçük RTT değerini tutar.

```
if (coarse_timeout_expires)
     ssthresh = (BWE * RTTmin) / seg_size;
     if (ssthresh < 2)
          ssthresh = 2;
     endif
     cwnd = 1;
endif
```

Şekil 2.9: TCPW zaman aşımı durumu

TCP Westwood gönderilen bir segman için RTO süresi içinde alındı alamazsa TCP Reno'da olduğu gibi ssthresh değerini yarıya indirmiyor. sshthresh tahmini bant genişliği oranında bir değere çekiliyor. Şekil 2.9'te de görülen bu yönteme Speedy Recovery adı verilmiş.

Sonuç olarak TCP Westwood TCP segmanlarının kaybolmasının ardından gerekli müdahaleyi yaparken segmanın kaybolduğu anda hesaplanan bant genişliği değerine bakar. Böylece ssthresh ve cwnd değerleri her segman kaybında körü körüne bire ya da yarı değerine indirilmez, hesaplanan bant genişliği oranında bir değere eşitlenir. Bu sayede TCP Westwood telsiz ortamda paket bozulmalarından dolayı meydana

gelen segman kayıplarında debisini ayarlarken TCP Reno'ya göre daha akıllı davranmış olur. TCP Westwood'un artılarından bir tanesi de sadece kaynak tarafında yapılacak değişikliklerle çalışabilmesidir. Yazarlar yaptıkları simulasyonlarla TCP Westwood'un telsiz ortamlarda TCP Reno'ya göre başarımının yüksek olduğunu göstermişler.

2.2.4 TCP HACK

TCP HACK (**H**e**A**der **C**hec**K**sum), telsiz linklerde bozulmaya uğrayıp kaybolan segmanları, tıkanıklık yüzünden kaybolan segmanlardan ayırt etmek için TCP'nin üzerine geliştirilmiş bir tekniktir [4].

Varsayılan maksimum TCP segmanı büyüklüğü 536 sekizli veri (payload) ve 20 sekizli başlık (header) olmak üzere 556 sekizlidir. Buna 20 sekizli uzunluğundaki IP başlığı da eklenince varsayılan maksimum TCP segmanı büyüklüğü (MSS) 576 sekizli oluyor. Burada dikkat edilmesi gereken nokta başlık alanın paketin % 5'inden küçük olduğudur. TCP HACK metodu bu bilgiye dayanarak, kablosuz ortamda taşınan bir TCP segmanında iletim sırasında bozulma olursa bu bozulmanın segmanın data kısmında olma olasılığının çok yüksek olduğu varsayımı üzerine kurulmuştur. TCP başlığının bozulma olasılığı daha azdır.

TCP HACK, bozulmaya uğramış TCP segmanından başlık bölümünü (eğer başlık bozulmamış ise) kurtararak iletişim kanalında tıkanıklık mı olduğunu yoksa iletim sırasında bir bozulma mı oldugunu anlayabiliyor. Bunu yaparken bütün TCP segmanı için kullanılan TCP checksum hesabı dışında TCP başlığı için de ayrı bir checksum hesabı yapıyor.

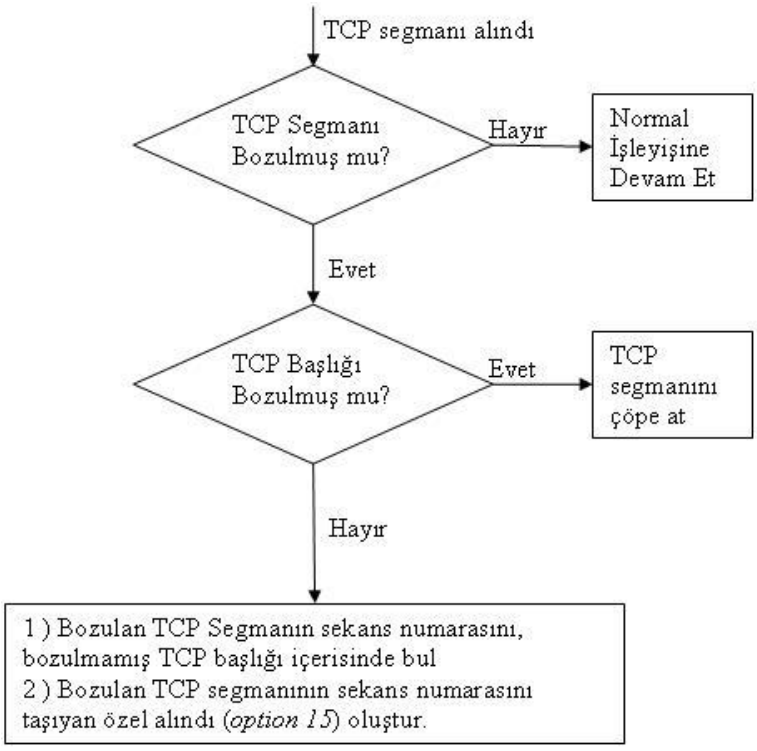
RFC 793'de [5] açıklandığı gibi TCP'de checksum alanı başlık ve veri (payload) de dahil olmak üzere segmandaki tüm 16-bit kelimelerin 1`e tümlenmiş bir toplamını içerir. IP'deki checksum'ın aksine (IP'de sadece başlık için checksum hesaplanır.) TCP'de checksum bütün segman için hesaplanıyor. TCP alıcısı tarafında, iletim sırasında meydana gelen bozulmalar nedeniyle checksum kontrolü segmanın bozulduğuna karar verirse, segman çöpe atılır. Fakat genelde çöpe atılan bu segmanlarda başlık kısmı kurtarılabilir durumdadır. Çünkü bozulmanın sadece veri alanında olma olasılığı, bozulmanın başlık alanında olma olasılıpından yüksektir. TCP HACK başlık için TCP segmanına ayrı bir checksum koymayı önerir. Böylece checksum kontolünden geçememiş segmanların başlık kısmı kurtarılabilir.

TCP HACK tekniğinin kullanılabilmesi için TCP alıcısında ve TCP kaynağında bazı değişiklikler yapılması gerekir.

TCP HACK Alıcısında Yapılması Gereken Değişiklikler:

TCP alıcısı, bir segman aldığında ilk iş olarak checksum kontrolünü yapar. Segman checksum kontrolünü geçerse, TCP segmanı kaynaktan hedefe kadar bozulmadan gelmiş demektir ve normal işleyiş devam eder.

TCP checksum kontrolü başarısız olursa, alıcı Header Checksum kontrolü yapar. Header Checksum kontrolü sonucunda segmanın başlık kısmında bozulma olmadığını anlarsa kaynağa özel bir alındı gönderir. Kaynağa gönderilen bu özel alındı bozulan segmanın sekans numarasını taşır ve segmanın bozulduğunu kaynağa bildirir. Böylece kaynak tıkanıklık algoritmalarını çalıştırmadan bozulan segmanı tekrar gönderebilir. Kaynak bozulan segmanı yeniden iletirken mevcut tıkanıklık penceresine bakar. Tıkanıklık penceresi izin veriyorsa veri gönderebilir. Tıkanıklık penceresini küçültmez. Şekil 2.10 TCP HACK alıcısında yapılan değişiklikleri göstermektedir.

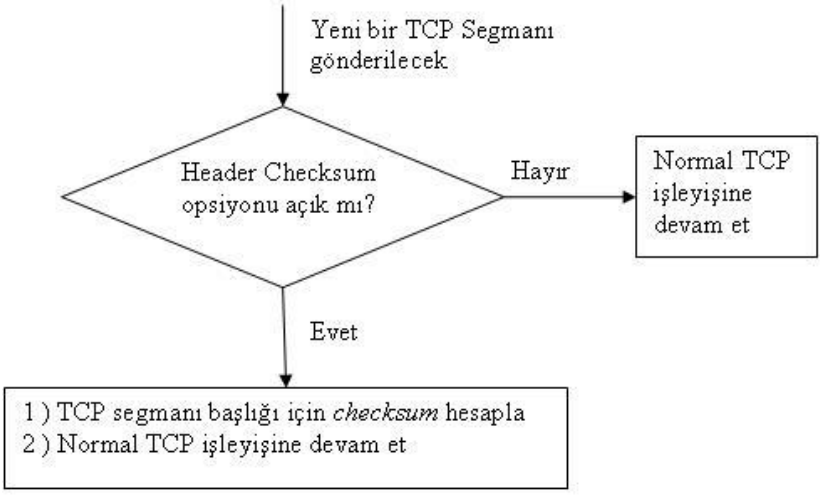


Şekil 2.10: TCP HACK alıcısında yapılan değişiklik

TCP HACK Kaynağında Yapılması Gereken Değişiklikler:

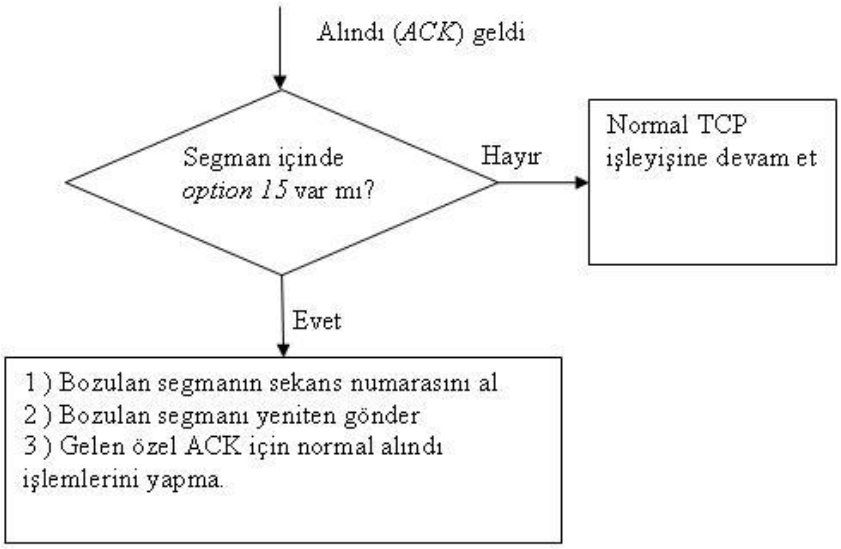
TCP kaynağı veri iletimine başlamadan önce, kurulan TCP bağlantısı için Header Checksum opsiyonunun kullanılıp kullanılmayacağına bakar. Bağlantı kurma esnasında kaynak ve hedef arasında Header Checksum opsiyonu kullanımına karar verilir.

Header Checksum opsiyonu kullanılmayacak ise; kaynak, normal standart TCP'de olduğu gibi veri iletimini yapar. Header Checksum opsiyonu kullanılıyorsa; kaynak, TCP başlığı için bir checksum hesaplayıp bunu TCP segmanının opsiyon kısmında Header Checksum opsiyonu alanına yazar. TCP kaynağında yapılan değişiklikler Şekil 2.11'de gösteriliyor. Bölüm 3.7.2 de TCP opsiyonları ile ilgili açıklamalar bulunmaktadır.



Şekil 2.11: TCP kaynağında yapılan değişiklikler

TCP HACK tekniğinde, kaynakta alındıların işlenmesinde de değişiklikler yapılıyor. Bu değişiklikler Şekil 2.12'te görülmektedir.

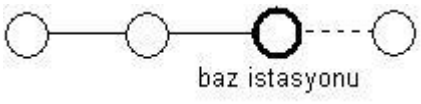


Şekil 2.12: TCP HACK Özel Alındı (ACK) İşleyişi

TCP segmanları IP datagramları, IP datagramları da ikinci katman çerçeveleri içerisinde taşınıyor. TCP segmanı içerisinde bir bozulma meydana gelirse ikinci katmanda CRC (Cyclic Redundancy Check) kontrolü sırasında hata meydana gelecek ve TCP segmanı 4. katmana (aktarım katmanı) çıkamadan çöpe atılacak. Yazarlar bozulan TCP segmanlarının aktarım katmanına kadar çıkartılması için simulasyonlarında kullandıkları ethernet kartlarının sürücülerini CRC kontolünü geçemeyen paketleri çöpe atmayacak şekilde değiştiriyorlar. Böylece network kartlarına gelen bozulmuş paketler çöpe atılmadan TCP/IP yığınında üst katmanlara veriliyor.

2.2.5 TCP Segmanlarının Hedefe Varışları Arasındaki Zaman Farkından Yararlanarak Telsiz Linklerde Kaybolan TCP Segmanlarını Tespit Etme Yöntemi

Bu teknikte yazarlar paket kayıplarının sebebini ayırt edebilmek için alıcı tarafında, gelen paketler arasındaki zaman farklarından yararlanıyorlar [6].

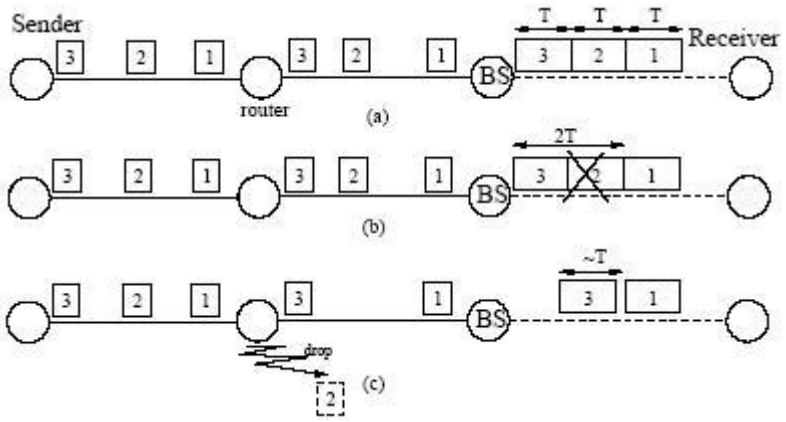


Şekil 2.13: Model

Şekil 2.13'de düz çizgi telli bir hattı, kesik kesik çizgiler ise telsiz bir hattı temsil ediyor. Makalede kullanılan senaryoda Şekil 2.6'daki gibi telli ve telsiz linkler bulunmaktadır ve bu linklerden telsiz olanı darboğaz linktir. Hareketli bir alıcının sabit bir kaynak ile haberleştiği bir model düşünülmüştür.

Bu teknik paket kayıplarını doğru bir şekilde ayırt edebilmek için bazı varsayımlarda bulunuyor:

- Baz istasyonunda ve alıcıdaki paket işleme zamanı ihmal edilebilir.
- Bağlantının geçtiği hat üzerinde sadece son link telsiz.
- Telsiz link darboğaz.
- Kaynak "bulk" veri iletimi yapıyor. Kaynağın bağlantı boyunca sürekli transfer edecek verisi var. (FTP sunucusu olabilir.)



Şekil 2.14: TCP segmanlarının alınması arasındaki zaman farkları

Şekil 2.14, kaynaktan çıkıp, hedefe ulaşan segmanların arasında oluşabilecek zaman farklılıklarını göstermektedir. 3 durum söz konusudur :

1. TCP segmanlarının hepsi alıcıya ulaşıyor. Segman kaybı yok. Bu durumda segmanların alıcıya varış zamanları arasında T kadar bir süre vardır. T yaklaşık olarak baz istasyonunda bir paketin telsiz ortama bırakılması için geçen süreye eşittir.

2. 2 numaralı segman kablosuz ortamda bozulmaya uğruyor. Bu durumda alıcıya gelen iki segman (1 ve 3) arasında geçen süre 2T olur. Çünkü 2 numaralı segmanı kablosuz ortama bırakmak için T süre harcanmıştır.

3. 2 numaralı segman tıkanıklık yüzünden aradaki yönlendiricinin (baz istasyonu) kuyruğundan çöpe gidiyor. Bu durumda alıcıya ulaşan 1 ve 3 numaralı segmanlar arasında yaklaşık T süre fark vardır.

Bu bilgiler ışığında yazarlar aşağıdaki heuristic'i oluşturuyorlar :

- T_{min} minimum "inter-arrival time" olsun.
- P_o alıcı tarafından alınan sırası bozuk segman. (out of order packet)
- P_i , P_o dan önce alınmış en son sırası düzgün segman.
- T_g, , P_i , P_o segmanlarının varışları arasındaki zaman farkı.
- n , P_i , P_o arasındaki kayıp segman sayısı olsun.

$(n+1)T_{\min} \le T_g < (n+2)T_{\min}$ ise n tane kayıp segman, kablosuz ortamda bozulmuş kabul edilir. Aksi halde tıkanıklık yüzünden aradaki yönlendiricilerde çöpe atılmışlar kabul edilir.

Bahsedilen yöntemin iyi sonuçlar verebilmesi için bağlantının geçtiği son "hop" telsiz link olmalı ve telsiz linkin bant genişliği diğer linklere (bağlantının geçtiği telli linkler) göre az olmalıdır.

2.2.6 Snoop Protokolü

Snoop protokolü, telsiz ortamlarda TCP'nin veri iletim başarımını artırmak için tasarlanmıştır [8]. Protokolün ana düşüncesi, baz istasyonunda TCP segmanlarını ön belleğe almak ve telsiz link üzerinde kayıp sezildiği anda kaynağın haberi olmadan kaybolan segmanları yeniden telsiz linke göndermektir.

Snoop protokolü, uçlarda çalışan TCP birimlerinin güncellenmesini gerektirmez. Baz istasyonuna yönlendirme katmanından önce eklenecek bir snoop ajanı telsiz ortamlarda TCP'nin yaşadığı başarım problemini çözmeye yeter.

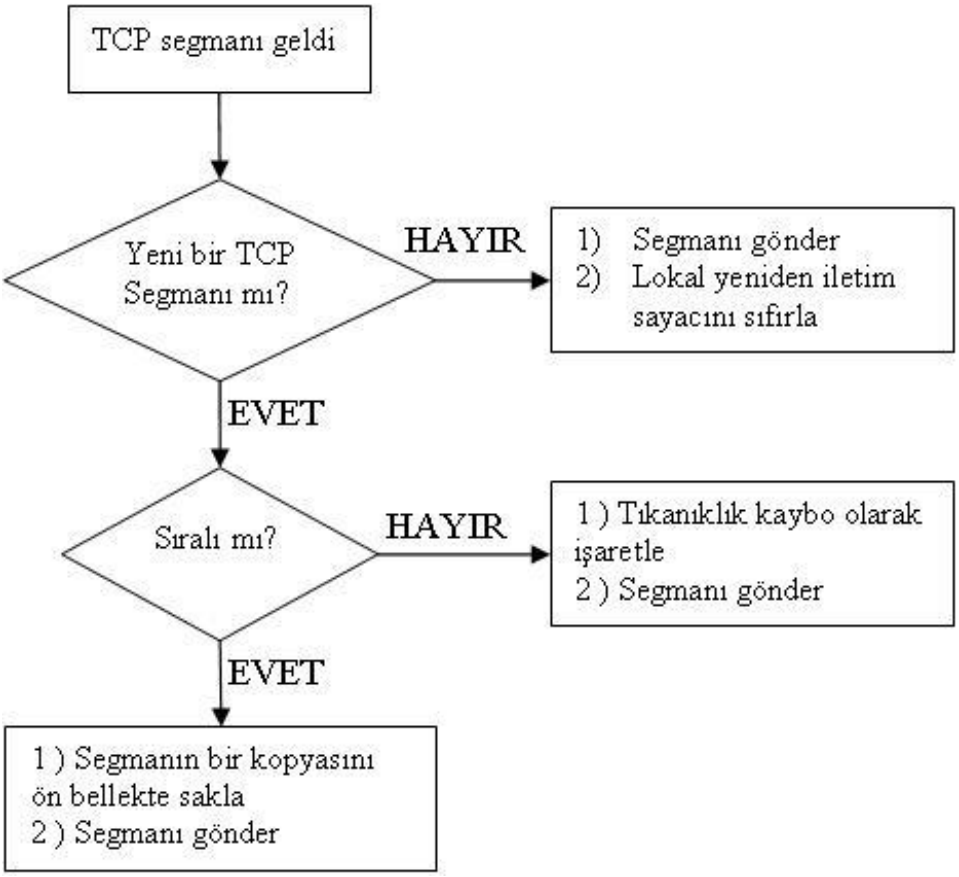
Snoop Protokolünün Çalışma Prensibi:

Sabit bir konaktan hareketli bir konağa, baz istasyonu üzerinden veri iletimi olduğu durumda, TCP segmanları snoop ajanı üzerinden geçecektir. Snoop ajanı, üzerinden geçen her TCP bağlantısı için ayrı bir tampon alan tutar ve bu tampon alanda kaynağa gönderilen fakat alındısı gelmeyen TCP segmanlarını saklar. Snoop ajanı sabit TCP kaynağından bir TCP segmanı aldığı zaman şekil 2.15'te görülen snoop_data algoritmasını çalıştırır. Snoop ajanı sıralı bir TCP segmanı aldığı zaman bu segmanın bir kopyasını önbelleğinde saklar ve segmanı telsiz linke iletir. Snoop ajanına gelen segman yeni bir TCP segmanı değilse bu segman kaynak tarafından yeniden gönderilen bir segmandır. Bu durumda snoop ajanı segmanı telsiz linke iletir ve ajan içinde tutulan lokal yeniden iletim sayacı sıfırlanır. Snoop ajanı yeni verinin hedefe ulaştığını haber veren alındı alırsa, ön belleğinden, alındısı gelen TCP segmanını siler ve alındıyı kaynağa gönderir.

Snoop ajanı, TCP segmanının hedefe ulaşmadığını iki durumda fark eder.

1. Gönderilen TCP segmanı için tekrarlanan alındı geldiği durum.
2. Snoop ajanının lokal RTO süresi boyunca yeni bir alındı alamadığı durum.

Snoop ajanı, kaybolan TCP segmanını, ön belleğinde mevcut ise yeniden telsiz linke iletir. Böylece TCP kaynağının telsiz linkte meydana gelen bu kayıptan haberi olmaz. TCP kaynağının gereksize yere Fast Retransmit ya da Slow Start algoritmalarını çalıştırıp debisini kısması önlenmiş olur.



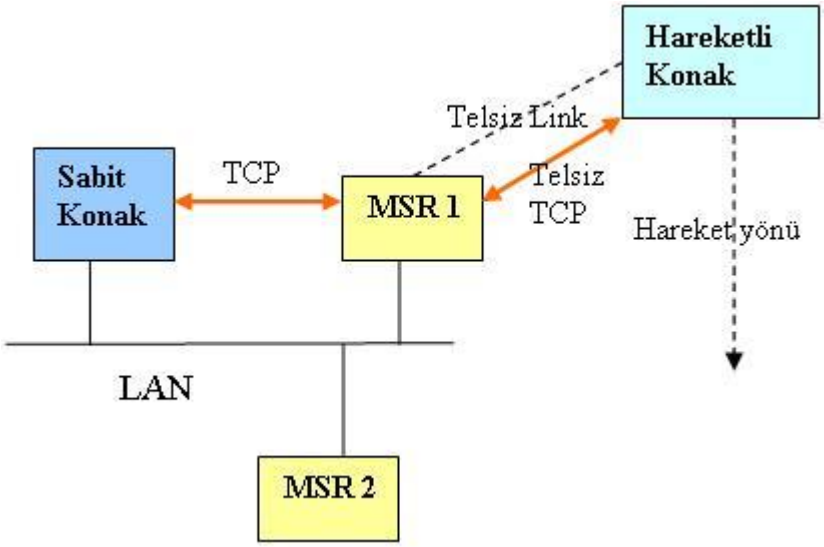
Şekil 2.15: Snoop agent snoop_data() prosedürü.

TCP kaynağının sabit, TCP alıcısının telsiz ortamda hareketli olduğu durumlarda, snoop protokolünün TCP veri iletim başarımına büyük katkı sağladığı simulasyonlar ile görülmüştür[9]. Fakat kaynağın telsiz ortamda bulunduğu durumlarda baz istasyonuna yerleştirilen snoop ajanı uçlarda çalışan TCP versiyonlarına ilk durumdaki kadar yardımcı olamaz. Bunun sebebi TCP segmanlarının telsiz link üzerinde, snoop ajanına gelmeden kaybolmasıdır. Snoop ajanı kendisine gelmeyen (telsiz link üzerinde kaybolan) TCP segmanlarını SACK[19] kullanarak kaynaktan tekrar isteyebilir. Fakat Snoop'un bu özelliği kaynakta SACK destekleyen bir TCP'nin çalışıyor olmasını gerektirir.

2.2.7 Indirect TCP

Indirect-TCP (I-TCP), telsiz ortamlarda TCP'nin karşılaştığı problemleri, TCP bağlantısını iki farklı bağlantıya ayırarak çözmeye çalışır. Bu bağlantılardan birincisi TCP kaynağı ve baz istasyonu arasında, ikincisi de baz istasyonu ve TCP alıcısı arasında kurulur. TCP kaynağının ve TCP alıcısının bu ara bağlantılardan haberi

yoktur. TCP kaynağı ve hareketli konağa hizmet veren baz istasyonu arasında normal bir TCP bağlantısı kurulur. Baz istasyonu ve hareketli konak arasında telsiz linkten haberdar olan, telsiz ortamda çalışabilecek şekilde tasarlanmış bir TCP versiyonu kullanılır. I-TCP yönteminde bağlantıyı ikiye ayıran baz istasyonuna MSR (Mobility Support Router) adı verilmiştir.



Şekil 2.16: I-TCP ve MSR

I-TCP yöntemi, TCP'nin uçtan uca çalışma mantığına aykırı bir yöntemdir. TCP bağlantısı iki ayrı bağlantıya bölünmüştür. MSR'larda (baz istasyonları) üzerlerinden geçen bütün TCP bağlantıları ile ilgili durum bilgisi tutulmak zorundadır. Bu da bağlantı sayısı arttıkça I-TCP yönteminin ölçeklenebilirliğini azaltmaktadır.

I-TCP yönteminde, bağlantının öteki ucunda (telsiz linkin öteki ucunda) bulunan konağın hareketli olabileceği ve baz istasyonu değiştirebileceği de düşünülerek TCP bağlantısının durumunun (TCP session) bir MSR'dan öteki MSR'a nasıl taşınacağı da açıklanmıştır. Şekil 2.16'da I-TCP'nin kullanılabileceği tipik bir topoploji görülmektedir.

3. KAYIP AYIRT EDİCİ TCP

Uygulamalara güvenilir ve doğru şekilde veri taşıma servisi sunan TCP, gerçek zamanlı bilgilerin taşınması için uygun bir dördüncü katman protokolü değildir. Ses ve görüntü iletimi ve gerçek zamanlı oyunlar için TCP yerine gecikmeye ve gecikmeler arasındaki farka (jitter) daha duyarlı protokoller kullanılmalıdır. Gerçek zamanlı veriler hedefe zamanında ulaştırılamazsa hiç bir değer taşımazlar. Bu verilerin TCP'de olduğu gibi yeniden transfer edilmesi bir fayda sağlamaz. Bu sebeplerden dolayı gerçek zamanlı verileri taşırken UDP [11] ya da RTP[12] (UDP üzerinde) kullanılır.

RTP'de, TCP'de olduğu gibi bir tıkanıklık kontrolü yoktur. Haberleşme ağında farklı protokolleri kullanan uygulamalar arasında haksızlığa yol açmamak için güvenilir olmayan (unreliable) trafiği taşımada kullanılan protokoller, TCP-friendly tıkanıklık kontrolü algoritmaları ile birlikte kullanılır. TFRC [13], TCP-friendly tıkanıklık kontrolü sağlayan bir yöntemdir ve RTP ile birlikte kullanılabilir.

Tıkanıklık kontrolü yöntemlerinin ortak sorunu, kaybolan segmanların tıkanıklık yüzünden mi yoksa telsiz linklerde meydana gelen bozulmalar yüzünden mi kaybolduğunu tam olarak ayırt edememeleridir. TCP de TFRC de kaybolan her segman için tıkanıklık kontrolü algoritmalarını çalıştırıp, veri transfer debisini kısmaktadır. Bu durum segmanların tıkanıklık dışında bir sebepten kaybolduğu durumda veri iletim başarımında kayıplara sebep olmaktadır.

"AED: An Accurate and Explicit Loss Differentiation Mechanism" başlıklı makalede [14] yazarlar RTP paket kayıplarının kaybolma sebeplerini ayırt etmeye yarayan bir yöntemden bahsediyorlar. Bu yöntemi kullanarak RTP ile birlikte çalışan TFRC'ye paketlerin hangi sebepten dolayı kaybolduğunu bildirmeyi amaçlıyorlar. Böylece TFRC paket kaybını anladığı anda, yerine göre debisini kısacak, yerine göre debisini kısmadan kaybolan paketin yeniden iletilmesini sağlayacak.

Bu tez çalışmasında:

- [14]'de bahsedilen "kayıp ayırt etme yöntemi"ni baz alarak yeni bir yöntem geliştirildi.

- Kaybolan segmanların hangi sebeplerden dolayı kayboldukları anlaşıldıktan sonra TCP Reno'dan türetilen yeni bir TCP versiyonu, Kayıp Ayırt Edici TCP, geliştirildi.

Detayları aşağıda anlatılacak olan yeni yöntem, telsiz linkler üzerinden de geçen TCP bağlantısının, veri iletim başarımını artırmayı amaçlamaktadır.

3.1 Yeni Yöntem ve Ajanlar

Telsiz linklerde bozulararak kaybolan segmanları ve tıkanıklık yüzünden aradaki yönlendiricilerde düşürülen segmanları ayırt edebilmek için TCP bağlantısının geçtiği yol üzerine ajanlar yerleştiriliyor. Bu tez çalışmasında bahsedilen yeni yöntem ara düğümlere yerleştirilen ajanlardan yararlanmaktadır.

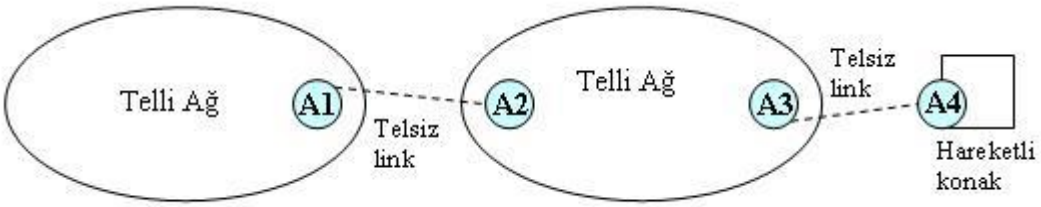
Ajanlar, üzerlerinden geçen paketlerin içindeki TCP segmanlarını incelerler, sekans numarasına bakarak hangi segmanların kaybolduğunu anlayabilirler. Şekil 3.1'de görüldüğü gibi ajan üzerinden geçen segmanların sekans numaralarını inceleyerek 5 sekans numaralı segmanın üzerinden geçmediğini anlayabilir.

Şekil 3.1: Ajan üzerinden geçen segmanların sekans numaraları

Ajanların görevi kaybolan segman bilgilerini bir şekilde kaybolmayan segmanlara yazmaktır ve bu bilginin hedefteki konağa kadar ulaşmasını sağlamaktır.

3.2 Ajanların Konumlandırılması

TCP segmanlarını inceleyip güncellemekle görevli olan ajanlar telsiz linklerin başına ve sonuna yerleştiriliyor. Telsiz ortamdaki bir konakta bulunan bir ajan da telsiz link sonunda konumlandırılmış bir ajan gibi düşünülebilir. Telli ve telsiz linkler barındıran heterojen bir haberleşme ağında ajanların konumlandırılması Şekil 3.2'de görülmektedir.



Şekil 3.2: Ajanların konumlandırılması

Telsiz link başında konumlanmış bir ajan (Şekil 3.2'deki A1 ajanı) kendisinden önceki linklerde tıkanıklık yüzünden kaybolmuş segmanları tespit edebilir. Telsiz link sonunda konumlanmış bir ajan (Şekil 3.2'deki A2 ajanı) kendisinden önceki telsiz linkler üzerinde bozulmaya uğrayıp kaybolmuş segmanları tespit edebilir.

3.3 Geçmiş Penceresi

TCP bağlantısının geçtiği yol üzerinde telsiz linklerin başına ve sonuna yerleştirilen ajanlar, TCP segmanlarını güncelleyerek, kayıp segman bilgilerini kaybolmayan TCP segmanlarına yazarlar. Ajanlar TCP segmanı içerisindeki Geçmiş Penceresi (GP) adı verilen bir alanı güncellerler. GP alanının TCP segmanının neresinde olduğu, protokolün uygulama detayları (implementation) bölümünde anlatılacaktır.

Yeni yönteme göre her segman kendisinden önce gelen N tane segman için geçmiş penceresinde durum bilgisi tutar. Durum bilgisi 3 farklı değer alabilir:

1. Segman kaybolmamıştır. (N)
2. Segman tıkanıklık yüzünden kaybolmuştur. (C)
3. Segman telsiz link üzerinde bozulmaya uğrayıp kaybolmuştur. (W)

6 sekans numaralı segmanın N=4 için taşıdığı geçmiş penceresi tablo 3.1'de görülmektedir.

Tablo 3.1: Geçmiş Penceresi

Sekans numarası	**5**	**4**	**3**	**2**
Durum Bilgisi	**C**	**W**	**N**	**N**

Tablo 3.1'e göre, 6 sekans numaralı segman içerisinde 5 sekans numaralı segmanın tıkanıklık yüzünden, 4 sekans numaralı segmanın telsiz link üzerinde bozulmalar yüzünden kaybolduğu bilgisi vardır. 2 ve 3 sekans numaralı segmanlar kaybolmadan ajan üzerinden geçmiştir.

Geçmiş penceresinde tutulan 3 farklı durum bilgisini bir bit (0,1) ile gösterilebilir. Geçmiş penceresinde yazan değerler ajanların konumlarına göre farklı anlamlar taşır.

Telsiz link başında konumlanmış bir ajan için:

- 1 : Telsiz linkte bozulmaya uğrayıp kaybolan segmanı
- 0 : Tıkanıklık yüzünden kaybolan segmanı ya da kaybolmayan segmanı belirtir.

Telsiz link sonunda konumlanmış bir ajan için:

- 1 : Tıkanıklık yüzünden kaybolan segmanı
- 0 : Telsiz linkte bozulmaya uğrayıp kaybolan segmanı ya da kaybolmayan segmanı belirtir.

Örnek :

Tablo 3.2: 6 sekans numaralı segman için örnek geçmiş penceresi

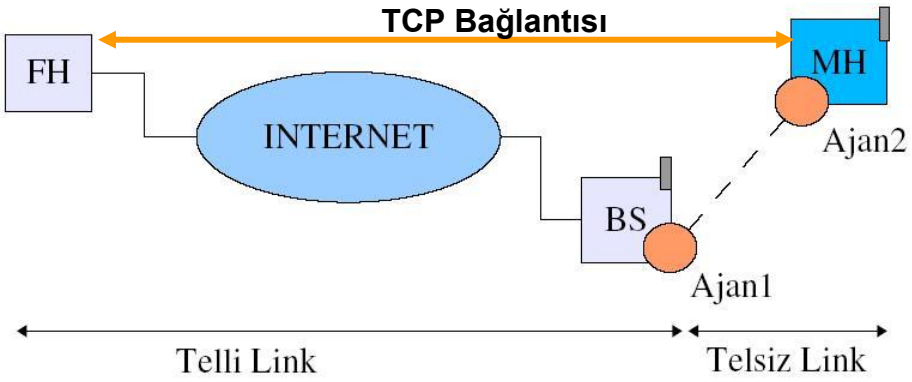
5	**4**	**3**	**2**
0	**1**	**0**	**0**

Telsiz link başında konumlanmış bir ajan üzerinden sırasıyla 1,2,3,6,7 numaralı segmanlar geçmiş olsun. 4 ve 5 numaralı segmanlar ajan üzerinden geçmiyorlar. Ajan, 6 sekans numaralı segmanın içindeki geçmiş penceresini tablo 3.2'deki gibi görüyor. 5 numaralı segmanı temsil eden bit değeri 0 olduğu için ajan bu segmanın tıkanıklık yüzünden kaybolduğunu anlıyor. 4 numaralı segmanı temsil eden bit değeri 1 olduğu için bu segmanın da telsiz link üzerinde kaybolduğunu anlıyor.

Ajan telsiz link sonunda konumlanmış olsaydı, 5 numaralı segmanın telsiz link üzerinde kaybolduğunu, 4 numaralı segmanın tıkanıklık yüzünden kaybolduğunu anlayacaktı.

3.4 Geçmiş Penceresinin Güncellenmesi

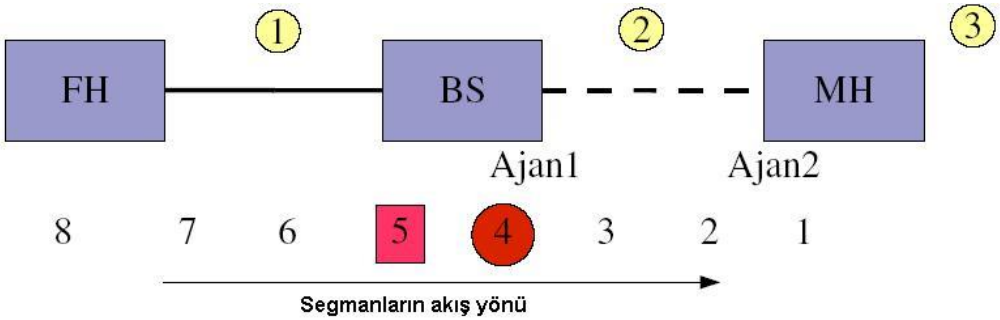
Telsiz link başındaki bir ajan tıkanıklık yüzünden kaybolmuş segmanları tespit edebildiği için bu ajana daha önceden telsiz linklerde kaybolmuş segmanları bildirmek gerekir. Aynı şekilde telsiz link sonundaki bir ajan telsiz link üzerinde kaybolmuş segmanları tespit edebildiği için bu ajana da tıkanıklık yüzünden kaybolmuş segmanları bildirmek gerekmektedir.



Şekil 3.3: Telli ve telsiz link topoloji

Ajanların geçmiş pencerelerini güncellemelerinin bir örnek ile açıklanması:

Örnek şekil 3.3'deki basit topoloji üzerinde açıklanacaktır. Sabit bir konaktan (FH) hareketli bir konağa (MH) doğru veri akışının olduğu varsayılıyor. FH ve MH arasında uçtan uca çalışan TCP bağlantısı kurulmuş olsun. TCP segmanları telli linklerden geçip baz istasyonu (BS) üzerinden telsiz linke aktarılırlar. Telsiz linkten de MH'ye ulaşırlar. Bu senaryoda iki tane ajandan yararlanılıyor. Ajan1 baz istasyonunda yani telsiz link başında , ajan2 hareketli konakta yani telsiz link sonunda konumlanmış olsun.



Şekil 3.4: Telli ve telsiz link topoloji, segmanların akış yönü

FH'dan MH'a sırasıyla 1,2,3,4,5,6,7,8 numaralı segmanlar iletilmek isteniyor. 5 numaralı segman telli link üzerinde tıkanıklık yüzünden kaybolmuş olsun. 4 numaralı segman da Şekil 3.4'deki gibi telsiz link üzerinde kaybolmuş olsun.

Segmanlar sabit konaktan çıktıklarında segmanların üzerlerinde taşıdıkları geçmiş pencereleri şekil 3.5'deki gibidir.

Seq No	8	7	6	5	4	3	2	1
Geçmiş Penceresi	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Şekil 3.5: Geçmiş Pencerereleri-1

Segmanlar kaynaktan çıktıklarında geçmiş pencerelerindeki bütün değerler ilk değer olan 0'dır. 5 sekans numaralı segman baz istasyonuna gelmeden, sabit konak ve baz istasyonu arasındaki bir yerde tıkanıklık yüzünden kayboluyor. Baz istasyonu üzerinden sırasıyla 1,2,3,4,6,7,8 sekans numaralı segmanlar geçiyor. Baz istasyonunda bulunan ajan1, 5 numaralı segmanın üzerinden geçmediğini farkeder ve 6,7,8 numaralı segmanların geçmiş pencerelerini Şekil 3.6'daki gibi günceller.

Seq No	8	7	6	5	4	3	2	1
Geçmiş Penceresi	0	0	1	0	0	0	0	0
	0	1	0	0	0	0	0	0
	1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Şekil 3.6: Geçmiş Pencerereleri-2

6,7 ve 8 numaralı segmanların geçmiş pencerelerinde 5 numaralı segmanı temsil eden alandaki bit değeri 1 yapılır. Böylece ajan1'den sonra gelecek olan ajan2' ye tıkanıklık yüzünden kaybolmuş segmanlar bildirilmiş olur. (ajan2 telsiz link sonuda bulunuyor ve sadece telsiz link üzerinde kaybolan segmanları tespit edebiliyor.)

Segmanlar baz istasyonu üzerinden telsiz linke transfer edildikten sonra 4 sekans numaralı segman telsiz link üzerinde bozulmaya uğrayıp kayboluyor. Bu durumda ajan2 üzerinden sırasıyla 1,2,3,6,7,8 numaralı segmanlar geçer. Ajan2, 6 numaralı segman üzerinde taşınan geçmiş penceresini tablo 3.3'deki gibi görür.

Tablo3.3: 6 sekans numaralı segmanın geçmiş penceresi

5	**4**	**3**	**2**
1	**0**	**0**	**0**

Ajan2, telsiz link sonunda bulunduğu için ve 4 ve 5 numaralı segmanlar ajan2 üzerinden geçmediği için bu geçmiş penceresini şu şekilde yorumlar:

- 4 numaralı segman telsiz linkte kaybolmuş
- 5 numralı segman tıkanıklık yüzünden daha önceki linklerde kaybolmuş.

Ajan2 segmanları hareketli konakta çalışan TCP'ye vermeden önce geçmiş pencerelerini şekil 3.7'deki gibi günceller.

Seq No	8	7	6	5	4	3	2	1
Geçmiş Penceresi	0	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	0
	0	1	0	0	0	0	0	0
	1	0	0	0	0	0	0	0

Şekil 3.7: Geçmiş Pencereleri-3

Hareketli konakta çalışan TCP yazılımı 6,7 ve 8 numaralı segmanların geçmiş pencerelerine bakarak 4 numaralı segmanın telsiz link üzerinde bozulmaya uğrayıp kaybolduğunu anlar ve kaybolan 4 numaralı segman için klasik tıkanıklık kontrolü

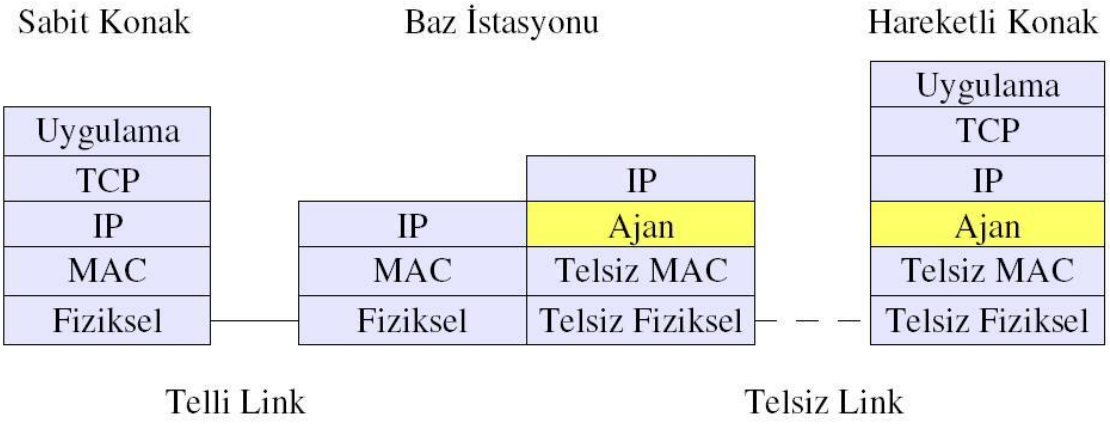
algoritmalarını çalıştırmaz. **K**ayıp **A**yırt Edici **TCP**'nin telsiz link üzerinde kaybolan bir segman tespit etmesi durumunda nasıl davranacağını ilerideki bölümlerde anlatılacaktır..

3.5 Ajanların TCP/IP Protokol Yığınındaki Yeri

TCP segmanlarını güncellemekle görevli ajanlar protokol yığınında 2. katman ve 3. katman arasına yerleştiriliyor. TCP segmanları internet ortamında IP datagramları içerisinde taşınır. Şekil 3.8'de görülen IP datagramında, IP başlığında TCP segmanının nerede başladığı bilgisi bulunur.

Şekil 3.8: IP datagramı içerisinde TCP Segmanı

Paketler yönlendirilmeden önce ajanlar IP datagramları içindeki TCP segmanlarını bulup, TCP segmanı içindeki geçmiş penceresi alanına ulaşabilirler. Şekil 3.9'de ajanların protokol yığınındaki yerleri gösterilmektedir.



Şekil 3.9: Ajanların protokol yığınındaki yeri

Telsiz linklerde kaybolan segmanları, tıkanıklık yüzünden kaybolan segmanlardan ayırdıktan sonra kaynak ve hedefte çalışan TCP yazılımlarında ne gibi değişiklikler yapılması gerektiği bir sonraki bölümde anlatılmaktadır.

3.6 TCP'de Yapılan Değişiklikler

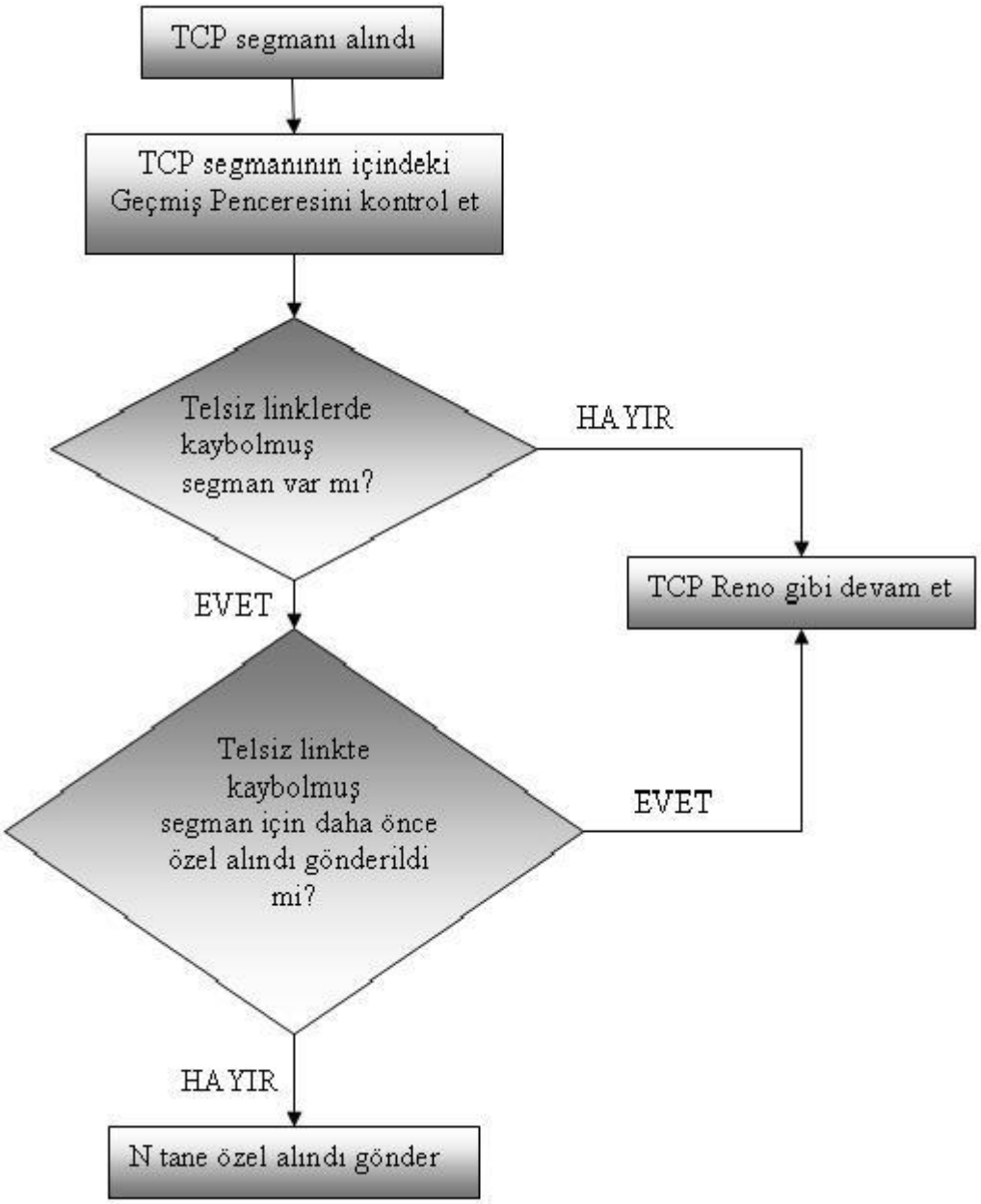
Bu bölümde kaynak ve hedef TCP varlıklarında yapılması gereken güncellemelerden bahsedilecektir.

3.6.1 Veri Segmanı İşleme Algoritması

TCP protokolünde hedef kaynağa segmanları aldığını alındılar (ACK) ile bildirir. Hedef en son aldığı segmanın sekans numarasını ACK segmanı içerisine yazar ve kaynağa gönderir. TCP'de ACK'ler kümülatiftir. N sekans numaralı bir ACK N'nin ve kendisinden önce gelen (N-1, N-2, N-3 ...) bütün segmanların alındığını onaylar. ACK'ler sayesinde kaynak gönderdiği segmanların hedefe ulaştığını anlayabilir ve yeni segmanlar transfer edebilir.

Hedefte çalışan TCP, segmanlardan birisinin telsiz linkler üzerinde kaybolduğunu anladığında bu durumu kaynağa bildirmekle görevlidir. Telsiz link kayıplarını hedef, kaynağa özel ACK'ler göndererek bildirir. Özel ACK'ler telsiz linkte kaybolan segmanın sekans numarasını taşırlar. Özel ACK'lerin nasıl hazırlandığı protokolün uygulama detayları (implementation) bölümünde anlatılacaktır. Hedefte çalışan Kayıp Ayırt Edici TCP, segman aldıkça şekil 3.10'da açıklandığı gibi davranır.

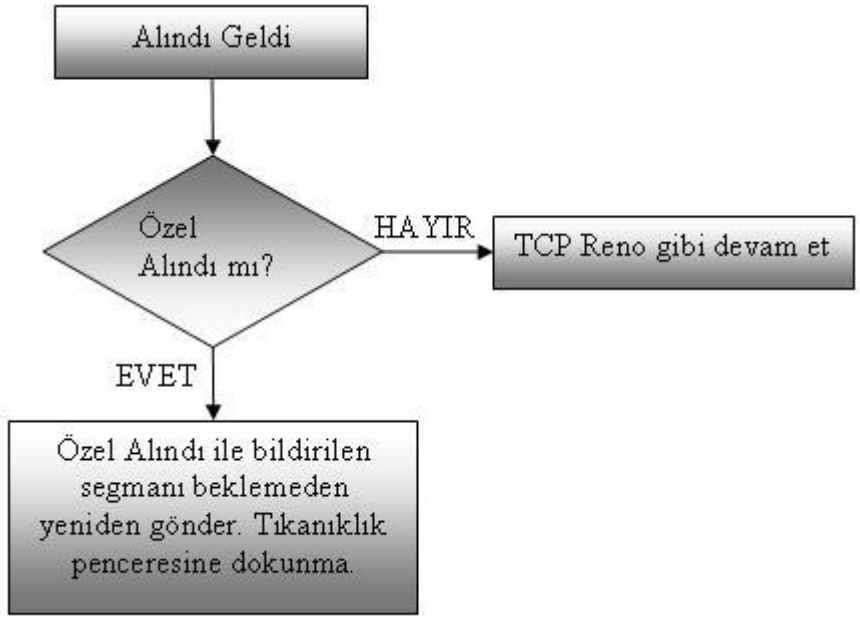
Hedef tarafından hazırlanan özel alındılar da hata oranı yüksek telsiz linkler üzerinde kaybolabilirler. Alındıların kaybolması durumunda kaynağa telsiz linkte kaybolan segman bilgileri gönderilemez. Bu problemi çözmek için hedef aynı özel alıdıdan birden fazla hazırlayıp, kaynağa gönderiyor. Kaç tane özel alındı gönderileceği parametre olarak TCP yazılımına verilebilir. Simulasyon sonuçlarına göre 3 adet özel alındı, alındıların kaybolma problemini çözmeye yetmektedir.



Şekil 3.10: Hedef'te çalışan KA-TCP'de veri segmanı işleme

3.6.2 Alındı İşleme Algoritması

Kaynak özel alındı almadığı durumlarda TCP Reno gibi davanır. Kaynak transfer ettiği bir segman için özel ACK alırsa, bu segmanın hedefe ulaşmadığını ve telsiz link üzerinde kaybolduğunu anlar. Özel alındı alan kaynak tıkanıklık kontrolü algoritmalarını çalıştırmaz. Debisini kısmadan, tıkanıklık penceresi açıklığı yeterliyse özel alındı ile bildirilen segmanı yeniden ve beklemeden transfer eder. Alındı işleme algoritması şekil 3.11'de gösterilmektedir.



Şekil 3.11: Kaynakta çalışan KA-TCP'de alındı işleme

3.7 Protokol Gerçekleme Detayları

Bu bölümde, tasarlanan Kayıp Ayırt Edici TCP'nin mevcut TCP segman yapısı kullanılarak nasıl gerçekleneceği anlatılacaktır.

3.7.1 TCP Checksum Güncellemesi

TCP, kendisinden hizmet alan uygulamalara güvenilir bir hizmet vermektedir. TCP kaynaktan hedefe giden TCP segmanlarının bozulmadan, doğru hedefe gittiğini checksum hesabı ile kontrol eder. Transfer edilen her segmana checksum bilgisi yazılır ve hedefte bu checksum kontrol edilir. Checksum kontrolünden geçemeyen hasarlı segmanlar çöpe atılır.

Checksum alanı başlık ve metin de dahil olmak üzere segmandaki tüm 16-bit kelimelerin 1`e tümlenmiş bir toplamını içerir. TCP'de UDP`nin kullandığına benzer bir sözde-başlık kullanır. Bu sözde başlık (pseudo header) içinde kaynak ve hedefin IP adresleri vardır.

Ajanlar, üzerlerinden geçen segmanların geçmiş pencerelerini güncellerse checksum'ı bozmuş olacaklar ve hedefte checksum bozuk olarak alınan segmanlar çöpe atılacak. Ajanların segmanı, checksum'ı değiştirmeden güncellemesini sağlayacak bir yönteme ihtiyaç duyulmaktadır.

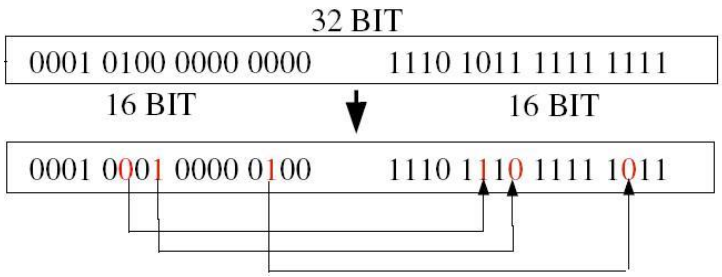
Geçmiş penceresi, TCP segmanı içerisinde opsiyonlar alanına yerleştirilebilir. Şekil 2.2 ‘te TCP segman yapısı görülmektedir. Geçmiş penceresi opsiyon alanının ilk 16 biti içerisine yazılırsa, ikinci 16 biti geçmiş penceresinin bire tümleyeni olarak kullanılabilir. Böylece geçmiş penceresinde bir değişiklik yapıldığında (opsiyonlar alanındaki birinci 16 bit), sonra gelen 16 bitte de 16 bit toplamın değerini değiştirmeyecek şekilde değişiklik yapıp checksum’ın değişmemesi sağlanır.

Şekil 3.10’da bu yöntem ile ilgili bir örnek verilmektedir. Geçmiş penceresindeki 6. bit değeri 1’den 0’a güncellendiğinde bir sonraki 16 bit içerisinde karşılık gelen bit değeri 0’dan 1’e güncelleniyor. Böylece ilk başta hesaplanan checksum bozulmamış oluyor.

```
     0001 0100 0000 0000  → GEÇMİŞ PENCERESI
+    1110 1011 1111 1111  → 1’E TÜMLEYENI
-------------------------
     1111 1111 1111 1111  → TOPLAM 1’E EŞİT

Geçmiş Penceresi güncellendikten sonra.

     0001 0001 0000 0100  → GEÇMİŞ PENCERESI
+    1110 1110 1111 1011  → 1’E TÜMLEYENI
-------------------------
     1111 1111 1111 1111  → TOPLAM 1’E EŞİT
```

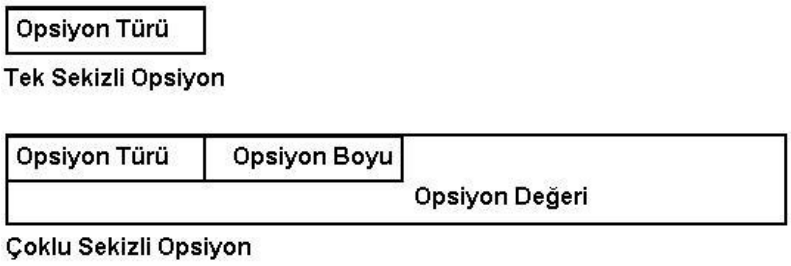


Şekil 3.12: TCP checksum güncelleme

3.7.2 TCP opsiyonları ve Özel Alındı Tanımlanması

Opsiyon alanı TCP`ye gelecekte yapılacak eklemeler düşünülerek tasarlanmıştır. IP datagramlarındaki opsiyon alanına benzer bir biçimde yapılandırılmıştır. TCP başlığı sonunda yer alan opsiyon alanı, TCP segmanı içerisinde 8 bit'in katları olarak bulunur. Checksum hesaplanırken TCP opsiyonları alanı da kullanılır. TCP opsiyonları şekil 3.13’de görüldüğü gibi 2 formatta olabilir:

- Tek bir sekizli, sadece opsiyon türü
- Bir sekizli opsiyon türü, bir sekizli opsiyon boyutu ve opsion değerini tutan sekizliler

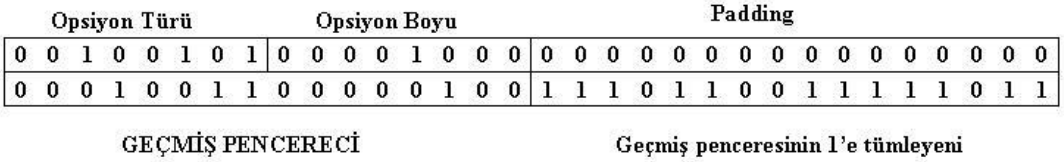


Şekil 3.13: TCP Opsiyonları

Kayıp Ayırt Edici TCP'yi gerçeklemek için TCP segmanındaki opsiyon alanını kullanabilir.

Opsiyon türü alanına daha önceden kullanılmamış bir opsiyon numarası verilebilir. Şekil 3.15'de değişik TCP versiyonları tarafından kullanılmakta olan TCP opsiyon numaraları listelenmektedir. Kayıp ayırt edici TCP için burada listelenmeyen bir numara opsiyon türü olarak seçilebilir.

Opsiyon türü 37 (Geçmiş Penceresi Opsiyonu) seçilirse 16 bitlik bir geçmiş penceresini taşıyan TCP segmanında opsiyon alanı şekil 3.14'deki gibi olur. Padding alanı opsiyon verisinin 32 bit sınırında başlaması için kullanılır ve 0 ile doldurulur.



Şekil 3.14: Geçmiş penceresi opsiyonu

Kind	Length	Meaning	Reference
0	-	End of Option List	[RFC793]
1	-	No-Operation	[RFC793]
2	4	Maximum Segment Size	[RFC793]
3	3	WSOPT - Window Scale	[RFC1323]
4	2	SACK Permitted	[RFC2018]
5	N	SACK	[RFC2018]
6	6	Echo (obsoleted by option 8)	[RFC1072]
7	6	Echo Reply (obsoleted by option 8)	[RFC1072]
8	10	TSOPT - Time Stamp Option	[RFC1323]
9	2	Partial Order Connection Permitted	[RFC1693]
10	3	Partial Order Service Profile	[RFC1693]
11		CC	[RFC1644]
12		CC.NEW	[RFC1644]
13		CC.ECHO	[RFC1644]
14	3	TCP Alternate Checksum Request	[RFC1146]
15	N	TCP Alternate Checksum Data	[RFC1146]
16		Skeeter	[Knowles]
17		Bubba	[Knowles]
18	3	Trailer Checksum Option	[Subbu & Monroe]
19	18	MD5 Signature Option	[RFC2385]
20		SCPS Capabilities	[Scott]
21		Selective Negative Acknowledgements	[Scott]
22		Record Boundaries	[Scott]
23		Corruption experienced	[Scott]
24		SNAP	[Sukonnik]
25		Unassigned (released 12/18/00)	
26		TCP Compression Filter	[Bellovin]

Şekil 3.15: TCP opsiyon numaraları

Kayıp Ayırt Edici TCP'de Özel Alındıları tanımlamak için de TCP segmanındaki opsiyon alanından yararlanılabilir. Özel Alındılar tanımlanırken birinci tipteki opsiyonlar kullanılabilir. Tek bir sekizliden oluşan bu opsiyonlar sadece opsiyon türünü belirtir. Opsiyon türü olarak 36 seçilirse hedef tarafından kaynağa gönderilen Özel Alındı içerisindeki opsiyon alanı şekil 3.16'daki gibi olur. Kaynak bir alındı aldığında bunun özel bir alındı (telsiz linkte bozulma habercisi) olduğunu opsiyon alanını kontrol ederek anlayabilir.

Şekil 3.16: Opsiyon türü değeri 36

4. KAYIP AYIRT EDİCİ TCP'NİN BENZETİM YOLUYLA SINANMASI

Tasarlanan TCP tıkanıklık denetimi yönteminin başarımını ölçmek için NS (Network Simulator) [15] yazılımı kullanıldı. NS, haberleşme ağları ile ilgili araştırmalarda kullanılmak üzere geliştirilmiş bir simulasyon yazılımıdır.

4.1 Benzetim Yoluyla Sınama Ortamı

Kayıp Ayırt Edici TCP'nin simulasyonunu NS ile yapabilmek için NS yazılımına yeni modüller eklendi.

- AedConnectionManager

  Baz istasyonu ve hareketli konaklar üzerinde Geçmiş Penceresi güncellemesi yapabilmek icin AedConnectionManager sınıfı yazıldı. Simulasyon sırasında hareketli konaklara ve baz istasyonlarına AedConnectionManager kuruluyor.

- AedTcpSinkAgent

  NS içerisinde hazır bulunan TcpSink'ten türetilen bu sınıf, TCP bağlantısında alıcı olan (hedef) TCP varlığını temsil ediyor. AedTcpSinkAgent sınıfı, aldığı TCP segmanlarının Geçmiş Pencerelerini okuyup, hangi segmanların telsiz linklerde kaybolduğunu anlamakla ve kaynağa haber vermekle görevlidir. Kaynağa haber vermek için Özel Alındı segmanları üretir.

- AedTcpSender

  AedTcpSender, NS içerisinde hazır gelen TcpRenoAgent sınıfından türetildi. AedTcpSender normal alındı aldığı durumlarda TCP Reno gibi davranır. Fakat alıcı tarafından (AedTcpSinkAgent) gönderilen Özel Alındılardan aldığı zaman bölüm 3.6.2 Alındı İşleme Algoritması'nda anlatıldığı gibi davranır.

- Yardımcı Sınıflar

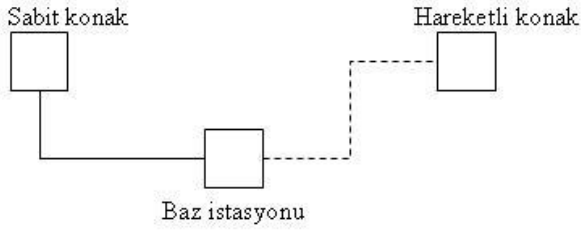
  - HistoryWindow: Bu sınıf TCP segmanı içerisinde saklanan Geçmiş Penceresi'ni temsil ediyor. Geçmiş Penceresi ile ilgili güncelleme işlemleri bu sınıfın metodları ile yapılıyor.

- AedData: Bu sınıf simulasyon içerisinde veri taşıyan TCP segmanlarını ve Özel Alındıları temsil ediyor.

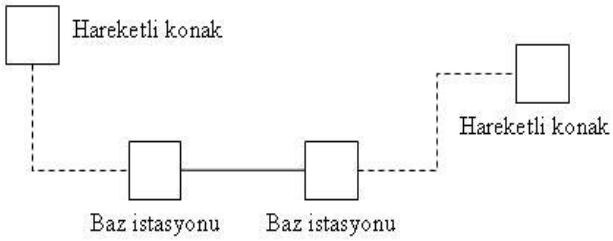
NS'e eklenen modüller ile simulasyonlar yapabilmek için TCL [16] betikleri yazıldı. TCL betikleri ile simulasyon topolojisi ve NS modüllerinin ayarlanması yapıldı.

4.2 Benzetim Yoluyla Sınama Topolojisi

Simulasyonlar yapılırken 2 tip topoloji kullanıldı. Birinci tip topolojide (Şekil 4.1) telli ortamda bulunan bir konak ile telsiz ortamda bulunan bir konak arasında, ikinci tip topolojide (şekil 4.2) ise telsiz ortamda bulunan iki konak arasında kurulan TCP bağlantısının simulasyonu yapıldı. Simulasyonlarda kullanılan haberleşme linklerinin kapasiteleri 10Mb ve her bir linkteki gecikme 50ms olarak seçildi.



Şekil 4.1: Simulasyon Topolojisi-1



Şekil 4.2: Simulasyon Topolojisi-2

Topolojileri gösteren şekil 4.1'de ve şekil 4.2'deki sürekli çizgiler telli linkleri, kesikli çizgiler telsiz linkleri göstermektedir. Kampüs içerisinde telsiz internet erişimi servisi alan bir dizüstü bilgisayarın internet üzerindeki bir FTP sunucusundan dosya indirmesinin simulasyonu şekil 4.1'deki topoloji ile yapılıyor. Şekil 4.2'daki topoloji, iki hareketli konak (2 cep telefonu) üzerinde TCP kullanan uygulamalar arasındaki veri iletiminin simulasyonu için kullanılmaktadır.

4.3 Benzetim Yoluyla Sınama Sonuçları

Kayıp Ayırt Edici TCP yönteminin veri iletim başarımını TCP NewReno[17] ve Snoop[8] protokolleri ile karşılaştırdık. Snoop ile yapılan simulasyonlarda kaynakta ve hedefte TCP SACK [19] kullanıldı.

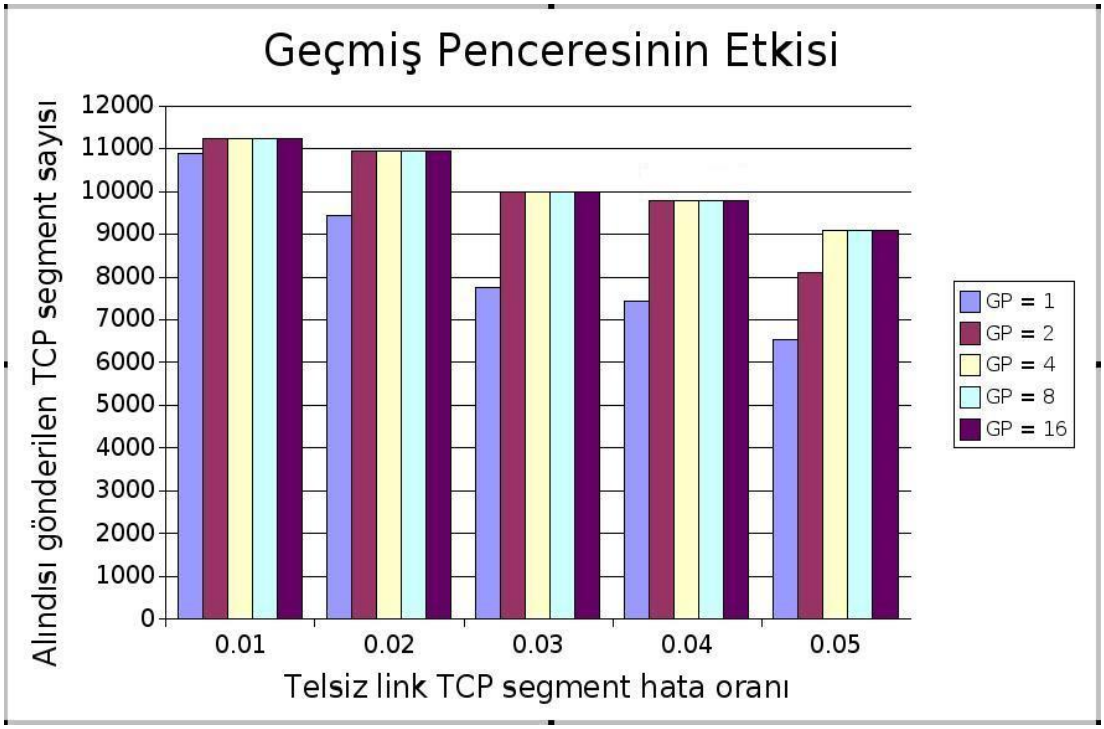
En uygun Geçmiş Penceresi boyutunun bulunması için Şekil 4.1'deki topoloji üzerinde aşağıda ayarları listelenmiş benzetim çalıştırıldı.

- Telsiz ortamda bulunan konakta ve telli ortamda bulunan konakta Kayıp Ayırt Edici TCP çalışmaktadır.
- Telsiz ortamda bulunan konakta ve baz istasyonunda Geçmiş Pencerelerini güncelleyen birer tane ajan bulunmaktadır.
- Telli link üzerinde TCP segman bozulma olasılığı 1/100000'dir. Alındı taşıyan TCP segmanları için segman bozulma olasılığı normal segmanlar için uygulanan bozulma olasılığının 0.04 katıdır. Bunun sebebi 1040 oktet uygulama verisi taşıyan TCP segmanlarına karşılık 40 oktet alındı taşıyan TCP segmanları kullanılmasıdır.
- Telli ortamda bulunan konaktan, telsiz ortamda bulunan konağa doğru veri akışı, ters yönde alındı akışı olmaktadır.
- Benzetim telsiz link üzerindeki hata oranı değiştirilerek 5 kez çalıştırılmıştır. Benzetim süresi olarak 200 saniye seçilmiştir. Telsiz link hata oranları sırasıyla 0.01, 0.02, 0.03, 0.04, 0.05 olarak ayarlanmıştır.
- Hedefte çalışan KA-TCP, telsiz linkte kaybolan TCP segmanları için 2 tane özel alındı göndermektedir.

1, 2, 4, 8 ve 16 bit Geçmiş Penceresi değerleri için, yukarıda anlatılan şartlarda telsiz link hata oranları değiştirilerek benzetim 5 defa çalıştırıldı ve sonuç olarak şekil 4.3'deki grafik elde edildi. Şekil 4.3'deki grafik, hadefe başarı ile iletilen TCP segmanlarının miktarını göstermektedir. Telsiz link hata oranı 0.01 iken ve geçmiş penceresi değeri 4 iken hedefe 11000'den fazla TCP segmanı iletilmiştir. Aynı şekilde telsiz link hata oranı 0.05 iken ve geçmiş penceresi değeri 2 iken hedefe yaklaşık 8000 segman iletilebilmiştir. Geçmiş Penceresi boyu 4, 8 ve 16 iken elde edilen sonuçlar örtüşmektedir. Peşpeşe 2'den fazla TCP segmanı telsiz linkler üzerinde kaybolursa, bu segmanların kaybolma bilgileri 2 bitlik bir geçmiş penceresiyle hedefe kadar iletelemez. Şekil'den de anlaşılacağı gibi telsiz linkte hata

oranının yüzde beş olduğu durumlarda 4 bitlik, 8 bitlik ve 16 bitlik Geçmiş Pencereleri aynı sonucları vermektedir. Şekil 4.3'den çıkartılacak sonuç, telsiz link hata oranı arttıkça kullanılacak geçmiş penceresi boyutunun da artırılması gerektiğidir.

Bundan sonra yapılan tüm simulasyonlarda Geçmiş Penceresi değeri 8 bit olarak seçilmiştir.

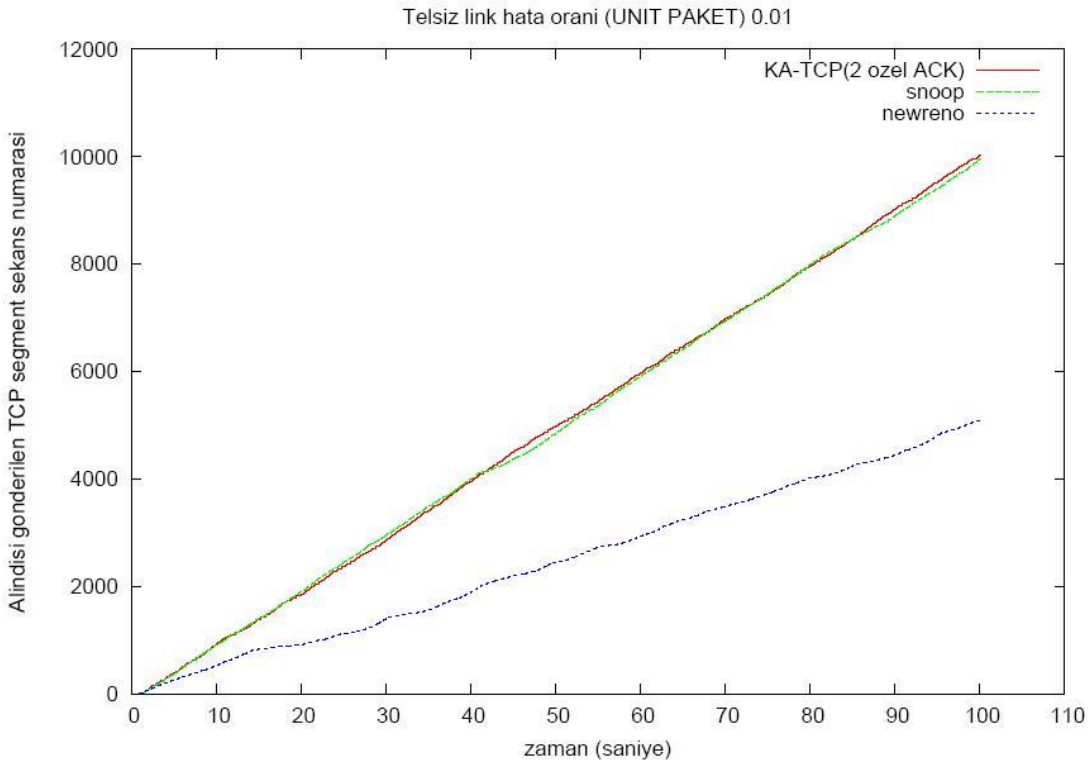


Şekil 4.3: Geçmiş Penceresi'nin Etkisi

Kayıp Ayırt Edici TCP, TCP NewReno ve Snoop protokollerinin başarımlarının benzetim yoluyla karşılaştırılması:

1. Kaynak telli ortamda, hedef telsiz ortamda

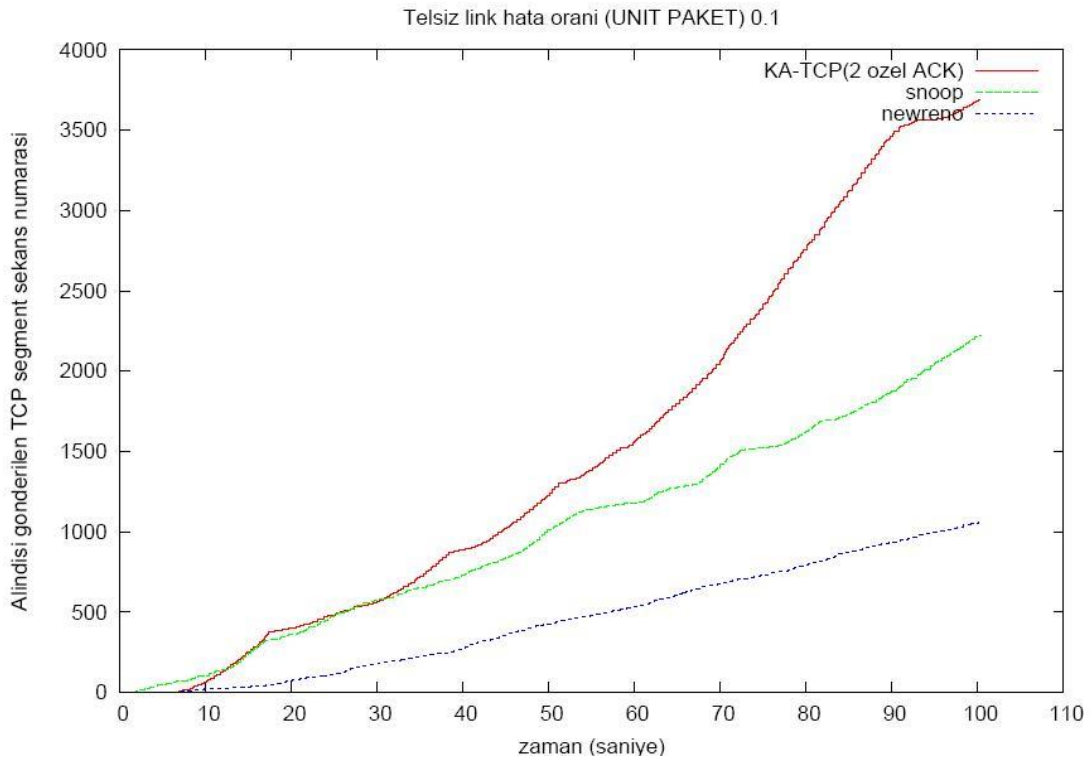
Şekil 4.1'deki topoloji üzerinde, TCP kaynağının telli ortamda bulunan konak üzerinde olduğu ve telsiz link hata oranının yüzde bir olduğu durumda, KA-TCP ve SNOOP yöntemlerinin veri iletim başarımları aynı çıkıyor. TCP NewReno telsiz linkler için tasarlanmadığı için veri iletim başarımı düşük olmuştur. Telsiz link üzerinde uygulanan paket bozulma olasılığı çok yüksek olmadığı için KA-TCP ve SNOOP arasında belirgin bir fark görülmemektedir.



Şekil 4.4: Kaynak telli ortamda, telsiz link hata oranı yüzde bir.

Şekil 4.4 TCP New Reno, Snoop ve Kayıp Ayırt Edici TCP'nin hedefe ilettiği TCP segmanlarının sekans numaralarının zamana bağlı olarak değişimini göstermektedir. Sonuç olarak telsiz link üzerindeki bozulma oranı az ise hangi telsiz linkler için özelleştirilmiş tıkanıklık denetimi yönteminin kullanılacağı önemli değildir.

Telsiz link üzerindeki bozulma oranı yüzde ona çıkartılırsa şekil 4.5'deki sonuc elde ediliyor.

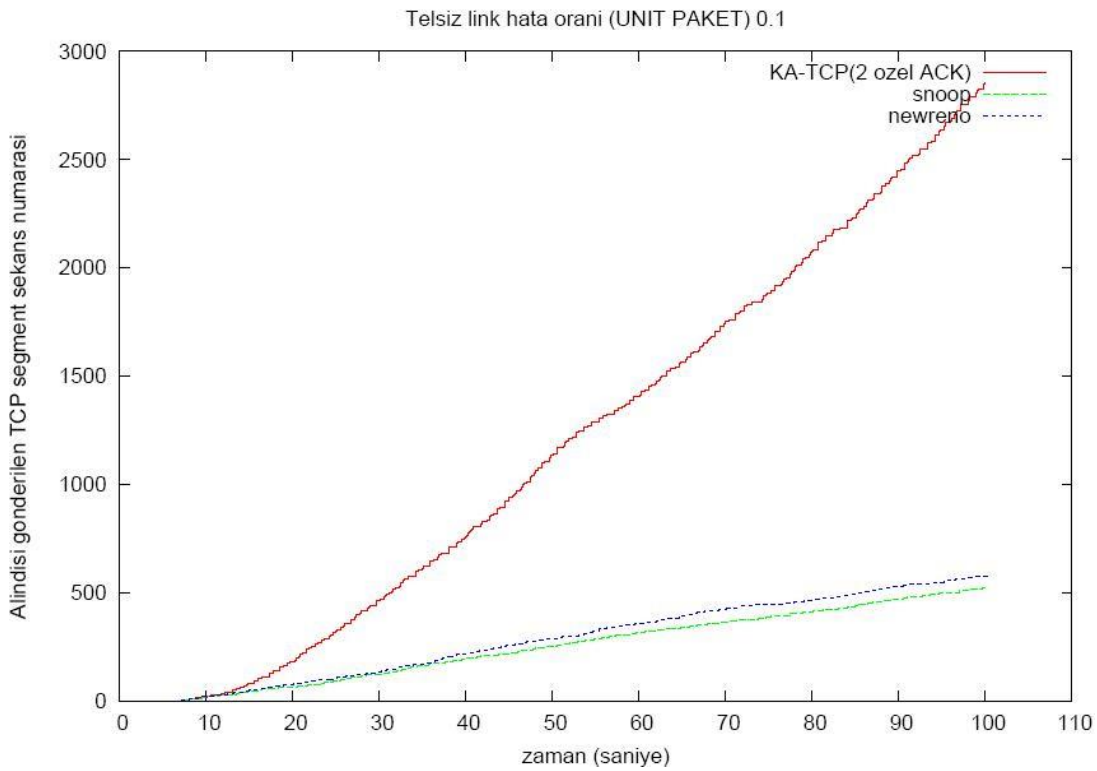


Şekil 4.5: Kaynak telli ortamda, telsiz link hata oranı yüzde on.

Hedefe en fazla TCP segmanı ileten yöntem 2 özel alındı kullanan Kayıp Ayırt Edici TCP olmuştur. İkinci ve üçüncü sıralarda sırasıyla SNOOP ve telli linkler için tasarlanmış TCP NewReno vardır. Bu sonuç beklenilen bir sonuçtur. Kayıp Ayırt Edici TCP telsiz link üzerinde kaybolan segmanları ayırt edebildiği için debisini kısmadan bu segmanların yeniden iletimini yapabiliyor. Snoop ve NewReno'da ise telsiz linkler üzerinde kaybolan segmanlar yeniden iletim sayaçlarını tetikleyerek kaynakların debilerini kısmasına yol açıyorlar.

2. Kaynak telsiz ortamda, hedef telli ortamda

Şekil 4.1'deki topolojide veri iletiminin telsiz ortamda bulunan konaktan telli ortamda bulunan konağa doğru olduğu durumda Kayıp Ayırt Edici TCP yine en iyi sonuçları veriyor. Fakat snoop protokolü kaynağın telsiz ortamda olduğu durumlarda istenilen veri iletim başarımını sağlayamıyor. Bunun sebebi bölüm 2.2.6 Snoop Protokolü'nde açıklanmıştır.

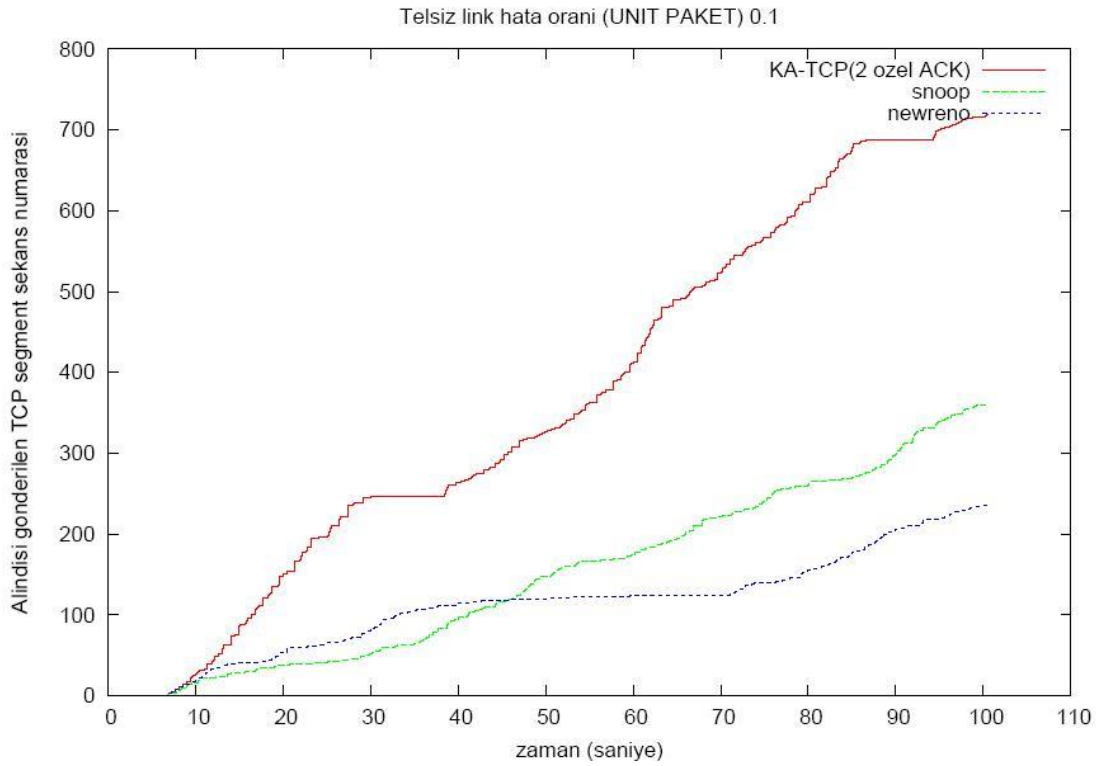


Şekil 4.6: Kaynak telsiz ortamda, telsiz hata oranı yüzde on.

Şekil 4.6'de görülen simulasyon sonucuna göre kaynağın telsiz ortamda olduğu durumda da beklenen sonuçlar alınıyor. Kayıp Ayırt Edici TCP yöntemi, Snoop ve TCP NewReno'ya göre daha fazla veri iletimi yapmış, TCP NewReno da Snoop'a göre daha fazla veri iletimi yapmıştır.

3. Kaynak ve hedef hareketli, şekil 4.2'deki topoloji

Simulasyonlar değişik topolojiler üzerinde denendi. Simulasyonlarda kullanılan ikinci tip topoloji, telsiz ortamda bulunan iki konak arasındaki TCP bağlantısının başarımını ölçmek için kullanılan şekil 4.2'deki topolojidir. Topolojideki iki telsiz linkin hata oranının 0.1 ve baz istasyonları arasındaki linkin hata oranının 0.00001 olduğu durumda simulasyon sonuçları şekil 4.7'deki gibidir.



Şekil 4.7: Kaynak ve hedef hareketli, telsiz link hata oranı yüzde sekiz.

Kaynak ve hedefin telsiz ortamda bulunduğu topolojide de simulasyon sonucları beklenildiği gibi olmuştur. En iyi sonucu 2 özel alındı gönderen Kayıp Ayırt Edici TCP en kötü sonucu da TCP New Reno vermiştir. Kayıp Ayırt Edici TCP kullanılarak, simulasyon süresince Snoop protokolünün taşıdığı TCP segman sayısınını iki katından daha fazla TCP segmanı taşınmıştır.

5. SONUÇLAR ve TARTIŞMA

Telsiz ortamlarda TCP tıkanıklık denetimi algoritmalarının yaşadığı problemleri çözmeyi amaçlayan bu tez çalışmasında Kayıp Ayırt Edici TCP (KA-TCP) adı verilen yeni bir teknik geliştirildi.

Telsiz linklerin temel özellikleri incelendi ve paket bozulma oranı yüksek olan telsiz linklerde mevcut TCP tıkanıklık denetimi algoritmalarının neden yetersiz kaldığı araştırıldı. Geliştirilen yeni tekniğin telsiz linkler için tasarlanmış Snoop protokolü ve telli linkler için tasarlanmış TCP NewReno ile karşılaştırılması yapıldı. Benzetim yoluyla karşılaştırma sonuçlarından da görüldüğü gibi Kayıp Ayırt Edici TCP tekniği segman kaybının sebebini öğrenebildiği için bahsedilen 2 teknikten de daha iyi sonuçlar vermiştir.

Tartışmalar:

IP datagramlarının şifrelenerek iletildiği durumlarda (IPSec) [19] ajanlar, IP datagramları içerisindeki TCP segmanı göremeyecekler ve Geçmiş Penceresini güncelleyemecekler. IP datagramlarının şifrelenmesi, KA-TCP metodunun düzgün çalışmasına bir engeldir.

TCP segmanları IP paketleri içerisinde taşındıkları için aynı bağlantıya ait TCP segmanları farklı yollar üzerinden dolaşarak hedefe ulaşabilirler. Teoride mümkün olan bu durum pratikte sık rastlanan bir durum değildir. Benzetimler sırasında TCP segmanlarını taşıyan IP datagramlarının hepsinin aynı yolu izleyerek hedef konağa ulaştığı varsayılabilir.

Ara düğümlerde konumlanmış ajanların TCP segmanları içerisindeki Geçmiş Penceresi alanını güncellemesi az da olsa bir gecikmeye sebep olacaktır. İstenmeyen bu gecikme RTT'de artış demektir. Fakat veri iletim başarımında elde edilen artış yanında segman güncelleme sırasında meydana gelen gecikme ihmal edilebilir. KA-TCP ajanlarına, Snoop protokolünde kullanılan ajanlardaki gibi akıl yüklenmiyor. Snoop ajanları TCP protokolü detaylarını bilmek zorundalar. KA-TCP ajanları sadece segman içindeki belirli bitleri otomatik olarak güncelledikleri için bilmeleri gereken tek şey Geçmiş Penceresi güncelleme tekniğidir. Geçmiş Penceresi

güncelleme, donanım seviyesinde yapılırsa ihmal edilebilecek kadar kısa sürelerde yapılacak bir işlemdir.

Kayıp Ayırt edici TCP diğer TCP versiyonlari ile de çalısabilecek şekilde tasarlanmıştır. Bağlantı başlangıcında KA-TCP opsiyonu etkin değilse KA-TCP, TCP-Reno gibi çalışmasına devam edecekektir. KA-TCP'yi kullanabilmek için ajanlar dışında, kaynak ve hedefte çalışan TCP varlıklarında güncelleme yapmak gerekmektedir.

KA-TCP'de hedeften kaynağa gönderilen özel alındılar haberleşme ağındaki kaynakları (bant genişliği gibi) kullandıkları için az da olsa tıkanıklığa sebep olabilirler. Fakat özel alındıları taşıyan IP datagramları düşük öncelikli olarak işaretlenirse, bu datagramlar tıkanıklık sezildiğinde yönlendiricilerde çöpe ilk atılacak paketler olacaktır. Böylece özel alındılar ile fazladan tıkanıklık yaratılmamış olacaktır.

TCP segmanlarını taşıyan IP datagramlarının parçalara ayrılarak (*fragmentation*) iletişim ağı üzerinde taşınması KA-TCP tekniğinin doğru çalışmasını engellemektedir. IP datagramının *n* parçaya bölünüp taşınması durumunda içerisinde taşıdığı TCP segmanı da *n* parçaya bölünmüş olacaktır. Bu durumda ajanlar üzerinden içerisinde TCP başlığı bulunmayan IP datagramları geçecektir. IP katmanında yapılan parçalara ayırma, segman kayıplarının sebebini ayırt etmede problemler yaratmaktadır. Benzetimlerde, TCP segmanını taşıyan IP datagramlarının parçalara ayrılmadan iletildiği varsayılıyor.

Gelecek Çalışmalar

Bu tez çalışmasında, KA-TCP tekniğinin telsiz ortamdaki başarımı, benzetim yoluyla Snoop ve TCP NewReno ile karşılaştırılmıştır. Geliştirilen yeni tekniğin Snoop ve TCP NewReno dışındaki diğer TCP versiyonları ile de karşılaştırılması gelecek çalışmalar arasındadır.

NS için yazılan modüllerin kolay kurulabilir hale getirilmesi ve KA-TCP modülünün NS içerisinde sınanması için test betiklerinin yazılması gerekmektedir.

TCP segmanlarının hedefe sırası bozuk geldiği durumlarda, hedefte çalışan ajanda tampon alan yönetimi yapılması da gelecek çalışmalar arasında yer almaktadır.

KAYNAKLAR

[1] **Van Jacobson**, 1988. Congestion Avoidance and Control, ACM 0-89791-279-9/88/008/0314

[2] **RFC 2001**, TCP Slow Start, Congestion Avoidence, Fast Retransmit, and Fast Recovery Algorithms

[3] **Tom Goff, James Moronski, D. S. Phatak**, **Vipul Gupta** Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments, IEEE Infocom 2000

[4] **R.K. Balan, B.P. Lee, K.R.R. Kumar**, **L. Jacob, W.K.G. Seah, A.L. Ananda** TCP HACK: a mechanism to improve performance over lossy links, Computer Networks 39 (2002), 347-361

[5] **RFC 793,** Transmission Control Protocol

[6] **Saad Biaz, Nitin H. Vaidya,** Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times At the Receiver, IEEE Symposium ASSET'99

[7] **I.F. Akyildiz, G. Morabito, S. Palazzo,** "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks.", IEEE/ACM Transactions On Networking, Volume 9, Number 3, Pages 307-321, 2001.

[8] **Hari Balakrishnan, Srinivasan Seshan, Elan Amir, Randy H. Katz,** "Improving TCP/IP Performance over Wireless Networks", In Proc. 1st ACM Int'l Conf. on Mobile Computing and Networking (Mobicom), November 95.

[9] **C. Casetti, M. Gerla, S. Mascolo, M.Y. Sansadidi, R. Wang,** "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks.", Wireless Networks Journal, Volume 8, Pages 467-479, 2002.

[10] **A. Bakre, B. R. Badrinath, "**I-TCP: indirect TCP for mobile hosts", Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)

[11] **RFC 768** - User Datagram Protocol

[12] **RFC 1889** - RTP: A Transport Protocol for Real-Time Applications

[13] **RFC 3448** - TCP Friendly Rate Control (TFRC)

[14] **Vijay Arya, Thierry Turletti, "**AED: An Accurate and Explicit Loss Differentiation Mechanism", 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)

[15] **NS, Network Simulator,** http://www.isi.edu/nsnam/ns/

[16] **TCL, Tool Command Language,** http://www.tcl.tk/

[17] **RFC 2582,** "The NewReno Modification to TCP's Fast Recovery Algorithm"

[18] **William Stallings,** "Wireless Communications and Networks"

[19] **RFC 2018,** "TCP Selective Acknowledgment Options"

EKLER

EK A. NS İçin Örnek TCL Betiği

Aşığada verilen örnek TCL betiğinde KA-TCP benzetimlerinin NS ile nasıl yapıldığı görülmektedir.

```
source "simulationUtil.tcl"

set simulationDuration [lindex $argv 0]
set lossyLinkErrorRate [lindex $argv 1]
set congestedLinkErrorRate [lindex $argv 2]
set historyWindowSize [lindex $argv 3]
set numberOfSpecialDupAcks [lindex $argv 4]
set endTime [expr ($simulationDuration+1)]

set ns [new Simulator]

set traceFile [open  ../results/aed_sourceSinkMobile.tr w]
set namTraceFile [open  ../results/aed_sourceSinkMobile.nam w]
set cwndFile [open ../results/aed_sourceSinkMobile.cwnd w]
set ssthreshFile [open ../results/aed_sourceSinkMobile.ssthresh w]

$ns namtrace-all $namTraceFile
$ns trace-all $traceFile

$ns color 1 Green
$ns color 2 Blue


proc createTopology { } {
      global    ns    ftpApplication    tcpAgent    lossyLinkErrorRate
congestedLinkErrorRate  numberOfSpecialDupAcks historyWindowSize

      set baseStationCM_1 [new AedConnectionManager]
      $baseStationCM_1 set history_window_size $historyWindowSize
      $baseStationCM_1 set received_packets_buffer_size 64
      $baseStationCM_1 set id 111

      set baseStationCM_2 [new AedConnectionManager]
      $baseStationCM_2 set history_window_size $historyWindowSize
      $baseStationCM_2 set received_packets_buffer_size 64
      $baseStationCM_2 set id 222

      set classifier_1 [new Classifier/Addr/Aed]
      $classifier_1 attach-connectionManager $baseStationCM_1

      set classifier_2 [new Classifier/Addr/Aed]
      $classifier_2 attach-connectionManager $baseStationCM_2
```

```
        set mobileHost_1 [$ns node]
        set mobileHost_2 [$ns node]

        set baseStation_1 [nodeWithClassifier $classifier_1]
        set baseStation_2 [nodeWithClassifier $classifier_2]
        $baseStation_1 shape "hexagon"
        $baseStation_2 shape "hexagon"

        $ns duplex-link $mobileHost_1 $baseStation_1 1Mb 10ms DropTail
        $ns duplex-link$ baseStation_1 $baseStation_2 1Mb 10ms DropTail
        $ns duplex-link $baseStation_2 $mobileHost_2 1Mb 10ms DropTail

        set lossyLink1 [$ns link $mobileHost_1 $baseStation_1]
        set lossyLink2 [$ns link $baseStation_1 $mobileHost_1]
        set lossyLink3 [$ns link $mobileHost_2 $baseStation_2]
        set lossyLink4 [$ns link $baseStation_2 $mobileHost_2]

        set congestedLink1 [$ns link $baseStation_1 $baseStation_2]
        set congestedLink2 [$ns link $baseStation_2 $baseStation_1]

        addError $lossyLink1 $lossyLinkErrorRate
        addError $lossyLink2 $lossyLinkErrorRate
        addError $lossyLink3 $lossyLinkErrorRate
        addError $lossyLink4 $lossyLinkErrorRate
        addError $congestedLink1 $congestedLinkErrorRate
        addError $congestedLink2 $congestedLinkErrorRate

        set tcpAgent [new Agent/TCP/Aed]
        $ns attach-agent $mobileHost_1 $tcpAgent

        $tcpAgent set class_ 1
        $tcpAgent set window_ 32

        set ftpApplication [new Application/FTP]
        $ftpApplication attach-agent $tcpAgent

        set sink [new Agent/TCPSink/Aed]
        $sink dup-special-ack-count $numberOfSpecialDupAcks
        $ns attach-agent $mobileHost_2 $sink
        set sinkCM [new AedConnectionManager]
        $sinkCM set history_window_size $historyWindowSize
        $sinkCM set received_packets_buffer_size 64
        $sinkCM set id 999
        $sink attach-connectionManager $sinkCM

        $ns connect $tcpAgent $sink
}

createTopology

$ns at 0.1 "$ftpApplication start"
$ns at 0.1 "plot $tcpAgent"

$ns at $simulationDuration "$ftpApplication stop"
$ns at $endTime "finish"

$ns run
```

EK B. Benzetim Sonuçları

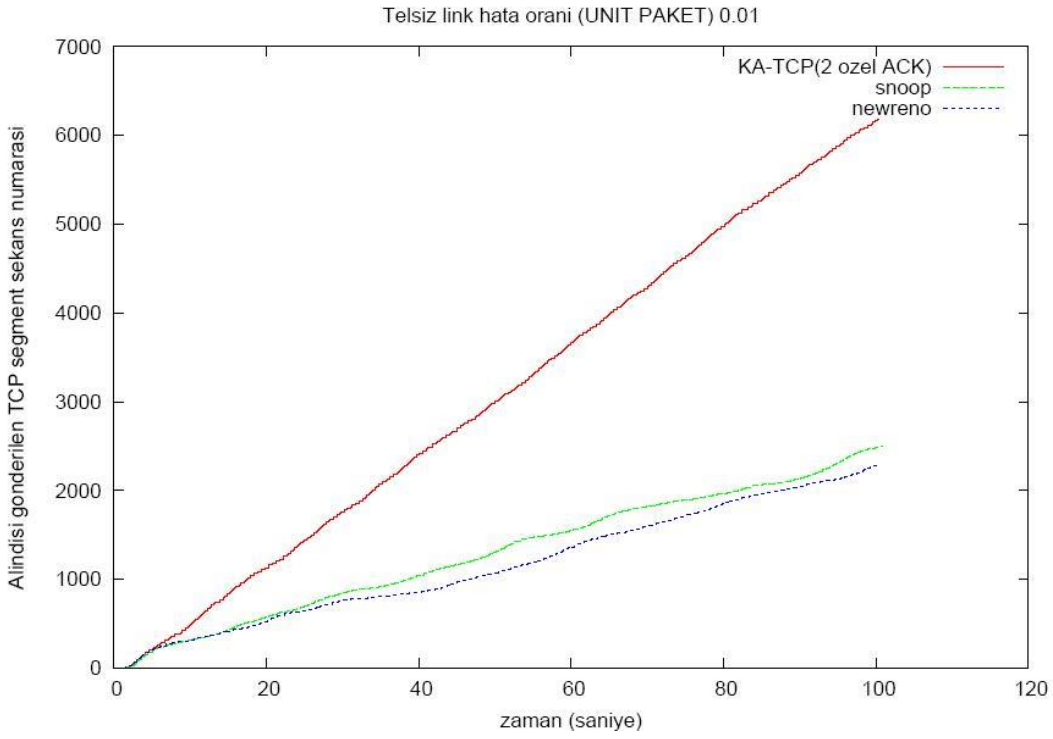
Benzetimlerde 2 özel alındı gönderen KA-TCP, Snoop ile beraber çalışan SACK TCP ve TCP NewReno performansları karşılaştırıldı.

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telli ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı 0.01, telli ve telsiz link kapasitesi 10Mb, her link üzerinde gecikme 50ms

Sonuç

Şekil B.1'de benzetim sonucunun grafiği var. KA-TCP, SNOOP ve TCP NewReno'dan daha iyi sonuç vermektedir. Kaynak telsiz ortamda olduğu zaman Snoop'un neden başarım kaybı yaşadığı bölüm 2.2.6 Snoop Protokolü'nde anlatılıyor.



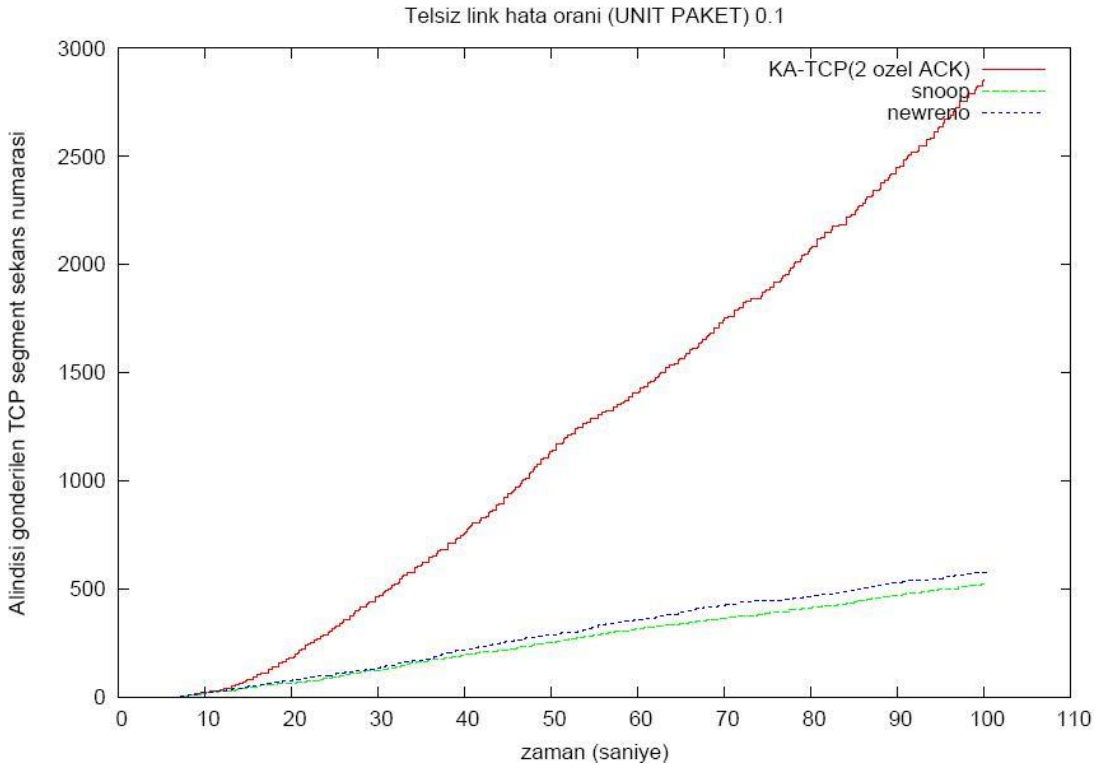
Şekil B.1: Kaynak telsiz ortamda, hedef telli ortamda, telsiz link hata oranı 0.01

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telli ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde on, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği bütün linklerde gecikme 50ms

Sonuç

Şekil B.2'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. TCP NewReno, 3. Snoop. Telsiz link hata oranı artıkça KA-TCP ile elde edilen veri iletim başarımı daha belirgin görülmektedir.



Şekil B.2: Kaynak telsiz ortamda, hedef telli ortamda, telsiz link hata oranı 0.1

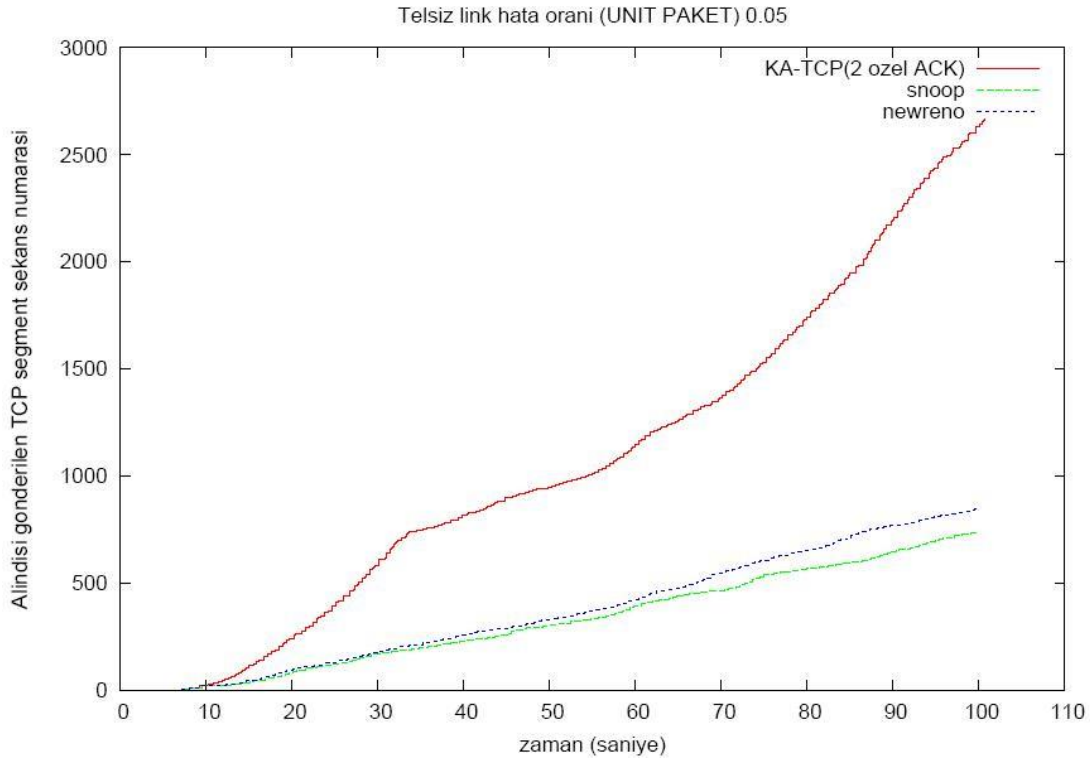
Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telli ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde beş, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği bütün linklerde gecikme10ms.

Sonuç

Şekil B.3'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. TCP NewReno, 3. Snoop.

Telsiz link hata oranı artıkça doğal olarak TCP veri iletim başarımı azalmaktadır. Bu bütün tıkanıklık denetimi algoritmaları için aynıdır. Fakat KA-TCP ile SNOOP ve TCP NewReno'ya göre daha fazla veri iletimi başarılmıştır.



Şekil B.3: Kaynak telsiz ortamda, hedef telli ortamda, telsiz link hata oranı 0.05

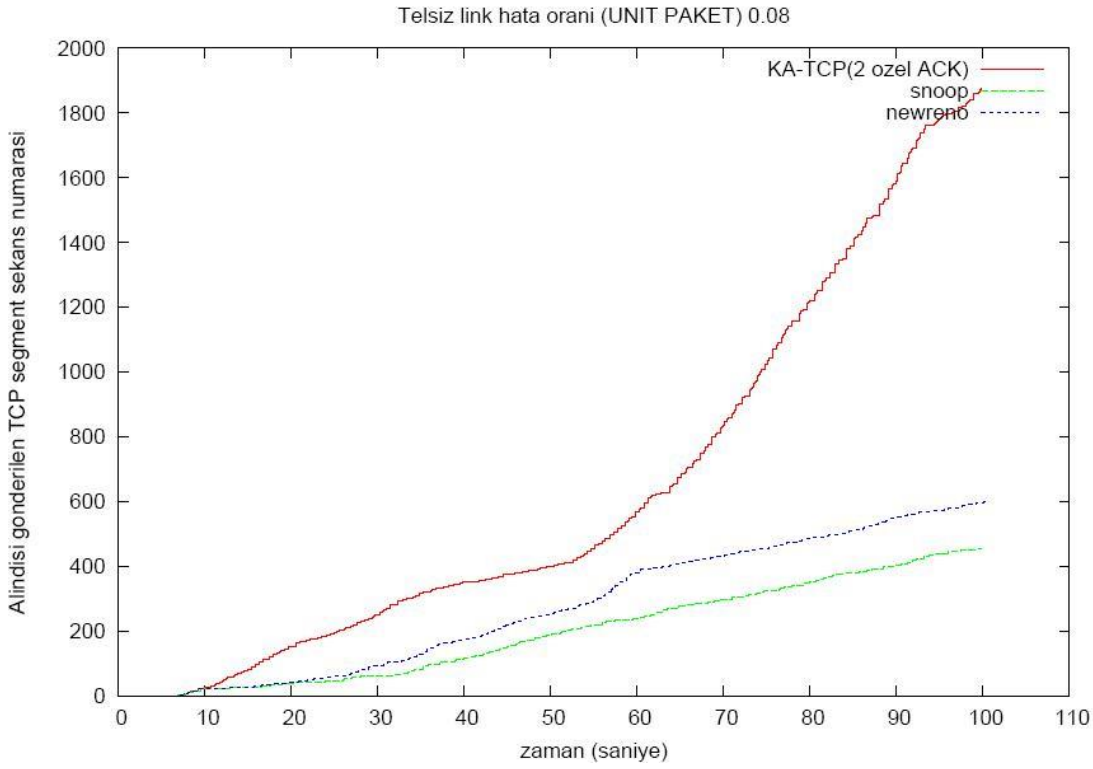
Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telli ortamda. (Şekil 4.2'deki topoloji)
- Telsiz link hata oranı yüzde sekiz, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme 50ms.

Sonuç

Şekil B.4'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. 2. TCP NewReno, 3. Snoop.

KA-TCP ile SNOOP ve TCP NewReno'ya göre daha fazla veri iletimi başarılmıştır.



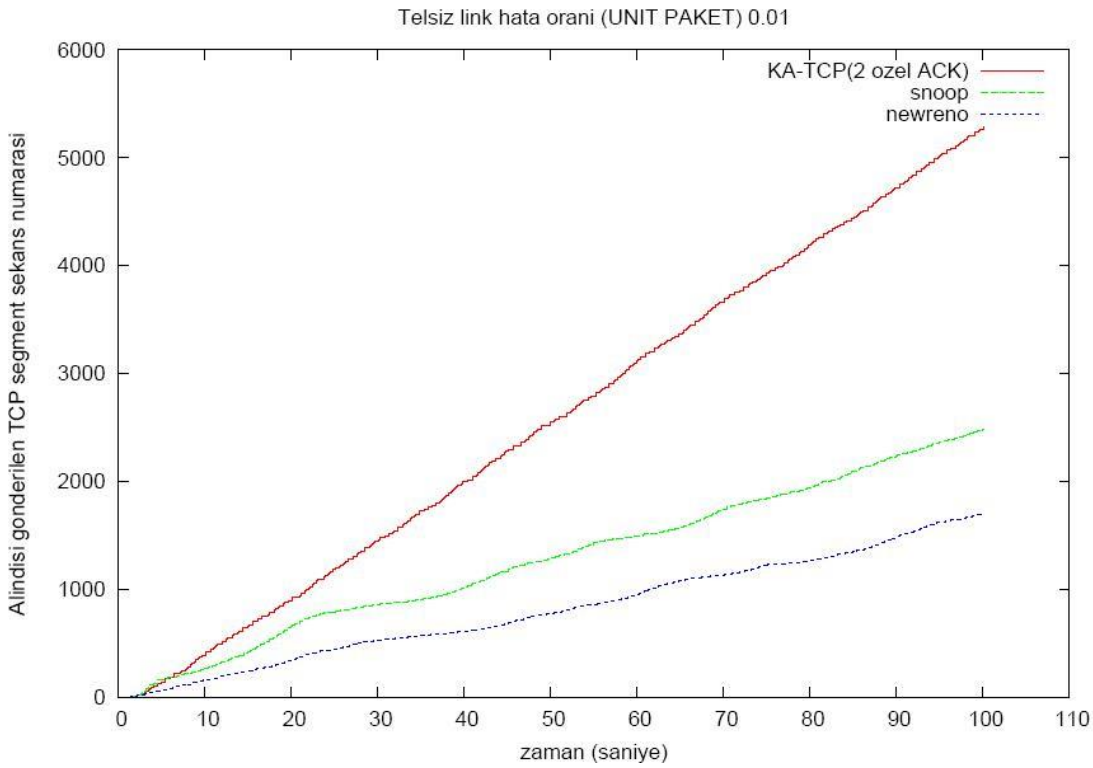
Şekil B.4: Kaynak telsiz ortamda, hedef telli ortamda, telsiz link hata oranı 0.08

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telsiz ortamda. (Şekil 4.2'deki topoloji)
- Telsiz link hata oranı yüzde bir, telli ve telsiz link kapasitesi 10Mb, TCP bağlantılarının geçtiği linklerde gecikme 50ms.

Sonuç

Şekil B.5'de benzetim sonucunun grafiği var. KA-TCP yöntemi, Snoop ve TCP NewReno'dan daha fazla segman taşımış. En kötü sonucu TCP NewReno'nun verdiği görülüyor.



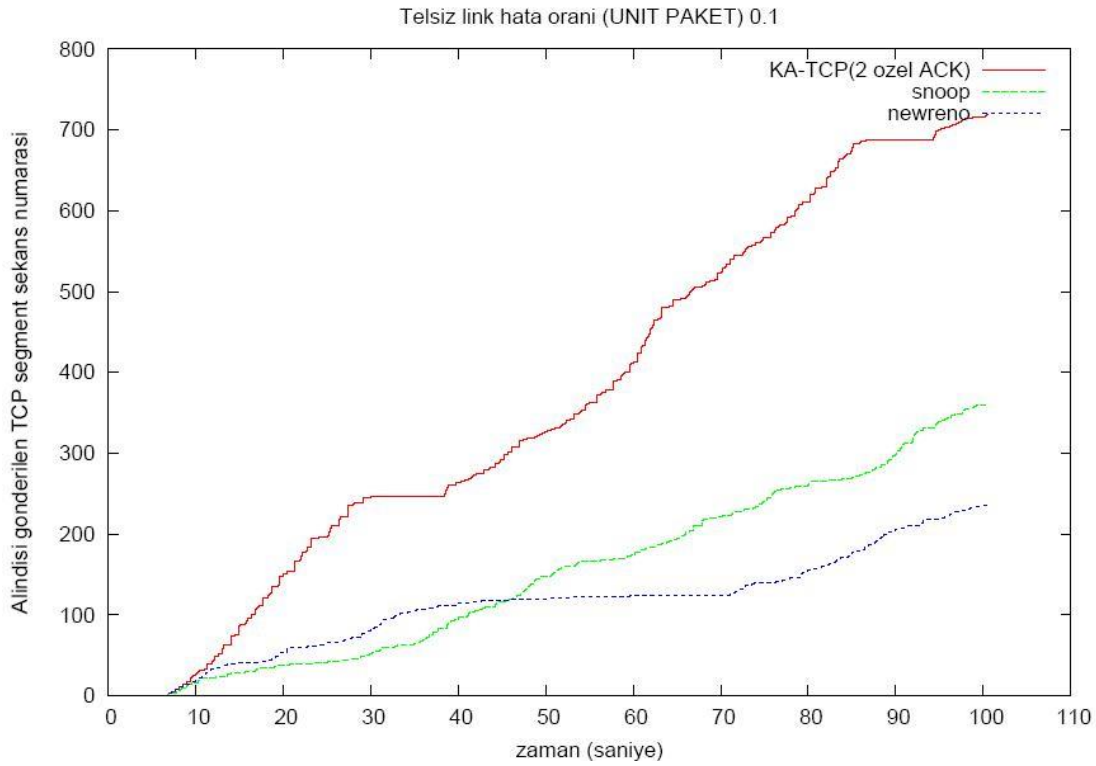
Şekil B.5: Kaynak telsiz ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.01

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telsiz ortamda. (Şekil 4.2'deki topoloji)
- Telsiz link hata oranı yüzde on, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme 50ms.

Sonuç

Şekil B.6'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno



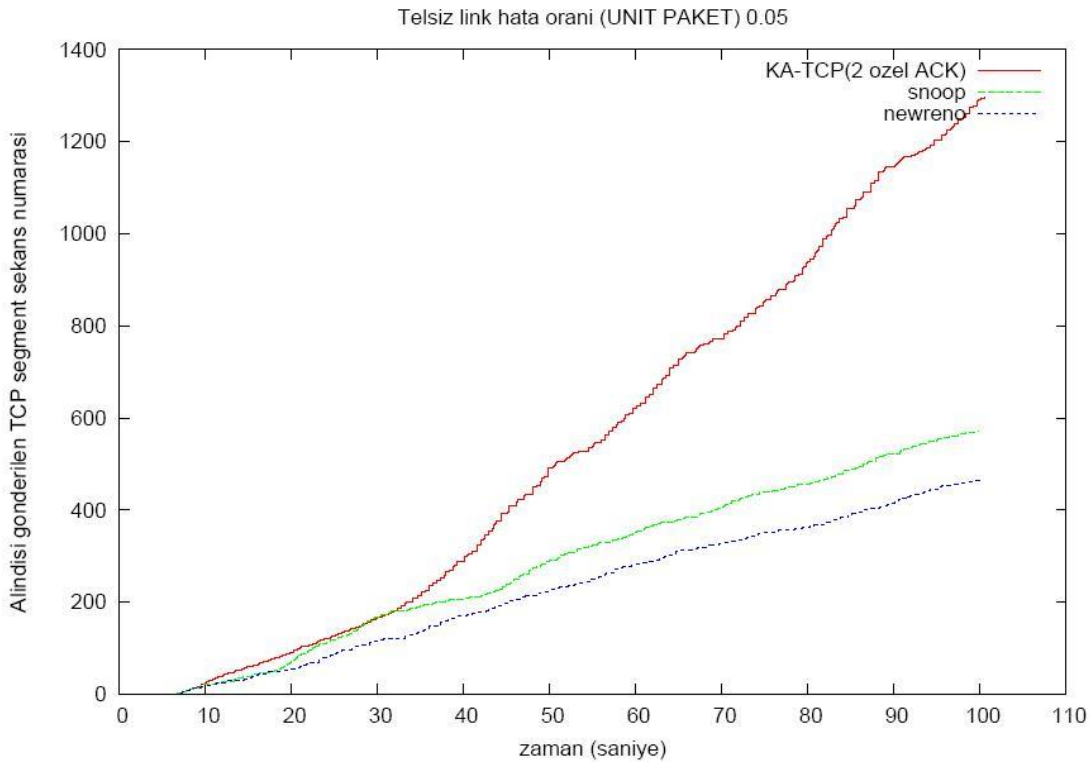
Şekil B.6: Kaynak telsiz ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.1

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telsiz ortamda. (Şekil 4.2'deki topoloji)
- Telsiz link hata oranı yüzde beş, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme10ms

Sonuç

Şekil B.7'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno



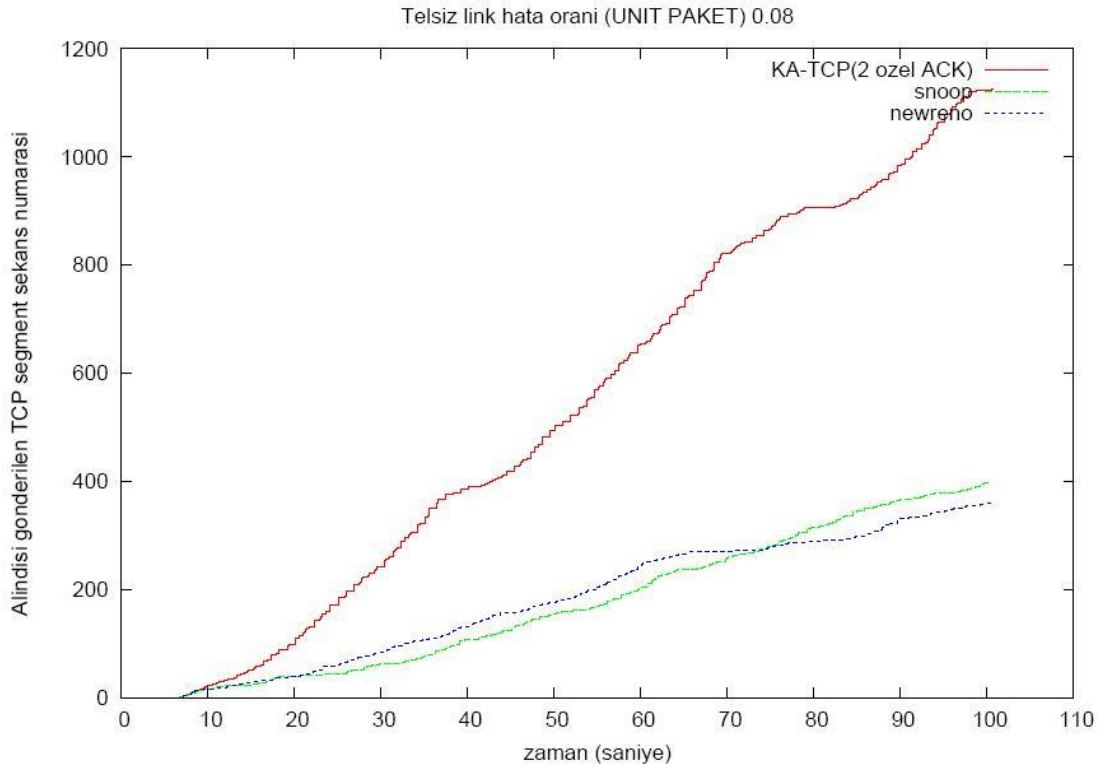
Şekil B.7: Kaynak telsiz ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.05

Benzetim Ortamı

- Kaynak telsiz ortamda, hedef telsiz ortamda. (Şekil 4.2'deki topoloji)
- Telsiz link hata oranı yüzde sekiz, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme10ms.

Sonuç

Şekil B.8'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno



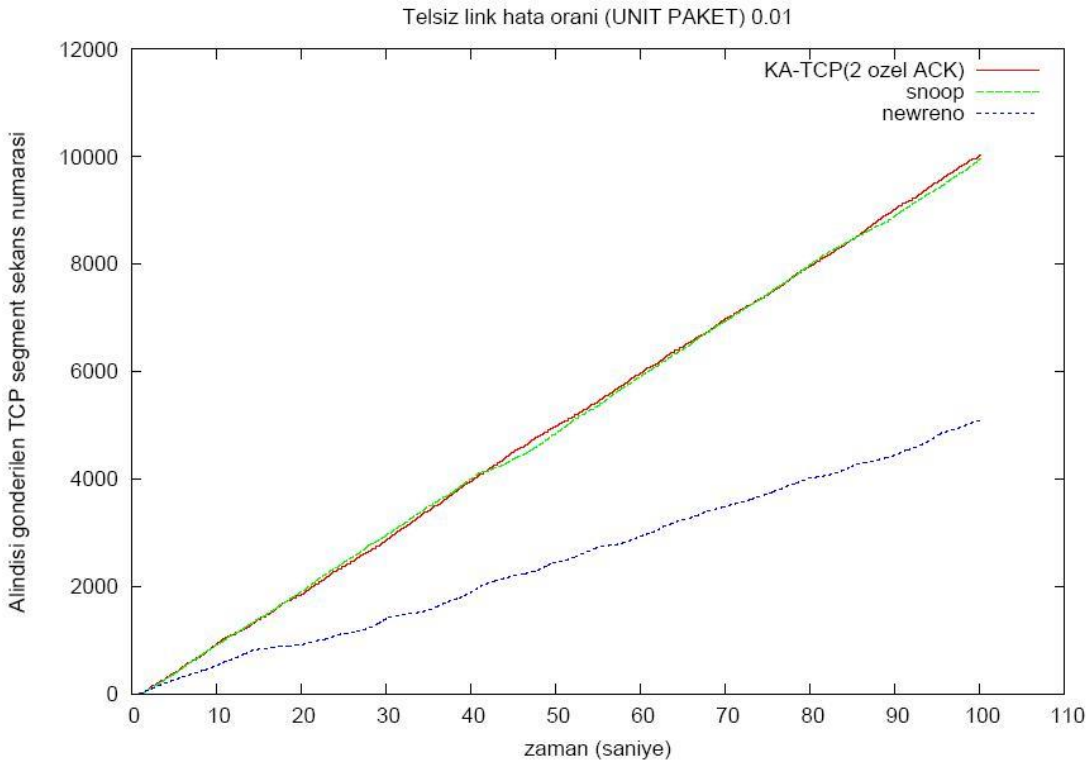
Şekil B.8: Kaynak telsiz ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.08

Benzetim Ortamı

- Kaynak telli ortamda, hedef telsiz ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde bir, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme10ms.

Sonuç

Şekil B.9'de benzetim sonucunun grafiği var. KA-TCP ve SNOOP benzer sonuçları vermiştir. Telli linkler için tasarlanmış olan TCP NewReno'nun elde ettiği veri iletim başarımı KA-TCP ve SNOOP'a göre daha azdır.



Şekil B.9: Kaynak telli ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.01

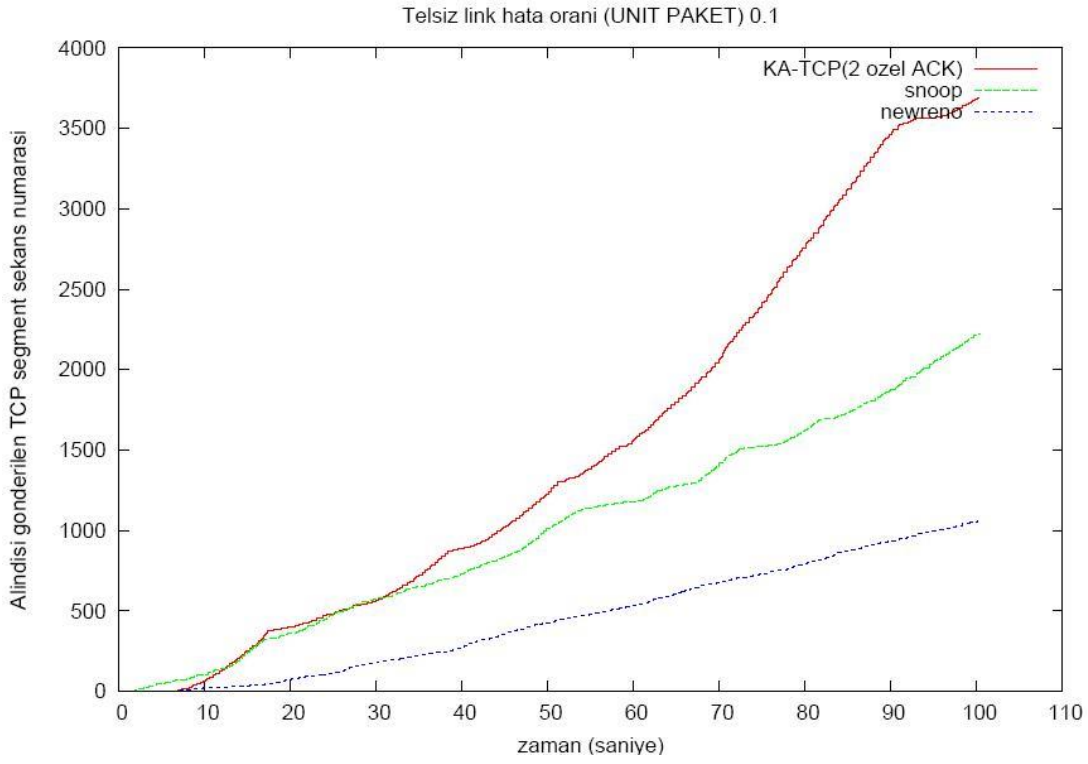
Benzetim Ortamı

- Kaynak telli ortamda, hedef telsiz ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde on, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme10ms.

Sonuç

Şekil B.10'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno

Grafikten de anlaşılacağı gibi, telsiz link hata oranı arttıkça KA-TCP ile elde edilen veri iletim başarımı diğer yöntemlere göre daha fazla olmaktadır.



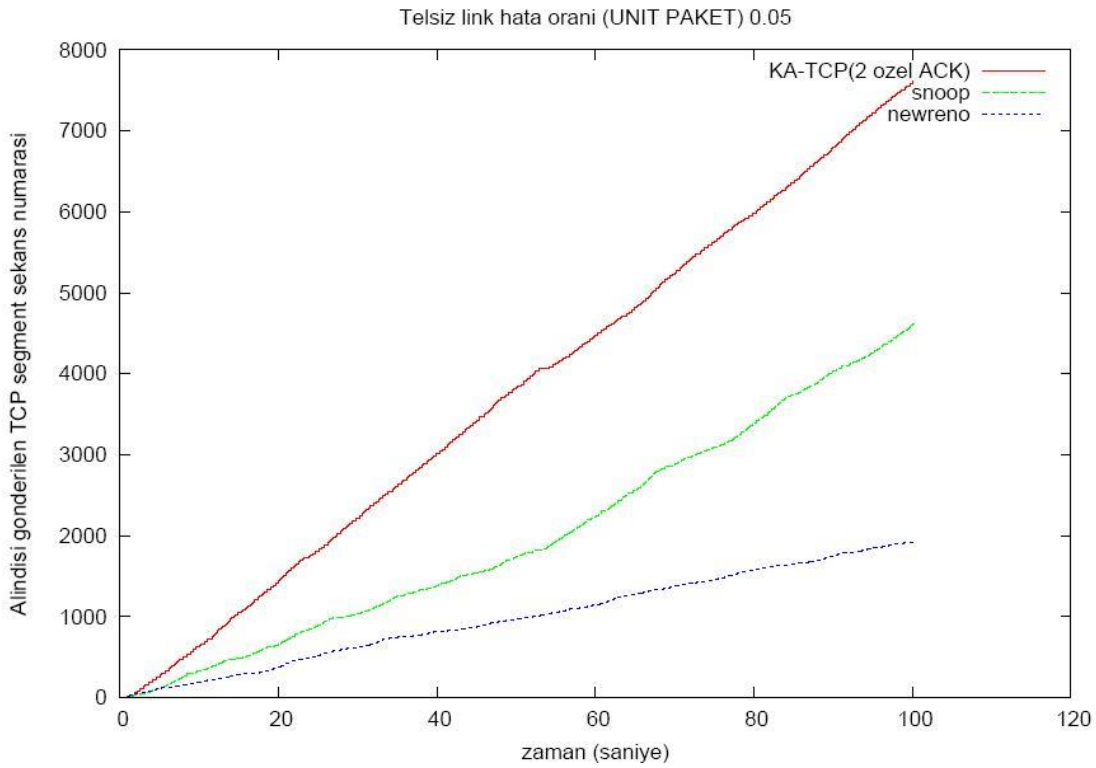
Şekil B.10: Kaynak telli ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.1

Benzetim Ortamı

- Kaynak telli ortamda, hedef telsiz ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde beş, telli ve telsiz link kapasitesi 10Mb, TCP bağlantısının geçtiği linklerde gecikme10ms.

Sonuç

Şekil B.11'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno



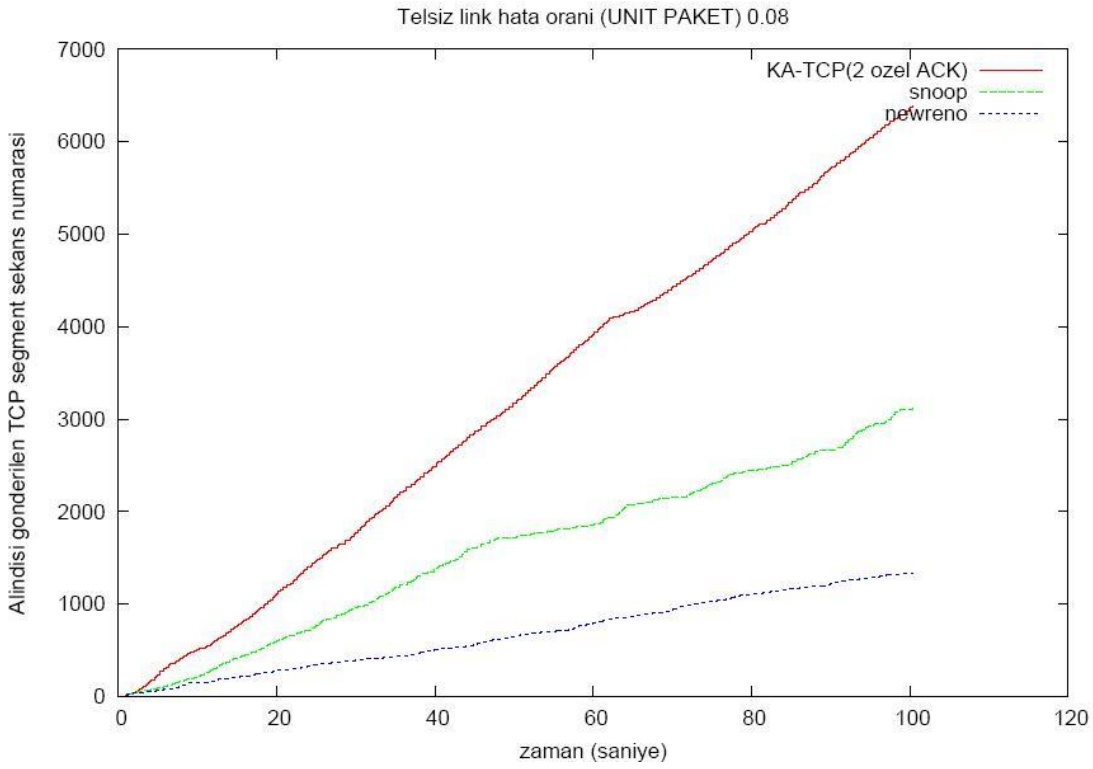
Şekil B.11: Kaynak telli ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.05

Benzetim Ortamı

- Kaynak telli ortamda, hedef telsiz ortamda. (Şekil 4.1'deki topoloji)
- Telsiz link hata oranı yüzde sekiz, telli ve telsiz link kapasitesi 10Mb, TCP bağlamtısının geçtiği linklerde gecikme10ms.

Sonuç

Şekil B.12'de benzetim sonucunun grafiği var. Yöntemlerin başarımları en iyiden en kötüye doğru : 1. İki özel alındı gönderen KA-TCP, 2. Snoop, 3. TCP NewReno



Şekil B.12: Kaynak telli ortamda, hedef telsiz ortamda, telsiz link hata oranı 0.08

ÖZGEÇMİŞ

İlkin Ulaş Balkanay, 4 Şubat 1979'da Ankara'da doğdu. Ortaokulu ve liseyi Denizli Anadolu Lisesi'de okuduktan sonra 1997 yılında Galatasaray Üniversitesi Bilgisayar Mühendisliği Bölümünü kazandı. 2002 yılında İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim dalında yüksek lisans eğitimine başladı.

2002-2003 yılları arasında Galatasaray Üniversitesi Bilgisayar Mühendisliği Bölümü'nde araştırma görevlisi olarak çalıştı. Şu anda, Haziran 2003 ayında girdiği Oksijen (www.oksijen.com) araştırma-geliştirme firmasında yazılım geliştirme mühendisi olarak çalışmaktadır.