TCP'deki GÜVENILIR VERI ILETIM PROTOKOLÜNÜN DOSYA TRANSFERI IÇIN HIZLANDIRILMASI

Ihsan GÜNES

Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü

Bilgisayar Mühendisligi Anabilim Dali

Subat – 2005

JÜRI VE ENSTITÜ ONAYI

Ihsan GÜNES'in **"TCP'deki Güvenilir Veri Iletim Protokolünün Dosya Transferi için Hizlandirilmasi"** baslikli **Bilgisayar Mühendisligi Anabilim** Dalindaki, Yüksek Lisans tezi.....tarihinde, asagidaki jüri tarafindan Anadolu Üniversitesi Lisansüstü Egitim-Ögretim ve Sinav Yönetmeliginin ilgili maddeleri uyarinca degerlendirilerek kabul edilmistir.

	Adi-Soyadi	**Imza**
Üye (Tez Danismani)	**: Yard. Doç. Dr. Cüneyt AKINLAR**	**……**
Üye	**: Prof. Dr. Ali GÜNES**	**……**
Üye	**: Yard Doç. Dr. Aydin AYBAR**	**……**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …… tarih ve …… sayili karariyla onaylanmistir.

Enstitü Müdürü

ÖZET
Yüksek Lisans Tezi

TCP'deki GÜVENILIR VERI ILETIM PROTOKOLÜNÜN DOSYA TRANSFERI IÇIN HIZLANDIRILMASI

IHSAN GÜNES

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisligi Anabilim Dali

Danisman: Yard. Doç. Dr. Cüneyt AKINLAR
2005, 61 sayfa

Güvenilir, sirali paket iletim hizmeti sunan Iletim Denetimi Protokolü (Transmission Control Protocol – TCP), yaygin olarak kullanilan FTP, HTTP gibi birçok protokolü de içeren Internet protokollerinin temelini olusturur. TCP'nin sirali iletim yapan dogasinin "head-of-line blocking" olarak bilinen soruna neden oldugu bilinen bir olgudur. Bu, TCP gönderici penceresinin içerisindeki ilk paket kayboldugunda, pencerenin, bu ilk paket basariyla gönderilinceye ve alici tarafta basariyla alinincaya kadar ileriye dogru kaymamasi durumudur. Bu tezde, bahsedilen soruna çözüm olarak TCP'ye , sirali olmayan paket iletimini destekleyen bir arayüz eklenmesi önerilmektedir. Bu sirali olmayan paket iletimini kullanan uygulamalarin paket siralamasini uygulama katmaninda yapabilme yetenegine sahip olmasi gerekmektedir. Bu arayüzün, varolan TCP uygulamalarina küçük degisikliklerle nasil uygulanabilecegi tanimlanmakta, popüler "ns" simülatorü kullanilarak bu uygulamanin diger TCP uygulamalariyla karsilastirmasi yapilmakta ve Reno TCP ile karsilastirildiginda %450'ye, Newreno TCP ile karsilastirildiginda ise %25'e varan performans artislari sagladigi gösterilmektedir.

Anahtar Kelimeler: TCP, Newreno, Reno, Akis Kontrolü, Tikaniklik Kontrolü

ABSTRACT

Master of Science Thesis

IMPROVING TCP's RELIABLE DATA TRANSFER PROTOCOL FOR FILE TRANSFERS

IHSAN GÜNES

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Assist. Prof. Dr. Cüneyt AKINLAR
2005, 61 pages

Transmission Control Protocol (TCP), which offers a reliable, in-order packet delivery service, is at the heart of many Internet protocols including FTP, HTTP. It is well known that in-order nature of TCP causes what is known as a "head-of-line blocking" problem. That is, when the first packet within a TCP sender window is lost, the window will not move forward until the packet is retransmitted and successfully recovered at the receiver. In this thesis we propose adding an unordered packet delivery service interface to TCP, that would remedy this problem. Applications using this unordered paket delivery interface would be required to perform packet reordering at the application layer. We describe how this interface can be implemented with simple changes to an existing TCP implementation, compare the resulting protocol to various TCP implementations using the popular "ns" simulator and show that the performance of bulk data transfer applications improve up to 450% compared to Reno TCP and up to 25% compared to NewReno TCP.

Keywords: TCP, Reno, Newreno, Flow Control, Congestion Control

IÇINDEKILER

SEKILLER DIZINI

SIMGELER ve KISALTMALAR DIZINI

ACK	Acknowledge
API	Application Protocol Interface
CBR	Constant Bit Rate
EACK	Extended ACK
FTP	File Transfer Protokol
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
NAM	Network Animator
NETBLT	Network Block Transfer
NS	Network Simulator
MSS	Maxmimum Segment Size
RDP	Reliable Data Protocol
RTT	Round Trip Time
SACK	Selective Acknowledge
SCTP	Stream Control Transmission Protokol
SMTP	Simple Mail Transport Protokol
SSN	Stream Sequence Numbers
UTCP	Unordered Transmission Control Protokol
UDP	User Data Protocol
TCP	Transmission Control Protokol
TSN	Transmission Sequence Number

1. GIRIS VE AMAÇ

Iletim kontrol protokolü (Transmission Control Protoc ol, TCP), Internet üzerinde bilgisayarlar arasinda veri transferi için kullanilan en yaygin protokoldür. Birçok uygulama katmani protokolü (örnegin; HTTP [1], FTP [2], SMTP [3]) haberlesmek için güvenilir ve baglanti-temelli olan TCP protokolünü kullanir. TCP nin cok yaygin olarak kullaniliyor olmasi, performansinin iyi olmasi gerekliligini de beraberinde getirir. Bu tezde, TCP'nin veri iletim mekanizmasi degistirilerek performansinin veri transferi uygulamalari için gelistirilmesi üzerine çalisilmistir.

1.1. Problemin Tanimi

TCP, IP aglari üzerinde güvenilir veri transferini saglar [4]. Internet teknolojilerinin kullaniminin son yillarda hiç kimsenin beklemedigi kadar yayginlasmasi, veri transferi için en yaygin olarak kullanilan TCP protokolünün performansinin gelistirilmesine yönelik çalismalara büyük bir ivme kazandirmistir. Son zamanlarda, TCP protokolünü çok kisitlayici bulan ve kendi güvenilir veri transfer protokollerini UDP [5] üzerinde birlestiren uygulamalarin sayisi giderek artmaktadir. TCP, güvenilir veri transferi ve kati bir sekilde sirali iletim yapmaktadir. TCP'nin sirali iletim yapmasindan kaynaklanan "head-of–line blocking" problemi, veri iletimini yavaslatmaktadir. Bu problem, göndericinin veri penceresi içerisindeki ilk paketinin kaybolmasi durumunda, pencerenin, bu paketin basarili bir sekilde iletilmesine kadar ileriye dogru kaymamasidir. Bu bekleme süresinde yeni bir paket gönderilmeyecegi için baglantinin veri bant genisligi verimli kullanilamamaktadir. Biz bu tezde, TCP protokolünü degistirerek güvenilir ve sirali olmayan veri transferi yapan UTCP (Unordered TCP) protokolünü gelistirdik. Bunun sonucunda, paketleri siralama isleminin TCP üzerinde getirmis oldugu yük ortadan kaldirilarak güvenilir veri iletiminin hizlandirilmasi saglanmistir. Paketleri siralama islemi, tamamen uygulama katmanina birakilmistir.

TCP'nin yukarida bahsedilen kisitlamalarindan dolayi meydana gelen gecikmeleri azaltmak için FlashGet [6] gibi bazi veri transferi uygulamalari, bir dosyayi indirmek için iki uç arasinda birden fazla TCP baglantisi kurarlar ve

dosyanin her bir parçasini farkli bir baglanti üzerinden indirirler. Dosya indirme islemi bittiginde parçalar birlestirilerek tek bir dosya elde edilir. Bu yaklasim verimliligi artirmaktadir. Çünkü her bir baglanti birbirinden bagimsizdir ve bir baglantida kaybolan bir paket diger baglantidan iletilen paketleri etkilemeyecektir. Fakat bu yaklasimin bazi dezavantajlari vardir. Iki uç arasinda bir dosyayi transfer etmek için birden fazla TCP baglantisi kurulmasi, hem istemci hem de sunucu isletim sistemleri üzerinde asiri yük olusmasina sebep olmaktadir. Bu durum da isletim sisteminin performansini olumsuz etkilemektedir. Bir diger dezavantaj ise her bir baglanti üzerinde "head-of-line blocking" probleminin devam etmesinden dolayi verimin düsmesidir.

Veri iletiminin hizlandirilmasi için SCTP [7], RDP [8], NETBLT [9] gibi bir çok protokol gelistirilmistir. SCTP, iki uç arasinda birden fazla mantiksal veri akisi kullanir. Bu mantiksal veri akislari üzerinde bir paketin kaybolmasi diger akislardaki paketleri etkilemeyecektir. Bütün veri akislari birbirinden bagimsizdir. Ancak her veri akisi yine kendi içinde sirali iletim yapmaktadir. RDP protokolünde güvenilir ve sirali olmayan veri iletimi saglanmistir. Ancak TCP'nin çok karmasik olan tikaniklik kontrolü (Congestion Control) algoritmasindan yoksundur ve çok kullanilmamaktadir. NETBLT protokolü büyük hacimli veri transferleri için gelistirilmistir. Ancak ara ag geçitlerindeki kisitlamalardan dolayi Internet üzerinde kullanilamamaktadir. Sonuç olarak bu protokollerin hiç biri TCP gibi yayginlasmamis, genellikle sadece arastirma için kullanilmislardir.

Bu çalismada, TCP'nin veri akis mekanizmasini degistirerek güvenilir ve sirali olmayan veri iletimi yapan ve TCP'nin akis kontrol (Flow Control) ve tikaniklik kontrol algoritmalarini da içeren UTCP protokolü gelistirdik.

1.2. Katkilar

Bu tezde bizim amacimiz TCP'nin büyük hacimli veri transferlerindeki performansini gelistirmektir. Bunun için TCP'nin veri iletiminin kontrolünü saglayan kayan pencereler algoritmasini degistirdik. Çalisma ortami olarak ag simülasyonlari için en yaygin olarak kullanilan NS (Network Simulator) [10] programi seçilmistir. NS içerisinde bulunun Newreno TCP kodunu degistirerek ve

eklentiler yaparak UTCP isimli yeni bir protokol gelistirdik. NS üzerinde simülasyonlarla UTCP'nin performansini diger TCP versiyonlari ile karsilastirdik.

1.3. Tez Organizasyonu

Bölüm 2'de ilk kisimda TCP hakkinda detayli bilgiler verilmistir. Bu bölümde ayrica TCP'nin performansini artirmak için yapilan çalismalar ve veri transferini hizlandirmak için gelistirilen diger protokollerden bahsedilmistir. Bölüm 3'te NS hakkinda bazi bilgiler verilmis, gelistirmis oldugumuz UTCP protokolünün çalisma prensiplerinin detaylari açiklanmis, ve UTCP ile farkli TCP versiyonlari simülasyonlar ile karsilastirilmistir. Bölüm 4'le elde edilen sonuçlar degerlendirilmis ve gelecekte bu konuda hangi çalismalarin yapilabileceginden bahsedilmistir.

2. ALTYAPI VE ILGILI ÇALISMALAR

2.1. Iletim Kontrol Protokolü (Transmission Control Protocol , TCP)

Bilgisayar aglarinin sürekli olarak gelismesi ve yayginlasmasi sonucunda farkli donanim özelliklerine sahip ag yapilari arasindaki iletisimin saglanmasi amaciyla standart bir iletim protokolünün gelistirilmesi ihtiyaci kaçinilmaz olmustur. Küresel bir iletisim agi olusturabilmek amaciyla TCP protokolü gelistirilmistir.

TCP baglanti temelli (connection-oriented) güvenilir bayt akisi hizmeti sunan karmasik bir iletim protokolüdür. Ayni zamanda en temel teknolojilerin TCP/IP trafigini kullanabilecegi sekilde esnektir. Hata, akis, ve tikanik kontrolü islevlerini sunan bir uçtan uça (end-to-end) protokoldür [4].

TCP protokolü, uygulama katmanindan aldigi verileri uygun parçalara bölüp her bir parçaya TCP basligi ekler. Bu parçalar TCP segmenti adi verilmektedir. TCP protokolü olusturmus oldugu bu segmentleri alt bir katman olan IP protokolüne verir.

Segment içerisine koyulabilecek maksimum data miktari MSS (maximum segment size) degiskeni ile sinirlandirilmistir. MSS, TCP uygulamasina baglidir (isletim sistemi tarafinda belirlenir) ve çogunlukla ayarlanabilinir. MSS'in aldigi genel degerler; 1,500 bayt, 536 bayt ve 512 bayt seklindedir [11].

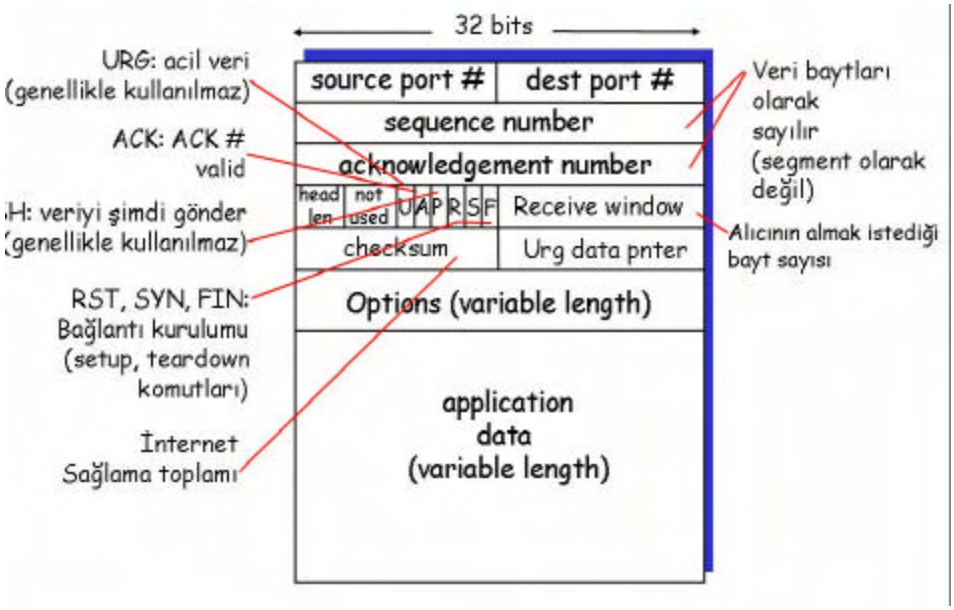
2.1.1. TCP Segment Yapisi

TCP segment baslik (header) ve veri (data) alanlarini içermektedir. Veri alani uygulama data parçalarini içermektedir. Yukarida da bahsettigimiz gibi MSS bu veri alaninin maksimum boyutunu sinirlamaktadir.

Sekil 2.1 TCP segmentinin yapisini göstermektedir.

- Kaynak (source) ve hedef (destination) port alanlari, verinin hangi uygulamadan gelip hangi uygulamaya gidecegini belirler.
- 32 bit sira numarasi (sequence number) ve 32 bit alindi bilgisi (acknowledgment) alanlari, TCP göndericisi ve alicisi için güvenilir ve sirali veri transferinde kullanilir.

- 16 bit alici penceresi (receive window) alani, akis kontrolü için kullanilir. Kisaca anlatirsak alicinin kabul edebilecegi bayt sayisini belirtir.
- Baslik uzunlugu (Header Length) alani, TCP basliginin uzunlugunu göstermektedir. TCP basligi, TCP, seçenekler (options) alanina bagli olarak degisken olabilir. (Genellikle seçenekler alani bos oldugu için TCP basligi 20 bayt'tir.)
- Seçmeli ve degisken uzunlugu olan seçenekler alani, gönderici ve alici maksimum segment boyutu (MSS) konusunda haberlesmek istedikleri zaman kullanilir.
- Bayrak (Flag) alani 6 bit içerir. ACK biti alindi bilgisi alaninda tasinan degerin geçerli oldugunu belirtir. RST, SYN ve FIN baglantinin kurulmasi ve koparilmasi için kullanilir. PSH biti ayarlandiginda, alicinin üst katmanlara acilen veri geçirmesi gerektigini gösterir. URG biti segment içerisinde acil olarak üst katmanlara iletilmesi gereken veri oldugunu belirtir.



Sekil 2.1 TCP Segmentinin Yapisi[11]

.

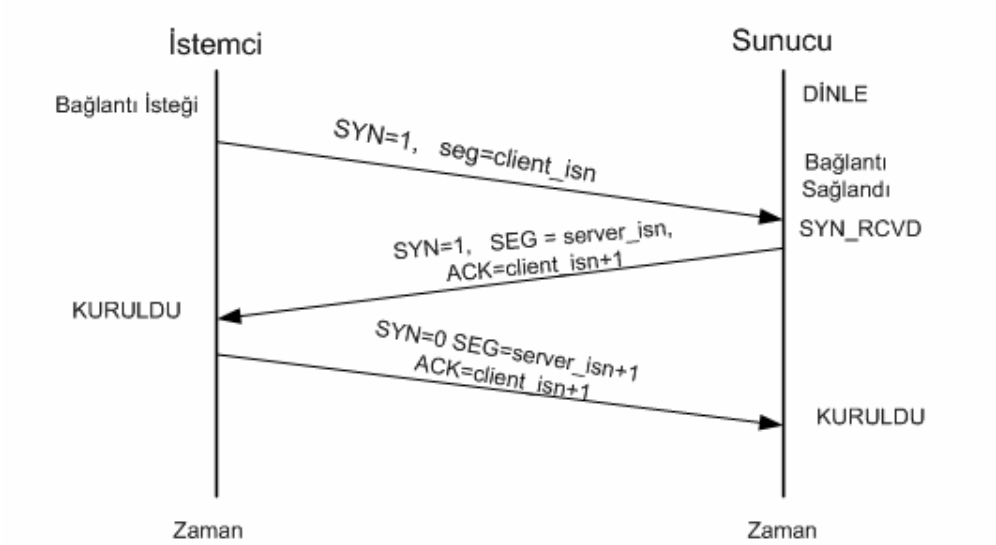
2.1.2. TCP Baglantisinin Kurulmasi ve Sonlandirilmasi

TCP, baglanti temellidir (connection-oriented), çünkü bir uygulamanin diger bir uygulamaya veri göndermesine baslamadan önce ilk olarak iki uç birbiriyle anlasmalidir. Iki uygulama da baglantiyi kurmak için birbirlerine öncü segmentler göndermek zorundadirlar. Bu birbirlerine gönderilen segmentler ile baglanti kurulma islemine "Three Way Handshake (Üç Yollu Antlasma)" denir [12]. Bir istemcideki bir uç'un sunucudaki bir uç ile bir baglanti baslatmak istedigini varsayarak baglanti kurulmasi için gerekli adimlari asagidaki sekilde ifade edebiliriz.

- Ilk olarak istemci taraftaki TCP, sunucu taraftaki TCP'ye özel bir segment gönderir. Bu özel segment uygulama katmani verisi içermez. Sadece segmentin basliginda bir bayrak biti içerir. Bu bayrak SYN olup degeri 1 olarak ayarlanmistir. Bu yüzden bu segmente SYN segmenti denilmektedir. Ayrica istemci TCP bir baslangiç sira numarasi (client_isn) seçer ve bu numarayi baslangiç SYN segmenti içerisindeki sira numarasi alanina koyar. Bu segment IP katmanina geçirilerek sunucuya iletilir.
- TCP SYN segmenti sunucuya geldiginde sunucu, baglanti için ara bellek ve degiskenler ayrilir ve baglantiyi kabul ettigine dair bir segment gönderir. Bu segment de uygulama katmani verisi içermez, fakat segment basliginda 3 tane önemli bilgi içerir. Bunlar 1 olarak ayarlanmis SYN biti, client_isn + 1 olarak ayarlanmis segment basligindaki ACK bilgisi ve sunucunun seçip segment basligindaki sira numarasi alanina koymus oldugu baslangiç sira numarasi (server_isn) bilgisidir. Sunucunun göndermis oldugu bu segment, "Baglantiyi senin baslangiç sira numaranla baslatmak için SYN paketini aldim. Bu baglantiyi onayliyorum ve baslangiç sira numaram server_isn bilgisini gönderiyorum" anlamini ifade etmektedir. Sunucunun göndermis oldugu bu segment SYNACK segmenti olarak da isimlendirilir.
- Istemci SYNACK segmentini aldiginda, baglanti için gerekli ara bellek ve degiskenleri ayirir. Istemci sunucunun göndermis oldugu

segmenti aldigini göstermek için son bir segment gönderir. Bu segmentin ACK bilgisi alanina sunucudan gelen segmentin alindigini belirtmek için server_isn + 1 degeri koyulur ve baglantinin kuruldugunu belirtmek içinde SYN biti 0 olarak ayarlanir.

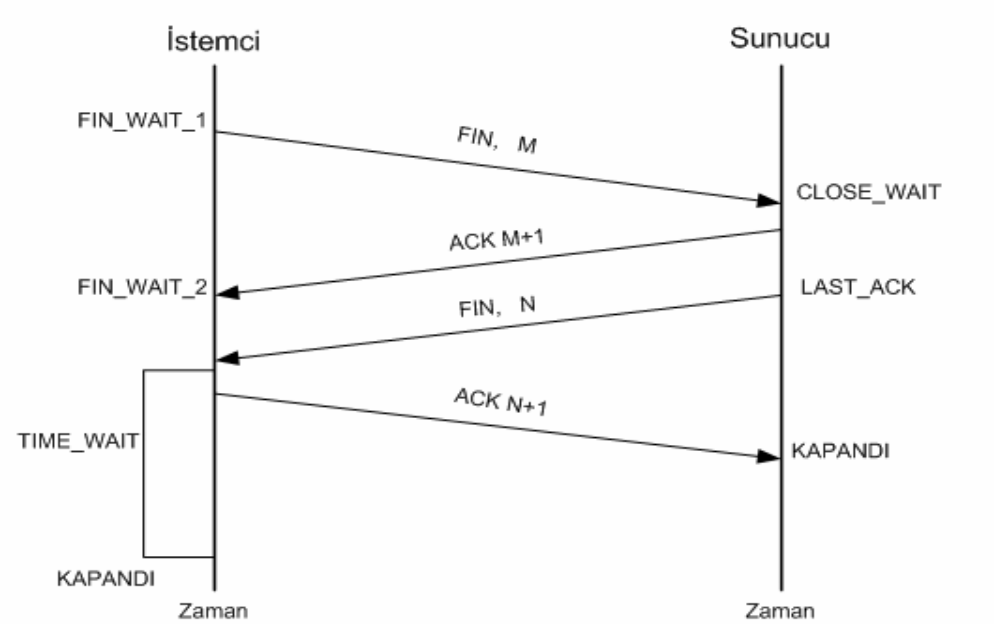
Yukaridaki üç adim tamamlandiktan sonra istemci ile sunucu birbirlerine veri içeren segment gönderebilirler. Bu segmentlerin hepsinde SYN biti 0 olacaktir. Bu üç paketin gönderilerek baglantinin kurulma islemi Sekil 2.2'de gösterilmistir. Bu baglanti kurulma islemine genellikle **three-way-handshake** olarak isimlendirilmektedir.



Sekil 2.2 Iki Uç arasindaki Baglantinin Kurulmasi

Iki uç arasinda veri iletimi bittikten sonra baglanti sonlandirilir. Baglanti kurulmasinda oldugu gibi sonlandirilmasinda da yapilmasi gereken bazi islemler vardir. Sekil 2.3'de bu adimlar gösterilmistir. Baglanti kurulurken 3 segmentin degisilmesi yeterli iken sonlandirma isleminde 4 segmentin degisilmesi gerekmektedir. Baglantiyi sonlandirmak için istemci TCP sunucuya özel bir segment gönderir. Bu segmentin basligi içerisinde 1 olarak ayarlanmis FIN biti vardir. Istemci bu segmenti gönderdikten sonra FIN_WAIT 1 evresine geçer. Sunucu, istemcinin gönderdigi segmenti onayladigina dair bir ACK bilgisi gönderir. istemci bunu aldiginda FIN_WAIT 2 evresine geçer. Bundan sonra sunucuda baglantiyi kapatmak istedigini belirten segmenti, basliginda bulunan

FIN bitini 1 olarak ayarlayip gönderir ve LAST_ACK evresine geçer. FIN_WAIT 2 durumundaki istemci bu segmenti aldiginda TIME_WAIT evresine geçer ve sunucunun FIN segmentini onaylayan ACK segmentini gönderir. Bu ACK segmentini alan sunucu baglantisini kapatir. TIME_WAIT evresinde olan istemcide daha önceden belirlenmis olan bekleme süresinden sonra baglantisini kapatir. Bu islemlerden sonra iki uç arasindaki TCP baglantisi sonlandirilmis olur.



Sekil 2.3 Iki Uç arasindaki Baglantinin Sonlandirilmasi

2.1.3. Güvenilir Veri Transferi (Reliable Data Transfer)

Bilindigi gibi Internet'in ag katmani hizmeti (IP hizmeti) güvenilir degildir. IP katmani paketlerin sirali ve güvenilir olarak iletilmesini garanti etmez. TCP katmani güvenilir olmayan IP hizmeti üzerinde güvenilir veri transfer hizmeti yaratir [4].

TCP alindi bilgisi (ACK) ve zamanlayici kullanarak güvenilir veri transferini saglar. Alici dogru aldigi bilgiler için ACK bilgisi gönderir, ve segmentler veya onlara ait olan ACK bilgilerinin kaybolmasi veya bozulmasi durumunda gönderici segmentleri tekrar gönderir. Alicinin kayip veya kopya segmentleri belirlemesi amaciyla her bir paket için sira numarasi kullanir.

Gönderici kaybolan paketleri ve ACK'lari belirleyebilmek için bir tane zamanlayici kullanir ve bu zamanlayicinin süresi doldugunda pencere içinde en küçük sira numarasina sahip segment'i tekrar gönderir.

Sekil 2.4'de basit bir TCP göndericinin tanimi yapilmistir. Veri iletimi ve tekrar gönderim ile ilgili üç temel olay tanimlanmistir. Birinci olayda, veri üst katmandan alinip segment formatina getirilerek IP katmanina geçirilmektedir. Eger zamanlayici baska bir segment için çalismiyorsa zamanlayici baslatilir. Ikinci olay ise zamanlayicinin süresinin dolmasi durumudur. Zaman asimina neden olan segment tekrar gönderilir ve zamanlayici tekrar baslatilir. Üçüncü olayda TCP aldigi "Y" ACK bilgisini SendBase degeri ile karsilastirilir. SendBase en eski ACK bilgisi gelmemis paketi göstermektedir. TCP birikimli ACK bilgisi kullandigi için "Y" kendinden once gelen paketlerin alinmis oldugunu gösterir. Eger Y>SendBase ise ACK bilgisi gelmemis bir veya daha fazla paketin iletildigini gösterir ve SendBase degeri Y olarak atanir. Eger halen ACK edilmemis paket var ise zamanlayici tekrar baslatilir.

```
NextSeqNum = InitialSeqNum + 1
SendBase = InitialSeqNum + 1    /* == LastByteAcked + 1 */
loop (forever) {
   switch(event)
      event: Yukarıdaki uygulama katmanınadan veriler alındı
         1. NextSeqNum sıra numaralı yeni bir paket yaratıldı.
         2. if (eğer zamanlayıcı çalışmıyorsa)
            2.1. zamanlayıcıyı başlat - TimeOutInterval süresi kadar
            sonra zaman aşımına uğrar
         3. segmenti IP katmanına geçir
         4. NextSeqNum = NextSeqNum + length(data)
      event: zamanlayıcı zaman aşımına uğradı
         1. ACK bilgisi gelmemiş en küçük sıra numaralı segmenti
            tekrar yolla
         2. zamanlayıcıyı başlat TimeOutInterval süresi kadar sonra
            zaman aşımına uğrar
      event: y değerli ACK bilgisi alındı
         1. if (y > SendBase) {
               SendBase = y
               if (Eğer ACK bilgisi gelmemiş segmentler var ise)
                  restart timer
            }
} /* loop forever sonu */
```

Sekil 2.4 TCP Göndericisi Algoritmasi

2.1.3.1 Hizli Tekrar Gönderim Algoritmasi

Tekrar gönderimler için oldukça uzun olan zaman asiminin beklenmesi TCP ile ilgili problemlerden birisidir [11]. Bir segment kayboldugunda, bu uzun zaman asimi süresi, göndericiyi kaybolan paketi tekrar göndermesinden önce uzun süre bekletmektedir. Bu durumda iki uç arasindaki gecikmeleri artirmaktaydi. Neyse ki, gönderici çogunlukla paket kayiplarini, aldigi kopya (duplicate) ACK bilgilerine bakarak zaman asimi (timeout) süresini beklemeden tespit edebilmektedir. Alici bekledigi segmentin sira numarasindan daha büyük bir sira numarali segment aldigi zaman, veri akisinda bir açik tespit eder. Bu açik da bir segmentin kaybolmasindan veya segmentleri tekrar siralama isleminden kaynaklanir. TCP sirayi bozan bir segment aldigi zaman en son sirali aldigi segment için tekrar ACK bilgisi gönderir. Bu tekrar gönderilen ACK bilgilerine kopya (duplicate) ACK denilmektedir.

TCP arka arkaya birçok segmentleri gönderdigi için, bir segment kayboldugunda, büyük olasilikla arka arkaya bir çok kopya ACK bilgileri olacaktir. Eger TCP göndericisi, ayni segment için 3 tane kopya ACK bilgisi aldiginda, bu 3 tane ACK bilgisine ait olan segmenti takip eden segmentin kaybolmus oldugunu kabul eder. Bu durumda TCP, 3 tane kopya ACK bilgisi aldiginda tekrar gönderim algoritmasini (Fast Retransmit) çalistirir [13]. Tekrar gönderim algoritmasi zamanlayicinin zaman asimina ugramasini beklemeden eksik olan segmenti tekrar gönderir.

Sekil 2.5'de tekrar gönderim algoritmasinin nasil çalistigi gösterilmistir.

Sekil 2.5 Hizli Tekrar Gönderim Algoritmasi

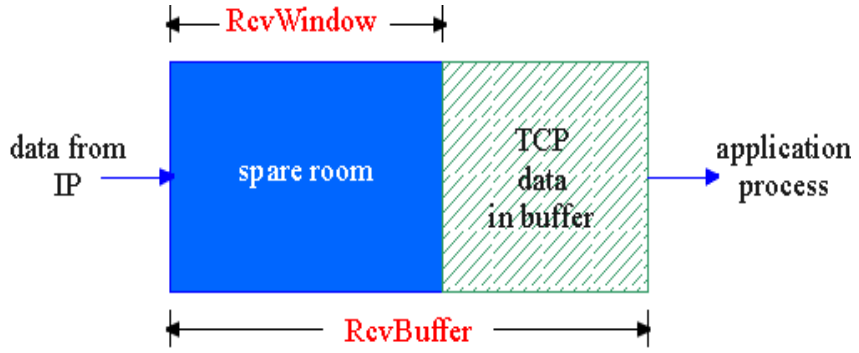
2.1.3.2 Akis Kontrolü

TCP baglantisi verileri dogru ve sirali almaya basladigi zaman, onlari alicidaki tampon bellege aktarir. Bu verilerle ilgili olan uygulama bunlari tampon bellekten alarak okur. Fakat uygulama baska görevlerle de mesgul olabilecegi için uzun süre bu verileri tampon bellekten okuyamayabilir. Eger uygulama verileri okumada çok yavas kalirsa, gönderici çok fazla ve çok çabuk veri göndererek tampon bellegin tasmasina neden olabilir.

TCP akis kontrolü hizmeti saglayarak göndericinin tampon bellekte bir veri tasmasi yaratmasini engeller. Akis kontrolü göndericinin veri gönderimi ve alicidaki uygulamanin bu verileri okuma oranlarini karsilastirir.

Akis kontrolü TCP'nin kayan pencereler (sliding window) yönteminin uygulanmasi ile basarilir. Akis kontrolü, göndericinin alicinin kabul edebileceginden fazla veriyi göndermesini engeller. Alici her bir ACK bilgisiyle beraber alabilecegi pencere genisligi (rcvWin) degerini göndericiye bildirir. Gönderici, bu degere bakarak alicinin tampon belleginde ne kadar bos yer oldugu hakkinda bilgi sahibi olur ve pencere genisligini bu degere bakarak ayarlar.

Sekil 2.6'da alcinin tampon belleginin yapisi gösterilmistir.



Sekil 2.6 TCP Alicisi Geçici Bellek Yapisi

2.1.4. Tikaniklik Kontrolü (Congestion Control)

Tikaniklik kontrolü algoritmasi [14, 15] agdaki yogunluga bakarak göndericinin iletecegi veri miktari oranini belirler. Tikaniklik kontrolü algoritmasi göndericinin aga iletilebilecegi veri miktarini gösteren tikaniklik penceresi

Congwnd) degiskenini kullanir. Göndericinin iletecegi veri miktarini belirlerken ayni zamanda alicinin durumuna da bakmak gerekir. Iletilecek veri miktari **sndWindow = LastByteSent-LastByteAcked £ min(Congwnd, rcvWin)** seklinde olmalidir. Alicinin kabul edebilecegi veri miktari (rcvWin) ile göndericinin iletebilecegi (Congwnd) degerlerinin küçük olani göndericinin pencere boyutu (sndWindow) olarak belirlenir. Eger alicinin tampon belleginin çok büyük oldugunu kabul edersek rcvWin kisitlamasini iptal edebiliriz. Böylece göndericinin veri iletim orani sadece Congwnd tarafindan sinirlandirilir.

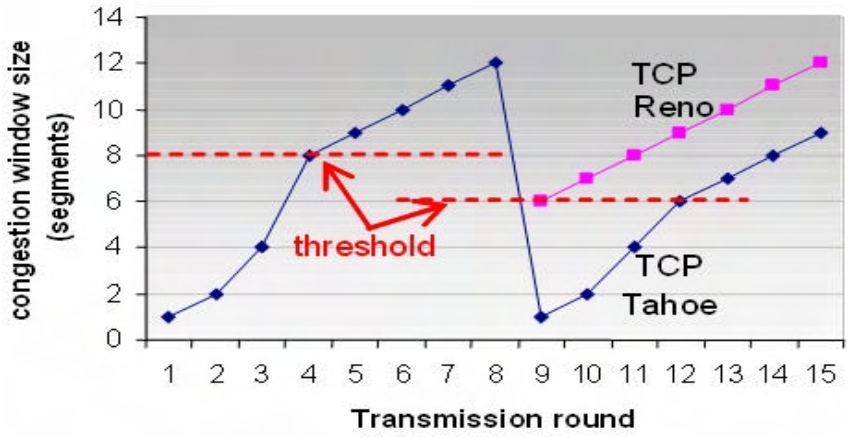
TCP'de gönderici, zaman asimi veya üç tane kopya ACK alinmasi ile bir paketin kayboldugunun belirlenmesi sonucunda, iki uç arasinda tikaniklik (congestion) oldugunu farkina varir. Tikaniklik fark edildigi zaman tikaniklik kontrol algoritmasi göndericinin veri iletim oranini azaltir.

TCP'de baglanti kuruldugu zaman Congwnd degeri 1 olarak ayarlanir ve tikaniklik belirlene kadar yani bir paket kaybolana veya zamanlayicinin zaman asimina ugramasina kadar artirilir. TCP ilk basta agdaki mevcut bant genisligi test ederek veri gönderme oranini ayarlanmaya çalisir. Baslangiçta 1 olarak ayarlanan Congwin degeri ilk paket kaybolma olayina kadar üssel olarak artar. Her bir ACK alindiginda Congwin degeri bir artirilir. TCP baglantisi kuruldugunda gönderici ilk segmenti aga gönderir ve ACK bilgisi bekler eger ACK bilgisi alinirsa gönderici 2 tane segment gönderir. Eger bu segmentler içinde ACK bilgisi gelirse Congwnd degeri alinan her bir ACK bilgisi için 1 artirilarak Congwnd degeri 4 olarak ayarlanir. Bu prosedür, göndericinin bir paketin kayboldugunu fark edene kadar devam etmektedir. Bu baslangiç sürecine **Slow Start** denilmektedir.

Bir paket kaybolma olayi belirlendiginde Congwin degeri ne olmalidir? TCP'nin Tahoe versiyonunda zaman asimi oldugunda veya üç tane kopya ACK alinmasi durumunda Congwin degeri 1 olarak ayarlanmaktadir [11]. Paket kaybolma aninda pencere boyutunu belirlemek için Threshold degiskeni kullanilir. Threshold degeri paket kaybolma olayindan hemen önce Congwin degerinin yarisi olarak atanir. Slow start tekrar basladiginda congwin degeri Threshold degerine kadar üssel olarak artar ve bu degerden sonrada dogrusal olarak artmaya devam eder. Bu olaya TCP'de **congestion avoidance** denir. TCP'nin Reno versiyonunda bu algoritma degistirilerek daha iyi bir tikaniklik

kontrol algoritmasi gelistirilmistir. TCP Reno'da üç kopya ACK alinmasindan sonra slow start olayinin baslamasi iptal edilmistir. Çünkü üç kopya ACK alinmasi kaybolandan sonra üç tane paketin iletildigini göstermekte, dolayisiyla zaman asimi durumundan farkli olarak, agin bazi segmentleri göndermeye olanagi oldugunu göstermektedir. Bunun için Reno TCP'de üç kopya ACK alinmasinda sonra Congwin degeri bire degil yariya düsürülmektedir. Congwin degeri dogrusal olarak artmaya devam etmektedir. Bu olaya **Fast Recovery** (Hizli Kurtarma) denilmektedir. Zaman asimi durumunda ise Tahoe da oldugu gibi Congwin 1 degerine düsürülmekte ve Threshold degerine kadar üssel olarak, daha sonra da dogrusal olarak artmaya devam etmektedir.

Sekil 2.7'de bir paket kaybolma olayi belirlendiginde Tahoe ve Reno TCP'nin davranislari gösterilmistir. Bu Sekil 2.7'ye göre 8. saniyede 3 tane kopya ACK bilgisinin alinmasi ile bir paket kaybolma olayi tespit ediliyor. Paket kaybolma olayi tespit edildiginde tikaniklik pencere boyutu (Congwnd) 12 degerine ulasmistir. Tahoe TCP sartsiz olarak Congwnd boyutunu 1 degerine ayarliyor ve Congwnd degeri üssel olarak arttirmaya basliyor. Bu durumda Tahoe TCP için Slow Start evresi tekrar basliyor. Reno TCP ise Congwnd degerini o anki degerinin yarisina yani 6 degerine ayarliyor ve bundan sonra dogrusal olarak Congwnd degerini artiriyor. Her iki durumda da Threshold degeri 6 olarak ayarlaniyor. Eger bir zaman asimi durumu olsaydi Tahoe ve Reno TCP Congwnd degerini 1 olarak ayarlayacakti. Daha sonra da her ikisi için slow start evresi baslayacaktir.



Sekil 2.7 TCP Tahoe ve TCP Reno'da Tikaniklik Kontrol Penceresi Büyüklügünün Degisimi

2.2. TCP'nin Gelisimi

TCP güvenilir, baglanti temelli, uçtan uça hata, akis ve tikaniklik kontrolü olan, Internet üzerinde en yaygin olarak kullanilan, ve veri paketlerinin güvenilir olarak iletilmesini saglayan tasima katmani protokolüdür. TCP'nin en temel uygulamasi yüksek hizli ve yüksek gecikmeli aglar için uygun degildir, bu yüzden protokolün performansini artirmak için birçok degisiklikler ve eklentiler yapilmistir.

W.Richard Stevens her bir modern TCP uygulamasinda dört tane temel tikaniklik kontrolü (congestion control) algoritmasi olmasi gerektigini belirtmistir [13]. Bu algoritmalar Slow Start, Congestion Control, Fast Retransmit ve Fast Recovery algoritmalaridir. En son iki algoritma TCP Tahoe gibi önceki uygulamalarin bazi eksikliklerini gidermek için gelistirilmistir. TCP Tahoe'da her bir paket kaybi olayinda **slow start** evresi basliyordu ve çok degerli olan bant genisligi bosa harcaniyordu. Modern TCP uygulamasi olarak bilinen TCP Reno, ve Newreno versiyonlari yukarida bahsedilen dört algoritmayi içermektedir.

Janey C. Hoe, TCP Reno versiyonunu degistirerek TCP tikaniklik kontrolü algoritmasinin baslangiç durumundaki performansini gelistirmistir [16]. Bu gelismeler uygun baslangiç threshold degeri bulunarak, baslangiç süresindeki paket kaybolma sayisinin minimum degere indirmeyi ve daha gelismis bir hizli tekrar gönderim algoritmasi yaratilarak birden fazla paket kayboldugunda, kaybolan paketleri tekrar gönderim zamanlayicinin zaman asimina ugramasini beklemeden kurtarmayi içermektedir. Sonuncu söylenen tikaniklik kontrolü algoritmasini TCP'nin Newreno versiyonu kullanmaktadir. Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow yüksek hizli ve yüksek gecikmeli aglarda TCP performansini artirmak için Selective Acknowledgment (SACK) seçenegini önermislerdir [17]. Bu seçenekle gönderici alicidan dogru alinmis paketlerin durumunu ögrenebilmektedir ve bu bir segmentin gidis dönüs süresinde birden fazla paketi tekrar göndermeyi saglamistir. Yüksek ve düsük gecikmeli yollarda, simülasyon temelli performans sonuçlari [18] içerisinde belgelendirilmistir, ve TCP SACK versiyonun, bir önceki TCP uygulamalari ile karsilastirildiginda ag performansini önemli derecede arttirdigi görülmüstür.

Matthew Mathis and Jamshid Mahdavi TCP SACK versiyonu güçlendirmek için FACK (Forward Acknowledgment) uygulamasini önermislerdir [19]. FACK veri kurtarma süresince TCP tikaniklik kontrol algoritmasinin performansini artirmistir.

2.3. Veri Transferinin Hizlandirilmasina Yönelik Uygulamalar

Internet üzerinde FTP, HTTP, TELNET gibi TCP'yi kullanarak veri iletimi yapan bir çok uygulama vardir. Genisleyen Internet ortaminda TCP'nin talepleri karsilamada yetersiz kalmasi, veri transferinin hizlandirilmasi için yapilan çalismalari tetiklemistir.

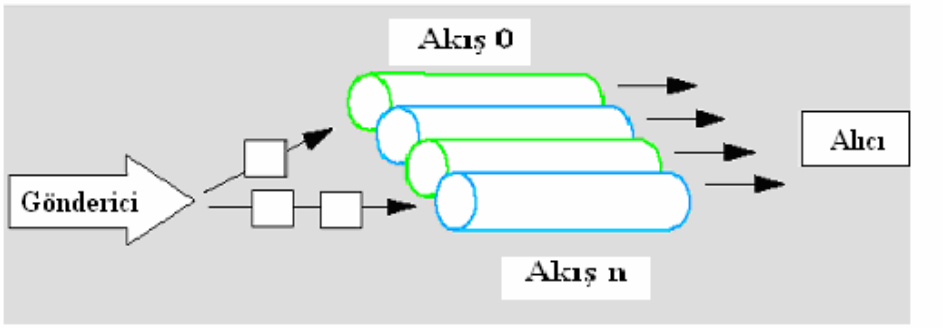
Örnek olarak bir FTP uygulamasi olan Flashget programi bir dosyayi Internet üzerinden indirirken o dosyayi parçalara ayirir ve iki bilgisayar arasinda birden fazla TCP baglantisi kurarak parçalarin her birini farkli baglanti üzeriden alir [6]. Her bir baglanti üzerinden yine sirali ve güvenilir veri iletimi yapilmaktadir. Indirme islemi baglantilar üzerinden es zamanli olarak yapilmaktadir. Indirme islemi bittiginde veri parçalari sirali olarak birlestirilerek veri bütünlügü saglanmaktadir. Flashget, birden fazla TCP baglantisi kurarak veri bant genisligini maksimum sekilde kullanmayi amaçlar ve veri indirme hizini %100 - %500 arasinda artirir. Flashget programinda veri iletiminin hizlandirilmasina karsin birden fazla TCP baglantisi kurulmasi bilgisayarlar ve ag üzerindeki yükün artmasina sebep olmaktadir. Bu çalismada TCP protokolü üzerinde herhangi bir degisiklik yapilmamistir, sadece TCP baglanti sayisi artirilarak iletimin hizlandirilmasina çalisilmistir.

2.4. Veri Transferinin Performansini Artirmak Için Gelistirilen Protokoller

TCP ve UDP iletim protokollerinin açiklarini kapatarak veri transferi performansini artirmak için yeni bir iletim protokolü olan SCTP (Stream Control Transport Protocol) gelistirilmistir. SCTP ilk baslarda haberlesme ve e-ticaret sistemleri için telefon sinyallerinin IP aglari üzerinden tasinmasi için tasarlanmistir. Devam eden çalismalarla SCTP, TCP gibi genel amaçli bir tasima protokolü olmustur.

SCTP protokolü TCP ve UDP'den farkli olarak daha gelismis iletim teknikleri sunmaktadir. TCP'nin kati bir sekilde sirali iletim yapmasindan

kaynaklanan “head-of-line blocking” problemi iletimde gecikmelere sebep olmaktaydi. Bu problem bir paket kaybolmasi ile paket sirasinin bozulmasi durumunda TCP’nin kaybolan paketin basarili bir sekilde iletilmesine kadar yeni bir paket iletimine izin vermemesidir. SCTP protokolü, birbirinden bagimsiz birden fazla mantiksal veri akisi (multistreaming) kullandigi için kismen TCP’ye göre “head-of-line blocking” problemin in yaratmis oldugu gecikmeyi azaltmistir. Sekilde görüldügü gibi iki uç arasinda birden fazla mantiksal kanallar vardir bu kanallarda akan veriler birbirlerinden bagimsizdirlar. Her kanal kendi içerisinde sirali olarak iletim yapar ve bir kanalda kaybolan veri diger kanallardaki veri akisini etkilememektedir.



Sekil 2.8 SCTP Çoklu Veri Akisi

Kanallar içerisinde her bir veri parçasi için veri sirasi ve güvenilirligi akis sira numarasi (Stream Sequence Number SSNs) ile saglanir. “Head-of-line blocking” problemi sadece kanallar arasinda engellenmistir. Yani bir kanalda bir veri kayboldugu zaman sadece o kanaldaki kaybolan veri tekrar basarili bir sekilde iletilene kadar veri iletimi engellenecektir, diger kanallardan veri akisi devam edecektir. SCTP protokolünde güvenirligi saglamak için verilerin hangi kanalda olduguna bakilmaksizin her bir veri için tekil bir iletim sira numarasi (Transmission Sequence Number kullanilir,TSN) kullanilir. SSN sadece kanallar içerisindeki veri sirasi ve güvenirligini saglarken TSN, TCP’deki sira numarasi gibi, bütün kanallarda iletilen verilerin sirasi ve güvenirligini saglamaktadir.

Sonuç olarak SCTP’de de “head-of-line blocking” probleminin neden olmus oldugu gecikme tamamen engellenememistir. Bizim yaptigimiz uygulamada ise

sirali olmayan ve güvenilir iletim saglanarak "head-of-line blocking" problemi tamamen engellenmistir.

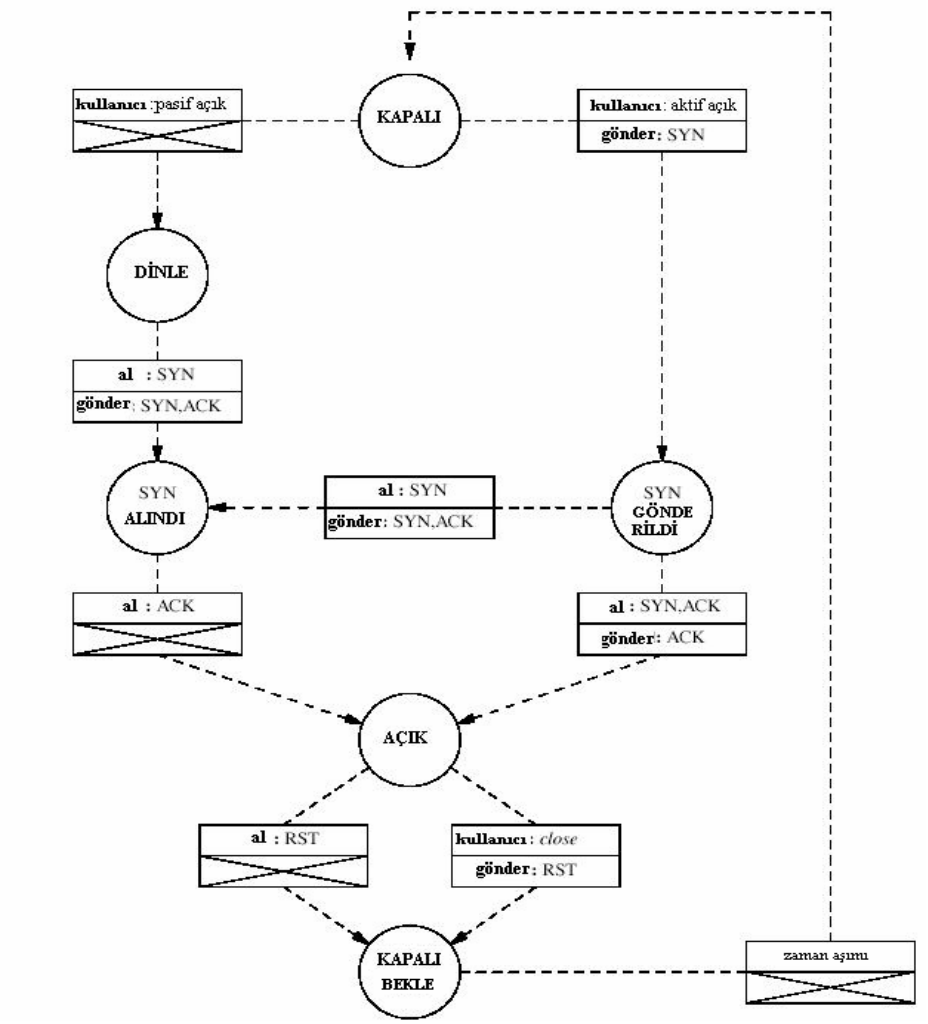
Bir baska IP katmani üzerinde çalisan iletim katmani protokolü RDP (Reliable Data Protocol) 1984 yilinda gelistirilmistir [8]. RDP protokolü UTCP gibi hem sirali hem de sirali olmayan veri iletimi yapabilmektedir. Bu protokolü gelistirirken tasarimcilarin amaci kaybolma orani daha düsük aglarda büyük hacimli veri transferleri yapabilecek basit bir protokol gelistirmekti. TCP'de oldugu gibi bir tikaniklik kontrolü mekanizmasi yoktur. IETF, RDP protokolünü bir deneysel protokol olarak kabul etmektedir. RDP hiç bir zaman TCP gibi yaygin olarak kullanilmamistir. Çogunlukla deneysel amaçlar için kullanilmaktadir.

RDP protokolünün durum diyagrami ve nasil çalistigi Sekil 2.9'da gösterilmistir. Buna göre bir baglantiyi açmak için iki yol vardir. Esler kapali durumda (closed state) baslarlar ve baglanti durumuna geçmek için aktif (active) veya pasif (passive) açma (open) istegini kullanirlar. Baglanti çift yönlüdür ve data alindigi anda ACK bilgisi gönderilir. Iki tarafta baglantiyi kapatmak için istekte bulunabilir.

RDP üst katmandan aldigi veri bloklari bir alt katman olan ag katmanina geçirir. TCP'de oldugi gibi gönderebilecegi paket sayisini sinirlayan pencere boyutu (window size) degiskeni kullanilir. RDP'de her bir segmente 32 bitlik bir sira numarasi verilir. RDP segmentlerin iletimini garantilemek için pozitif ACK bilgisi ve tekrar gönderim kullanir. Dogru olarak iletilen segmentler için ACK bilgisi gönderilir. Hasarli segmentler tarafindan atilirlar ve bunlar için ACK bilgisi gönderimi olmaz. Zamaninda alici bilgisayarlar tarafindan ACK bilgisi gönderilmeyen segmentler tekrar gönderilir [20].

RDP iki tip ACK bilgisi kullanir. Birincisi TCP'de oldugu gibi birikimli ACK bilgisidir. Bir birikimli ACK bilgisi belirtilen sira numarasina kadar olan bütün segmentlerin alindigini gösterir. Birikimli ACK bilgisi sirali paket iletimi hizmetinde kullanilir. Ikincisi de genisletilmis veya Birikimli olmayan ACK bilgisidir. Bu ACK bilgisi alicinin sira disinda gelen segmentleri onaylamasini saglar. Bu tip onaylama RDP basligindaki EACK bayraginin kullanilmasi ile saglanir. Bu tip onaylamanin islevi çok basittir. UTCP'de oldugu gibi alici aldigi

sira ile segmenti onaylar. Bir kullanici islemi baglantiyi baslattiginda, baglantida kullanilacak ACK bilgisi tipinde herhangi bir kisitlama yapamaz. Alici sira disi segment onaylamasini seçmeyebilir. Diger taraftan göndericide sira disi segment onaylama isini iptal etmeyi seçebilir.



Sekil 2.9 RDP Durum Diyagrami [20]

RDP protokolünde segmentler iki nedenden dolayi kaybolmus olarak kabul edilir. Ilki segmentler iletilirken kaybolmus veya zarar görmüs olabilirler. Ikincisi de segment alici tarafindan atilmis olabilir. Pozitif alindi bilgisi mekanizmasi sadece segmentin dogru olarak alindiginda ACK bilgisi gönderilmesini gerektirir.

Eksik paketleri belirlemek amaciyla RDP her bir segment için bir tekrar gönderim zamanlayici tutar. Bir segment için ACK bilgisi alindiginda o segment için tutulan zamanlayici iptal edilir. Eger zamanlayici ACK bilgisi gelmeden önce zaman asimina ugrarsa o segment tekrar gönderilir ve zamanlayici tekrar baslatilir.

RDP protokolünde TCP ve UTCP'de oldugu gibi bir tikaniklik kontrolü mekanizmasi yoktur. Sadece çok basit bir akis kontrolü algoritmasi vardir. Buna göre iki uç arasinda baglanti açildiginda, pencere boyutu belirlenir ve baglanti kapanincaya kadar degeri degismez.

RDP protokolü TCP gibi yaygin olarak kullanilmamaktadir. Daha çok deneysel amaçlarda kullanilmaktadir. RDP'nin yayginlasmamasinin nedenlerinden biride tikaniklik kontrolü mekanizmasinin olmamasi ve çok basit bir akis kontrolü mekanizmasina sahip olmasidir. Bu çalismada amacimiz tikanik kontrolü ve akis kontrolü mekanizmasina sahip ve dünyada en yaygin olarak kullanilan TCP'ye RDP'nin verimliligini ekleyerek TCP'nin performansini artirmaktir. Gelistirmis oldugumuz UTCP, gelismis bir akis ve tikaniklik kontrolüne sahip, güvenilir fakat sirali olmayan iletim yapan bir protokoldür.

Bir baska iletim katmani protokolü NETBLT (Network Bulk Data Transfer) 1985 yilinda gelistirilmistir [9]. NETBLT özellikle genis aglar üzerinde büyük hacimli veri transferi yapmak için gelistirilmistir. TCP gibi IP aglari üzerinde çalismasi için tasarlanmistir. TCP den farkli olarak akis kontrolü pencere kontrolü ile degil de veri transfer hizi kontrolü ile saglanmaktadir. Hiz kontrol parametresi baglanti kurulma islemi sirasinda ve baglanti süresince periyodik olarak istemci ve sunucu arasinda belirlenir. Belirlenen hiz kontrol parametresi ag geçidinin veya en yavas baglantinin hizini asmamalidir. NETBLT protokolünde veriler TCP'deki gibi bir bayt akisi seklinde degil büyük sabit boyutlu veri bloklari seklinde siralanir. Uygulama katmani bu veri bloklarini tasima katmanina bitisik olarak geçirmektedir. NETBLT protokolü ACK bilgisi beklemeden verileri bloklar halinde gönderir. Veri güvenirligi blok blok kontrol edilmektedir. Bir blok içindeki bütün paketler iletildiginde alici blok içinde bozulmus veya kaybolmus olan paketlerin tekrar gönderilmesini isteyecektir. Alici tabanli bir veri güvenirligi kontrolü oldugundan gönderici TCP'de oldugu gibi bir zamanlayici tutmaz. NETBLT protokolünde her paket için ACK bilgisi gönderilmez. Sadece kaybolan

veya bozulan paketlerin tekrar gönderilmesini saglayan bir bilgi gönderilir. Gönderici bu bilgiye bakarak kaybolan paketleri tekrar gönderir. NETBLT her ne kadar yüksek gecikmeli aglarda gecikmeyi azaltsa da ara ag geçitlerindeki kisitlamalardan dolayi kullanimi yayginlasmamistir. Internet üzerinde desteklenmedigi için genellikle deneysel amaçlari için kullanilmaktadir.

3. ÇÖZÜMLER ve UYGULAMALAR

3.1. TCP Nasil Çalisir?

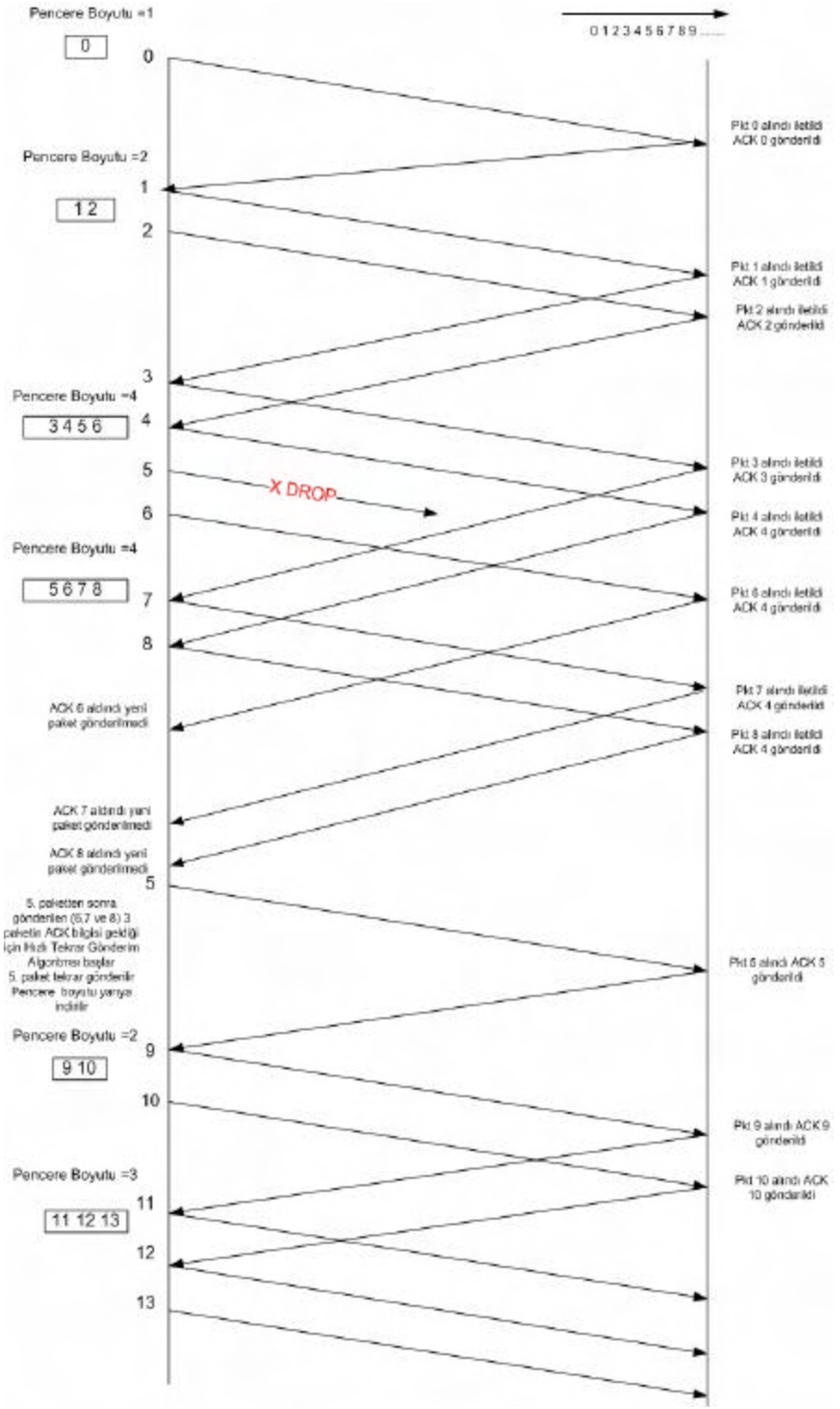
TCP'nin sirali paket iletimi yapmasindan kaynaklanan "head-of-line Blocking" problemi, paket iletiminde iki uç arasinda uzun gecikmelere sebep olmaktadir. Bu tezde bu gecikmeleri engellemek için, TCP uygulamasinin bazi algoritmalari degistirilerek TCP'nin güvenir ve sirali olmayan paket iletimi yapmasi saglanmistir. Paketleri siralama islemi uygulama katmanina birakilmistir.

Bu çalismada TCP'de veri akis kontrolünü, güvenirligini ve sirali iletimi saglayan kayan pencere protokolü ve tikaniklik kontrolü algoritmalarindan biri olan tekrar gönderim algoritmasi degistirilmistir.

TCP'deki Kayan Pencere Protokol'ünün iki temel görevi; veri iletiminin güvenilirligi ve akis kontrolünü saglamaktir. Kayan Pencere Protokol'ünde, her bir pakete sira numarasi verilir ve her bir paket aliciya ulastiginda göndericiye ACK bilgisi gönderilir. Eger gönderici kendisine gelen ACK bilgilerine bakarak bir paketin kayboldugunu anlarsa o paketi tekrar gönderir. TCP, bir paketin kayboldugunu, o paket için tutulan zamanlayicinin zaman asimina ugramasi veya o paket için üç tane ayni ACK bilgisinin gelmesi sonucunda anlar. Bu iki durumda da kaybolan paket tekrar yollanarak güvenilirlik saglanir. Kayan Pencere Protokolü ile akis kontrolü saglayarak veri bant genisligini daha verimli kullanir. Alici, gönderdigi her ACK bilgisi içinde kendi kabul edebilecegi bayt sayisini (recvWin) göndericiye iletir. Gönderici, bu aldigi deger ile tikaniklik kontrolü algoritmasinin agin trafigine bakarak göndericinin aga iletebilecegi veri miktarini belirten tikaniklik pencere boyutu (CongWnd) degeri ile karsilastirarak küçük olani kendi pencere boyutu (sndWindow) olarak ayarlamaktadir. Yani iletilecek veri miktari **sndWindow = LastByteSent-LastByteAcked £ min(Congwnd, rcvWin)** seklinde hesaplanmaktadir.

Burada pencere boyutu, göndericinin, pencere içerisindeki ilk paket için ACK alinmasini beklemeden gönderilebilecegi paket sayisini gösterir. Bunun sonucunda TCP göndericisinin, alicinin ve agin kabul edemeyeceginden fazla bayt veya paket göndermesi engellenmis olur.

Sekil 3.1'de Newreno TCP'nin güvenilir veri transfer protokolünün nasil çalistigi gösterilmistir. TCP baglantisi kuruldugunda baslangiç pencere boyutu 1 olarak belirlenmekte ve bir paket kaybolana kadar her alinan ACK bilgisi için pencere boyutu bir artirilmaktadir. Sekil 3.1'deki örnekte bir paket kayboldugunda, Newreno TCP'nin kaybolan paketi tekrar göndermek için nasil bir yöntem uyguladigi gösterilmistir. Sekil 3.1'de görüldügü gibi paketler sirali olarak üst katmana iletilip ACK bilgisi göndericiye iletilmektedir. Alinan her paketten sonra pencere bir adim saga dogru kaymaktadir. Sekil 3.1'de 5 sira numarali paketin kaybolmus oldugu varsayilmistir. Bu nedenle 6, 7 ve 8 sira numarali paketler alindigi halde 5 sira numarali paket alinmadigi için pencere saga dogru kaymayacak ve 5 numarali paket tekrar gönderilene kadar yeni paket gönderimi yapilmayacaktir. Bu bekleme süresinde veri iletim hattinin bos kalmasi, bant genisliginin verimli kullanilmasini engellemektedir. Bu bekleme TCP'nin kati bir sekilde sirali paket iletimi yapmasindan kaynaklanmaktadir. 2 numarali paket tekrar gönderilip ACK bilgisi geldikten sonra, pencere saga dogru kayarak yeni paket gönderilmektedir. Alici, 5 numarali paketin gelmedigini belirtmek amaciyla 5 numarali paketten sonra gelen paketler için de ACK 4 bilgisini gönderir. ACK 4 bilgisi alicinin 5 sira numarali paketi bekledigini göstermektedir. Kaybolan 5 numarali paket için üç tane ACK bilgisi geldiginde TCP'nin Hizli Tekrar Gönderim Algoritmasi çalismakta ve pencere sayisi yariya indirilerek 5 numarali paket tekrar iletilmektedir. TCP üç tane ayni ACK bilgisinin alinmasindan sonra pencere sayisini yariya indirerek veri iletimine devam etmektedir [11]. Sekil 3.1'de de de görüldügü gibi 5 sira numarali paket kaybolduktan sonra 6,7 ve 8 için ACK bilgisi gelmesine ragmen pencere boyutu artirilmamistir. Pencere boyutu hizli tekrar gönderim algoritmasinin bitimine kadar sabit kalmistir. Hizli tekrar algoritmasindan sonra pencere boyutu yariya indirilip dogrusal olarak artirilmaktadir. Pencere içerisindeki bütün paketler için ACK bilgisi geldiginde pencere boyutu 1 arttirilmaktadir. Sekil 3.1'de de görüldügü gibi pencere içerisinde bulunan 9 ve 10 sira numaralari paketler için ACK bilgisi geldiginde pencere boyutu 1 arttirilmaktadir. Bu tekrar gönderim algoritmasindan sonra pencere boyutunun dogrusal olarak artma sürecine **congestion avoidance** asamasi denilmektedir.



Sekil 3.1 TCP'nin Güvenilir Veri Iletim Algoritmasi

3.2. UTCP Nasil Çalisir?

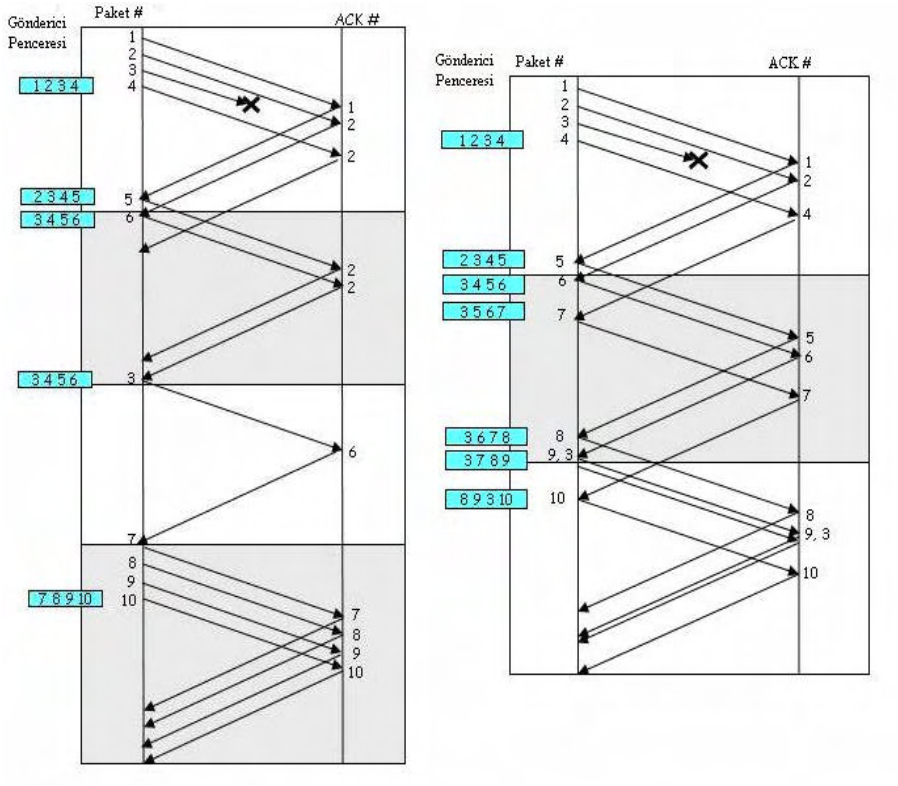
3.2.1. Neden UTCP Uygulamasi?

Sekil 3.2'de TCP ve UTCP'nin veri iletim mekanizmalari gösterilerek karsilastirilmistir. Pencere (window) boyutu 4 olarak belirlenmis ve 3 sira numarali paketin kayboldugu varsayilmistir. Ilk olarak Sekil 3.2(a)'da gösterilen TCP incelenirse 1, 2, 3 ve 4 numarali paketlerin gönderildigi görülmektedir. Alici 1 ve 2 sira numarali paketi almis ve onlara ait ACK bilgisini göndermistir. Paket 3 kayboldugu için alici 4, 5 ve 6 numarali paketleri aldiginda yine 2 numarali paketin ACK bilgisini göndermistir. 2 numarali paket için ACK bilgisinin tekrar gönderilmesi 3 numarali paketin iletilmedigin gösterir. Sekil 3.2(a)'da görüldügü gibi gönderici, 4, 5 ve 6 numarali paketler için ACK bilgisi almasina ragmen yeni paket gönderememistir. Çünkü veri penceresinin ilk paketi olan 3 numarali paket kayboldugundan, pencere bu paket dogru olarak iletilinceye kadar kaymayacak ve dolayisiyla gönderici yeni paket gönderemeyecektir. 6 numarali paketin alinmasindan sonra gönderilen ACK bilgisi ile 3 numarali paket için gelen kopya ACK sayisi üç olacagindan hizli tekrar gönderim (Fast Retransmit) algoritmasi baslar ve kaybolan 3 numarali paket tekrar gönderilir. Üçüncü RTT (Round Trip Time) sirasinda sadece 3 numarali paket iletilip ACK bilgisi alinmistir. Alici 4,5 ve 6 numarali paketleri alip arabellege koydugu için 3 numarali paket geldikten sonra 6 numarali paket için birikimli ACK bilgisi gönderir. bu da gönderici penceresinin saga dogru kayarak dört yeni paket göndermesine olanak saglar. Bu paketler de iletilip ACK bilgileri alindiginda transfer tamamlanacaktir. Sekil 3.2(a)'da görüldügü gibi transfer 4 RTT süresinde tamamlanmistir.

Dikkat edilirse 3 numarali paketin kaybolmasi "head-of-line blocking" problemine sebep olmustur. Bu yüzden ikinci RTT süresinde 3 numarali paket pencerenin basinda oldugu için sadece 2 yeni paket gönderilmistir. 3 numarali paket, pencerenin ilerlemesini engellediginden ve ilk olarak kaybolan 3 numarali paketin dogru olarak iletilmesi gerektiginden, 4 numarali paket için ACK bilgisi alindiginda yeni paket gönderilememistir. Bu durum 5 ve 6 numarali paketler içinde ACK bilgisi geldiginde de geçerlidir. Bu paketler sadece göndericinin 3 numarali paketin kayboldugunu fark etmesine ve 3 numarali paketi tekrar

göndermesine sebep olmustur. Bu yüzden 4 paket olan baglanti kapasitesinin sadece %25'i kullanilabilmistir.

Sekil 3.2(b)'de UTCP'nin veri iletim mekanizmasi gösterilmistir. Buna göre TCP'de oldugu gibi 1, 2, 3 ve 4 numarali paketler gönderilmis ve 3 numarali paketin kaybolmus oldugu varsayilmistir. 1 ve 2 numarali paketler dogru olarak iletilmis ve ACK bilgileri alinmistir. Alici 4 numarali paketi aldiginda diger paketlerin durumuna bakmaksizin 4 için ACK bilgisini gönderir. TCP'den farkli olarak birikimli ACK bilgisi yoktur. Her paket için bireysel ACK gönderilir. Paket 4 için ACK bilgisi göndericiye geldiginde, gönderici yeni bir paketi (7 numarali paket) hemen gönderir. Bu sonuçla pencere içerisinde mantiksal kanallar oldugu düsünülmekte ve bir ACK bilgisinin alinmasiyla yeni bir paket bosalan kanaldan hemen gönderilmektedir. Her bir ACK bilgisinin gelmesi yeni bir paketin gönderilmesini saglamaktadir. Sekil 3.2(b)'de görüldügü UTCP'de ayni dosya için veri transferi 3 RTT zamaninda bitmistir. TCP ile karsilastirildiginda %25 daha hizli çalismaktadir.



(a) TCP **(b) UTCP**

Sekil 3.2 TCP ile UTCP Protokollerinin Karsilastirilmasi

Paket 6 için ACK bilgisinin alinmasi ile 3 numarali paket tekrar gönderilmistir. Sekil 3.2(b)'de de görüldügü gibi 3 numarali paketten sonra gelen 4, 5 ve 6 numarali 3 paketin ACK bilgisinin gelmesinden sonra 3 numarali paket tekrar gönderilmistir. Dikkat edilirse 3 numarali paket için kopya ACK alinmasi söz konusu degildir. UTCP'de hizli tekrar gönderim algoritmasi TCP'ninkine benzemektedir. TCP'de hizli tekrar gönderim algoritmasi, pencerenin ilk paketi için 3 tane kopya ACK bilgisi alinmasi ile baslar, UTCP'de ise pencere içerisinde bulunan en küçük numarali paketten sonra gelen 3 paketin ACK bilgisinin alinmasi ile baslar. Sonuç olarak TCP paketleri pencere içerisindeki sira numarasina göre siralarken, UTCP paketlerin iletim zamanini dikkate alarak siralama islemi yapar.

3.2.2. Tikanik Kontrol Algoritmasi (Congestion Control) Olmayan UTCP

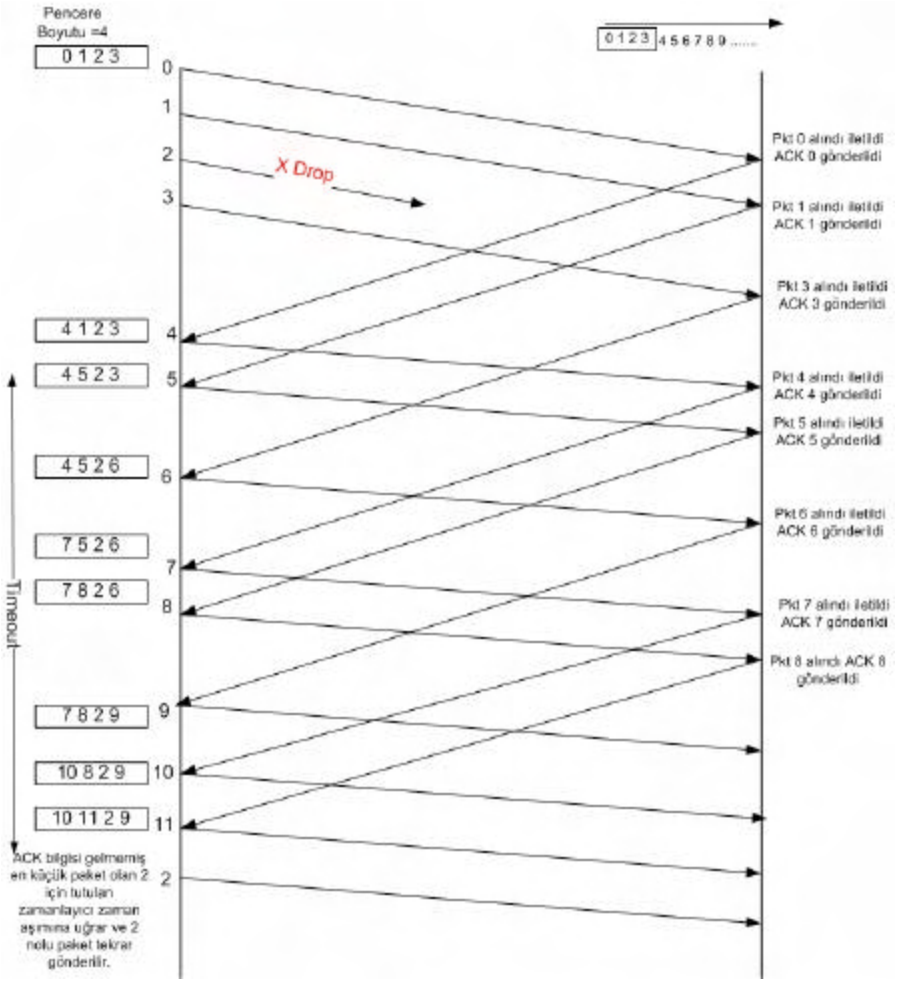
Çalismamizda Newreno TCP'nin degistirilmesi ile elde edilen UTCP protokolünde, iki uç arasindaki baglantinin birbirinden bagimsiz mantiksal kanallara ayrildigi düsünülmektedir. Bu kanal sayisi pencere sayisina esittir. Her bir kanal üzerinden paketler birbirinden bagimsiz olarak gönderilmektedir. Bir kanaldan gönderilen paket için ACK bilgisi geldigi anda, o kanal üzerinden yeni paket diger kanallarin durumuna bakilmaksizin hemen gönderilmektedir

Çalismamizin ilk asamasinda Newreno TCP'deki tikaniklik kontrolü algoritmasini çikartarak pencere sayisini isletim sisteminin tavsiye ettigi degere sabitlemis bulunmaktayiz. Sekil 3.3'teki örnekte pencere boyutunu 4 olarak alip uygulamamizin nasil veri iletimi yaptigi gösterilmistir

Degistirilmis olan Newreno TCP, güvenilir ve sirali olmayan iletim yapmaktadir. Sekil 3.3'te görüldügü gibi paketleri siralama islemi yapilmadigi için hiçbir bekleme söz konusu degildir. Alici, paketleri alir almaz paketin sira numarasina bakmazsizin uygulama katmanina göndermektedir. Paketlerin her birinin sira numarasi oldugu için siralama islemi uygulama katmani tarafindan kolaylikla yapilabilir.

Sekil 3.3'te 2 sira numarali paketin kayboldugu varsayilarak gelistirilen yaklasimimizin nasil bir yöntem izlendigi gösterilmistir. Uygulamamizda her zaman pencere içerisindeki ACK bilgisi gelmemis en küçük numarali paket için zamanlayici tutulur. Uygulamada görüldügü gibi 2 numarali paketin kaybolmasi

diger paketlerin gönderilmesini engellememektedir. UTCP'de her bir ACK alindiginda yeni bir paket diger paketlerin durumuna bakilmadan gönderilir. Sekil 3.3'te görüldügü gibi pencere içerisinde 4,5,2 ve 3 nolu paketler var iken en küçük sira numarali olan 2 nolu paket için zamanlayici tutulur. Zaman asimi süresinde 2 numarali paket için ACK bilgisi alinmazsa bu paketin kaybolmus oldugu kabul edilip tekrar gönderilir. Sekil 3.3'te 2 nolu paket için zaman asimi süresinde ACK bilgisi gelmedigi için 2 numarali paket tekrar gönderilir.



Sekil 3.3 Tikanik Kontrol Algoritmasi Olmayan UTCP'nin Güvenilir Veri Iletim Protokolü

Kaybolan paketleri tekrar göndermek için uzun olan zaman asimini süresinin beklenmesi performansi düsürmektedir. Bunun için UTCP

uygulamamiza bir sonraki bölümde bahsedecegimiz, kaybolan paketleri zaman asimi süresini beklemeden göndermeyi saglayan hizli tekrar gönderim algoritmasi eklenmistir. .

Uygulamamizdaki UTCP göndericisinin algoritmasi sekil 3.4'te basitlestirilmis olarak gösterilmistir. UTCP göndericisinde uygulama katmanindan veriler alindiginda her bir paket için UTCP segmenti olusturulur. Her bir segmente bir sira numarasi ve timestamp degeri verilir. Sonra da segment iletilmek üzere IP katmanina geçirilir. Eger bir zamanlayici çalismiyorsa o paket için zamanlayici çalistirilir. UTCP göndericisinde ikinci olay zamanlayicinin zaman asimina ugrama durumudur. TCP'de oldugu gibi UTCP göndericisinde de gibi en küçük sira numarali paket için bir tane zamanlayici tutulur. Uygulamamizda da bu durum aynidir. Eger zamanlayici zaman asimina ugramissa en küçük timestamp degerine sahip olan paket tekrar gönderilir.

```
NextSeqNum = InitialSeqNum + 1
SendBase = InitialSeqNum + 1
loop (forever) {
  switch(event)
    event: Yukarıdaki uygulama katmanınadan veriler alındı
      1. NextSeqNum sıra numaralı UTCP segmenti yarat.
        packetTimeStamp=time()
      2. if (eğer zamanlayıcı çalışmıyorsa)/* ilk giden paket için */
        SendBase= NextSeqNum
        zamanlayıcıyı başlat
        end if
      3. segmenti iletim için IP katmanına geçir
      4. NextSeqNum = NextSeqNum + length(data)
    event: Timer Timeout zamanlayıcı zaman aşımına uğradı
      1. En küçük TimeStamp değerine sahip paketi tekrar yolla
        packetTimeStamp=time()
        SendBase=PacketWithSmallestTimeStamp;
      2. Restart Timer /* zamanlayıcıyı Tekrar başlat */

      } /* loop forever sonu */
```

Sekil 3.4 UTCP Göndericisi Algoritmasi

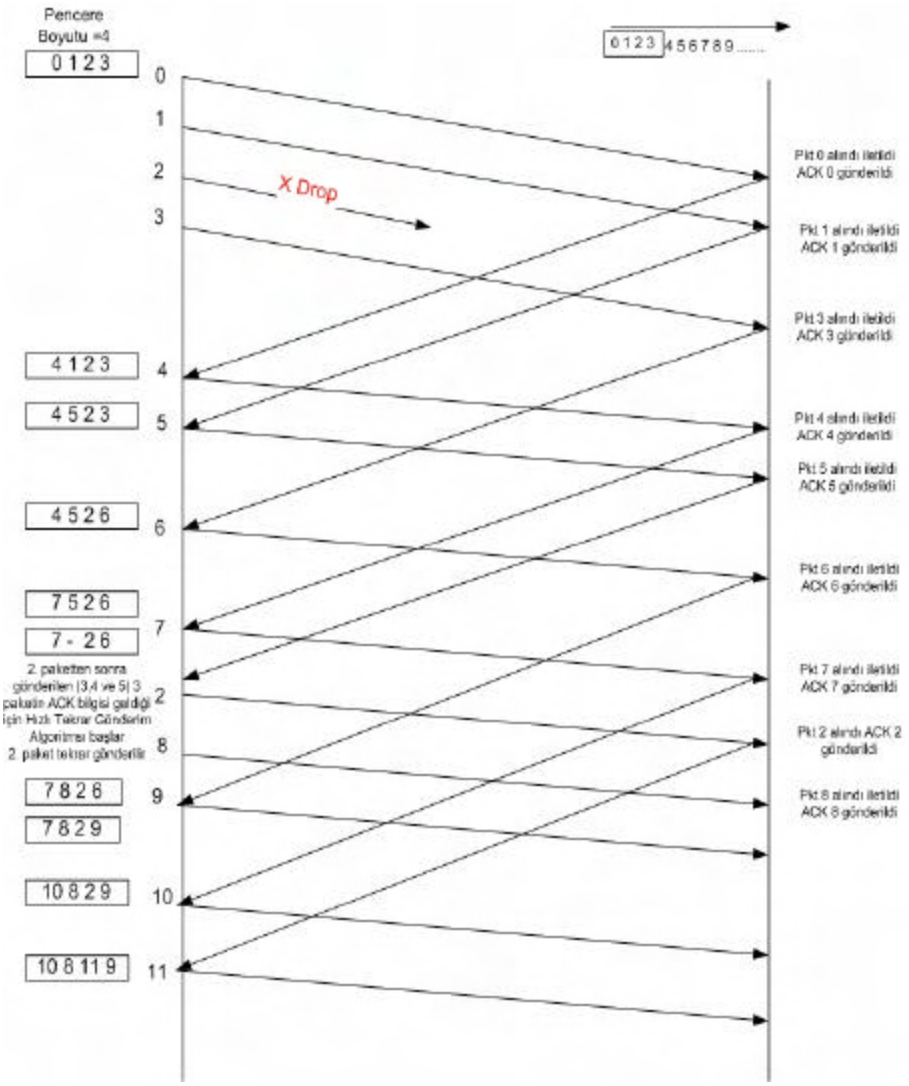
3.2.3. UTCP'de Hizli Tekrar Gönderim Algoritmasi (Fast Retransmit Algorithm)

Çalismanin bu kisminda, TCP'nin Hizli Tekrar Gönderim Algoritmasi üzerinde degisiklik yapilmistir. TCP'nin ilk versiyonlarinda, bir paketin tekrar gönderilmesi için o paket için tutulan zamanlayicinin zaman asimina ugramasi gerekiyordu. Bu sürenin çok uzun olmasi, kayip paketin tekrar gönderilmeden önce uzun beklemelere neden olmaktaydi. Bu bekleme süresini azaltmak için Hizli Tekrar Gönderim Algoritmasi gelistirilmistir. Hizli Tekrar Gönderim Algoritmasi, zaman asimi süresi dolmadan kayip paketin tekrar gönderilmesini saglar. TCP'de alici taraf, sirayi bozan bir paket aldigi zaman hemen en son sirada aldigi paket için bir ACK bilgisi gönderir. Bu ACK bilgisi, sirayi bozan bir paket alindigini ve hangi sira numarali paketin beklenildigini gösterir. Ancak TCP, ilk önce almis oldugu ayni ACK bilgisinin kaybolan bir paketten mi, yoksa paketleri siralama isleminden mi kaynaklandigini bilemez [13]. Üç tane ayni paket için ACK bilgisinin alinmasi durumunda, TCP bu paketin kaybolmus oldugunu kabul edip, bu paket için tutulan zaman asimi süresini beklemeden paketi tekrar gönderir.

Gelistirilen UTCP yaklasiminda, paketlerin sirali olup olmadigi kontrol edilmedigi için ayni ACK bilgisinin gönderilmesi söz konusu degildir. Göndericide ACK bilgisi gelmemis en küçük sira numarali paket için zamanlayici tutulur. Eger kendisinden sonra gelen 3 paketin alindi bilgisi geldiyse bu paket kaybolmus olarak kabul edilir ve zaman asimi süresi beklenmeksizin tekrar gönderilir.

Sekil 3.5'te 2 nolu paketin kaybolmus oldugu kabul edilip hizli tekrar gönderim algoritmasinin nasil çalistigi gösterilmistir. UTCP'de pencere içerisinde ACK bilgisi alinmamis en küçük paket için bir sayaç tutulur. Bu sayaç kendisinden sonra gelen paketler için ACK bilgisi geldiginde bir artirilir. Sayaç degeri 3 oldugunda bu paket kaybolmus olarak kabul edilip hizli tekrar gönderim algoritmasi baslatilir ve kaybolan paket tekrar gönderilir. Sekil 3.5'te 2 numarali paketten sonra gelen 3 paket için (3, 4 ve 5 numarali paketler) ACK bilgisi geldiginde hizli tekrar gönderim algoritmasi baslar ve 2 nolu paket tekrar gönderilir. Hizli tekrar gönderim algoritmasi UTCP'ye uzun zaman asimi süresi

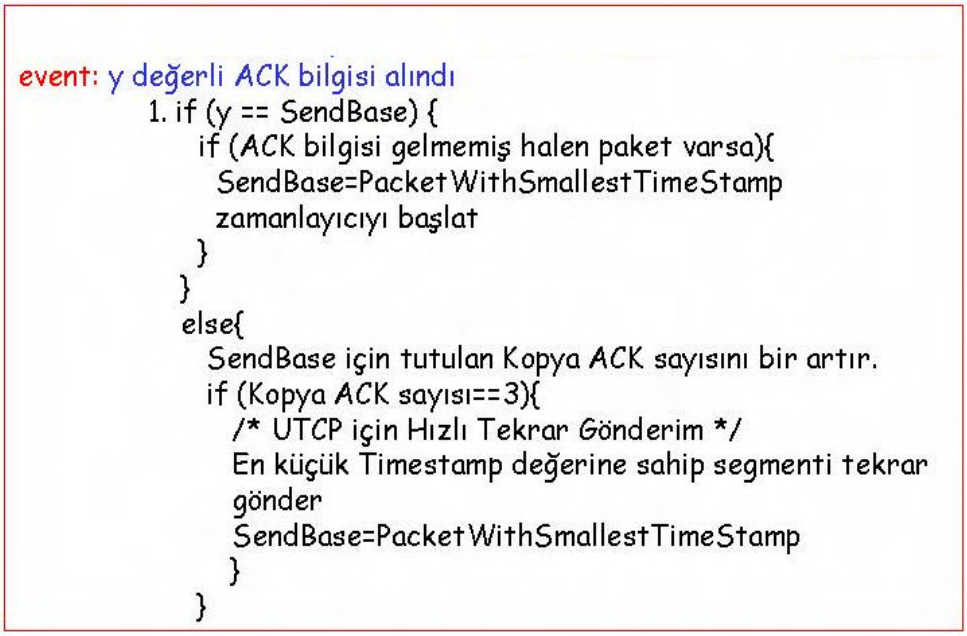
beklenmeden kaybolan paketleri tekrar gönderebilme olanagi saglamistir. Bunun sonucunda veri iletiminde meydana gelen gecikmeler azaltilarak performans artirilmistir.



Sekil 3.5 UTCP'de Hizli Tekrar Gönderim Algoritmasi Uygulamasi

Sekil 3.6'da UTCP'nin hizli tekrar gönderim algoritmasi basit sekilde gösterilmistir. Bu algoritmaya göre gönderilen paketler için ACK bilgisi alindiginda, öncelikle bu ACK bilgisinin en önce gönderilen paket için olup olmadigi kontrol edilir. Eger böyleyse sendbase degeri gönderilmis fakat ACK bilgisi gelmemis en küçük zamanli (timestamp) paket olarak atanir. Zamanlayici

bu sendbase segmenti için tekrar baslatilir. Eger gelen ACK bilgisi sendbase segmentine ait degilse bu bir paketin kaybolmus olabilecegini gösterir. Sendbase için tutulan kopya ACK sayisi bir artirilir. Eger sendbase segmentinden sonra gelen üç paketin ACK bilgisi gelirse yani kopya ACK sayisi 3 olursa sendbase segmentinin kaybolms oldugu kabul edilir ve bu segment tekrar yollanir. Sendbase segmenti yine en küçük timestamp degerine sahip olan segment olacak sekilde atanir.



Sekil 3.6 Hizli Tekrar Gönderim Algoritmasi

3.2.4. Tikaniklik Kontrolü Algoritmasinin UTCP Protokolüne Eklenmesi

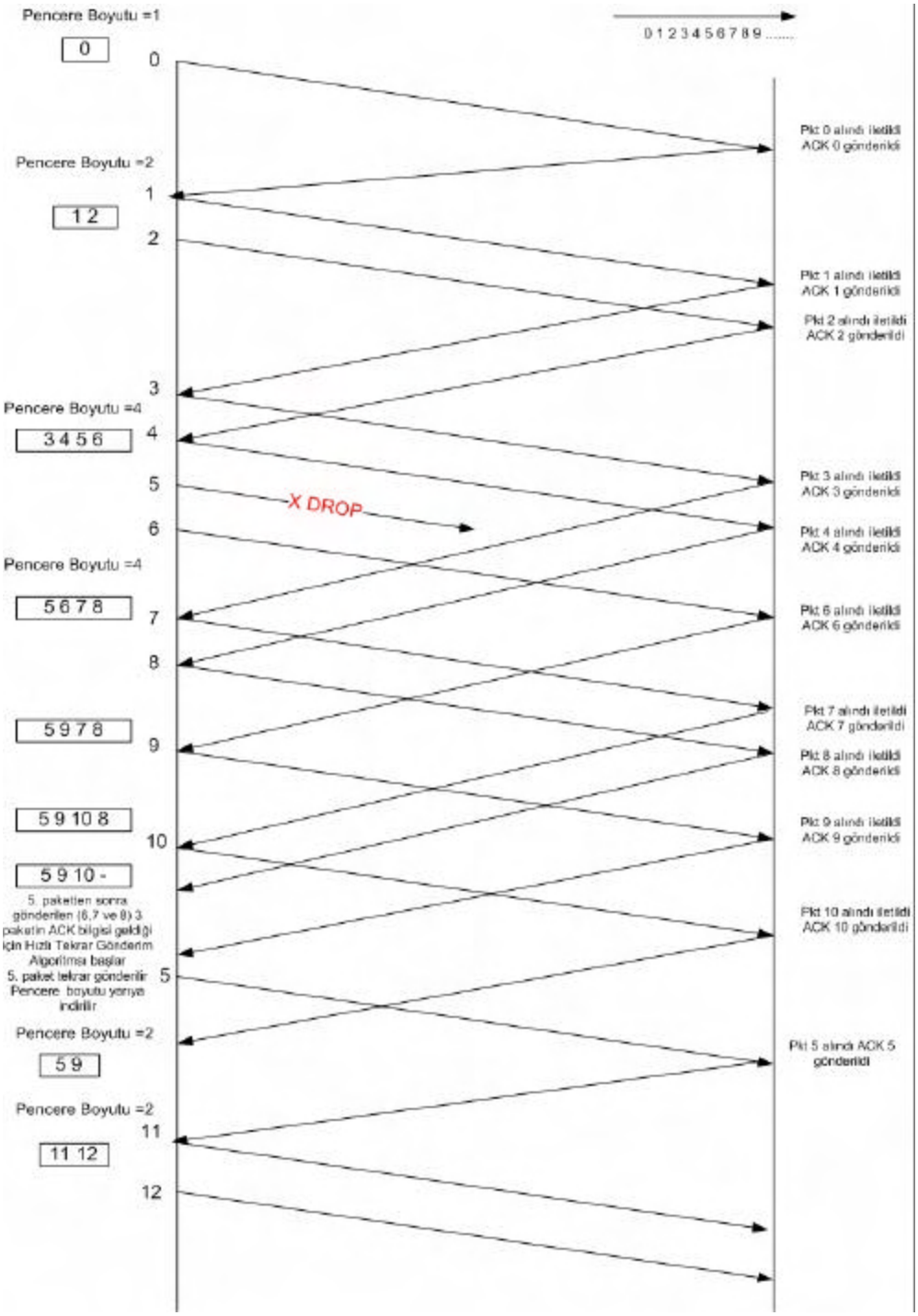
Uygulamanin bu kisminda tikaniklik kontrolü algoritmasi eklenmistir. Tikaniklik kontrolü algoritmasi TCP'nin agdaki veri trafigine bakarak göndericinin gönderebilecegi paket sayisi yani pencere boyutunu sinirlamasini saglamaktaydi. Uygulamamizda pencere içerisinden birbirinden bagimsiz mantiksal kanallar oldugu kabul edilmis ve pencere sayisi kanal sayisina esitlenmisti. Tikaniklik kontrolü algoritmasi ile pencere boyutu dinamik olarak degisecegi için kanal sayisi da buna bagli olarak degisecektir. TCP Newreno' da oldugu gibi zaman asimi oldugunda göndericinin tikaniklik pencere boyutu (CongWnd) bir olarak atanacaktir ve ACK bilgisi gelmemis paketler tekrar gönderilecektir. Zaman asimi süresinde kanal sayisi da pencere boyutuna bagli

olarak bir olarak atanacaktir. Eger bir paketin kaybolmasi durumunda ve kendisinde sonra gelen 3 tane paketin ACK bilgisi gelmesinde hizli tekrar gönderim algoritmasi çalisarak kaybolan paket tekrar gönderilmektedir. Hizli tekrar gönderim algoritmasindan sonra tikaniklik pencere boyutu dolayisiyla kanal sayisi yariya indirilecektir. Bir paket kaybolma olayi oldugunda hizli tekrar algoritmasinin bitimine kadar tikaniklik pencere boyutu dolayisiyla kanal sayisi sabit kalmaktadir.

Sekil 3.7'de uygulamamizin güvenilir ve sirali iletim yaparken nasil bir yöntem izledigi gösterilmistir. TCP baglantisi kuruldugunda pencere boyutu dolayisiyla kanal sayisi 1 olarak ayarlanmistir. Uygulamamizda TCP Newreno'da oldugu gibi ilk paket kaybolana kadar bu degerler üssel olarak artmaktadir. Alinan her bir ACK bilgisi için pencere sayisi artmaktadir. Sekil 3.1'de oldugu gibi sekil 3.7'de yine 5 numarali paketin kayboldugu varsayilarak uygulamamizin TCP Newreno'dan farkli olarak izledigi yöntem gösterilmistir. Sekil 3.1'e bakarsak TCP Newreno'da 5 numarali paket kaybolduktan sonra bu paketin tekrar dogru olarak iletilmesine kadar yeni paket iletimi yapilamamaktaydi. Uygulamamizda sirali bir iletim söz konusu olmadigi için böyle bir bekleme söz konusu olmamaktadir. Sekil 3.7'de görüldügü gibi 6 ve 7 numaralar için ACK bilgisi alindiginda bu paketlerin gönderildigi kanallar üzerinden hiç beklemeden yeni paketler iletilmektedir. Böylelikle baglantidan sürekli olarak veri gönderilerek veri bant genisligi TCP Newreno'ya göre daha verimli kullanilmaktadir.

Uygulamamizda alici tarafta paketlerin sira numarasina bakmaksizin aldigi sira ile üst katmana iletip ACK bilgisini göndermektedir. Sekil 3.7'de 8 numarali paket alindiginda uygulamamizda hizli tekrar gönderim algoritmasi çalismaktadir. Buna pencere sayisi dolayisiyla kanal sayisi yariya indirilerek kaybolan paketler gönderilmektedir. Kaybolan paketlerin tamami dogru olarak iletilene kadar ve hizli tekrar gönderim algoritmasinin basladigi anda kanallarda bulunan bütün paketler için ACK bilgisi gelene kadar yeni paket iletimi yapilmamaktadir. Sekil 3.7'de görüldügü gibi 5, 9 ve 10 numarali paketler için ACK bilgisi alinmadan yeni paket iletimi yapilmamistir. Hizli tekrar algoritmasindan sonra tikaniklik pencere boyutu dogrusal olarak artmaktadir. Slow Start asamasindan farkli olarak alinan her ACK bilgisi için bir artarak degil pencere içerisindeki bütün paketler

için ACK bilgisi alindiginda bir artmaktadir. Bu asamaya da daha önce bahsettigimiz gibi **Congestion Avoidance** denilmektedir.



Sekil 3.7 Tikanik Kontrol Algoritmasi Eklenmis UTCP'nin Güvenilir Veri Iletim Mekanizmasi

3.3. UTCP ile Programlama: Soket API

UTCP protokolü daha öncede söyledigimiz gibi güvenilir ve sirali olmayan veri iletimi yapmaktadir. Paketleri siralama islemi uygulama katmanina birakilmistir. Bu bölümde UTCP için bir dosya transfer uygulamasinin gerçeklestirilmesinden bahsedilecektir. Uygulamada soket uygulama protokol arabirimi (Application Protocok Interface, API) kullanilmistir. Soket API kullanicinin TCP/IP yigitini kullanmasini saglayan bir arabirim saglar. Sekil 3.8'de TCP'nin sirali paket iletimi hizmeti kullanilarak yapilan büyük boyutlu veri transferi uygulamasinin genel bir algoritmasi gösterilmistir. Uygulama, soket API'lerini kullanarak bir TCP soketi yaratir ve sunucuya baglanir. Daha sonra uygulama baglantidan gelen verileri okumaya devam eder ve verileri dosyanin sonuna ekler. Dosyanin transferi tamamlandigi zaman baglanti kapanacaktir. Uygulama TCP'nin sirali paket iletim arabirimini kullandigi için, paket siralama islemi TCP tarafindan yapilmakta uygulama katmani sadece sirali gelen paketleri dosya sonuna eklemektedir.

```
DownloadObjectUsingTCP()
        fd=Gelen verileri kaydetmek için dosyayı aç;
        Sock=TCP soketi yarat;
        Connect(sock, serverAddress); /* sunucuya bağlan */
        While (Soket açık iken)
          read(sock,buffer,size); /*TCP soketten gelen veriyi oku */
          write(fd,buffer,size); /*Dosyanın sonuna veriyi ekle */
        end-while
```

Sekil 3.8 TCP Kullanilarak Yapilan Veri Transferi Uygulamasinin Genel Algoritmasi

UTCP kullanarak büyük veri transferi uygulamalarini gerçeklestirmek için soket API'yi biraz degistirmeyi amaçladik. Sekil 3.9'da UTCP kullanarak büyük veri transferi uygulamasinin genel bir algoritmasi gösterilmistir. Öncelikle uygulama soket API'lerini kullanarak bir TCP soketi yaratir. Fakat sunucuya baglanmadan önce uygulama, setsockopt fonksiyonunu çagirarak UTCP protokol yiginini kullanmak istedigini bildirir. Daha sonra istemci sunucuya baglanir.

```
DownloadObjectUsingUTCP()
        fd=Gelen verileri kaydetmek için dosyayı aç;
        Sock=normal bir TCP soketi yarat;
        setsockopt(sock, UTCP);  /* soketi UTCP soketi yap*/
        Connect(sock, serverAddress);  /* sunucuya bağlan */
        While (Soket açık iken)
           getdatarange(sock, &size, &from, &to);  /* Paket boyutunu
                                                      ve aralığını getir */
           read(sock,buffer,size);   /*UTCP soketten gelen veriyi oku*/
           seek(fd,from);     /* Dosyadaki uygun yeri ara
           write(fd,buffer,size);     /*Veriyi Dosyaya yaz */
        end-while
```

Sekil 3.9 UTCP Kullanilarak Yapilan Veri Transferi Uygulamasinin Genel Algoritmasi

Istemcilerin veri degisimi sirasinda sunucuya UTCP kullanma istegini bildirmelerini saglamak amaciyla bir TCP seçimi alani eklemeyi amaçladik. Eger sunucu, istemcinin veri degisimi için UTCP kullanma istegini kabul ederse, UTCP seçenegini içeren bir cevap gönderir. Aksi halde baglanti varsayilan olarak sirali TCP olacaktir.

UTCP baglantisi kuruldugunda, istemci UTCP gelen paketleri uygulama katmanina aldigi sira ile gönderecektir. Paketleri siralama islemi tamamen uygulama katmani sorumlulugunda olacaktir. Paketleri tekrar siralamak amaciyla Rx kuyrugundaki ilk paketin boyutuyla birlikte paketin iletilen nesne içerisindeki kapsadigi baytlarin araligini döndüren getdatarange ismi verilen yeni bir soket katmani eklemeyi amaçladik. Uygulama katmani bu bilgiyi aldiktan sonra, baglantidan veriyi okuyarak arabellege koyar ve daha sonra dosya içerisindeki uygun yeri arar ve veriyi dosyaya yazar. Bu uygulama katmaninin siralama islemini nasil basit bir sekilde basardigini göstermektedir.

Dosyaya yazma islemi sirasinda bu ek arama isleminin UTCP ile kazanilmis olan sirali olmayan veri iletiminin faydasini azalttigi konusu belki tartisilabilir. Ancak uygulama katmaninin uygun bir kullanici arabellegi kullanarak siralama isleminin basit bir sekilde yapabilecegini savunmaktayiz. Uygulama katmani gelen verileri kullanici arabelleginde uygun yerlere koyacak ve arabellegin bütün verileri geldiginde bütün arabellegi dosyanin sonuna ekleyecektir. Bu islem, büyük uygulama katmani arabellegi kullanildiginda pratik olarak dosya arama islemini ortadan kaldirir. Ayni zamanda simülasyon sonuçlari incelendiginde,

paketlerin herhangi bir sira ile ulasmasina ragmen iletilen nesnenin belli bir araligini kapsayan paketlerin bir birlerine yakin olarak iletildikleri görülmektedir. Böylece dosya arama islemini ortadan kaldirmak için uygulama katmaninin çok büyük bir ara bellek kullanmasina ihtiyaç yoktur.

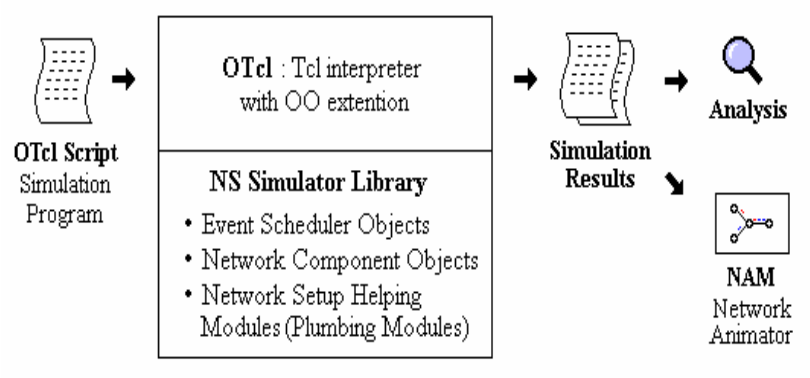
3.4. UTCP Protokolünün Degerlendirilmesi

Bu uygulamada kullanilan bütün simülasyonlar NS-2 ag simülasyon programi üzerinde çalistirilmistir. NS-2'nin seçilmesinin sebebi, NS-2'nin yaygin olarak kullanilan, kodu açik olan, ücretsiz olarak Internet üzerinden indirilebilen bir yazilim olmasindan dolayidir. NS-2 açik kod oldugu için mevcut olan protokoller üzerinde kolaylikla degisiklik yapilabilmekte ve yeni protokoller de tanimlanabilmektedir. Uygulamada yapilan simülasyonlar, Newreno TCP üzerinde yapilan degisikliklerin kazandirdigi performans artisini göstermek üzere seçilmistir.

3.4.1. NS-2 (Network Simulator)

NS-2 Berkeley Üniversitesinde C++ ve OTCL dilleri ile yazilarak gelistirilmis, nesne tabanli ayrik olaya dayali bir ag Simulator programidir. NS yerel ve genis alan aglarinin simülasyonu için oldukça faydali bir programdir. NS programi TCP ve UDP gibi ag protokollerini, trafik kaynagi gibi davranan FTP, TELNET, Web, CBR ve VBR uygulamalarini, yönlendirici (Router) kuyruk yönetim mekanizmasi saglayan Drop Tail, RED ve CBQ uygulamalarini ve Dijkstra gibi yönlendirici algoritmalarini içermektedir [21].

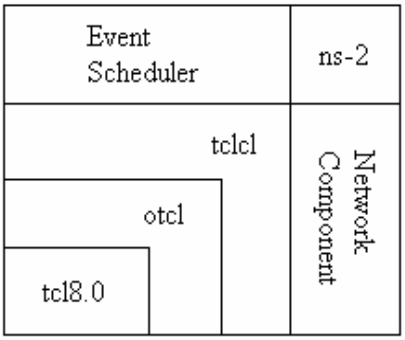
Sekil 3.10'da NS'nin kullanici görünümü basitlestirilmis olarak gösterilmistir. NS, bir simülasyon olayi düzenleyicisi, ag bileseni nesne kütüphanesi ve ag kurulum modül kütüphanelerinden olusmaktadir. Kullanici bir ag simülasyonunu kurmak ve çalistirmak için, bir görev düzenleyicisini baslatan, ag nesnelerini ve kütüphanedeki fonksiyonlari kullanarak ag topolojini kuran ve görev düzenleyicisi boyunca trafik kaynaklarinin paket iletimini ne zaman baslayip bitirecegini söyleyen betigi (script) yazmalidir.



Sekil 3.10 NS'nin Basitlestirilmis Kullanici Görünümü

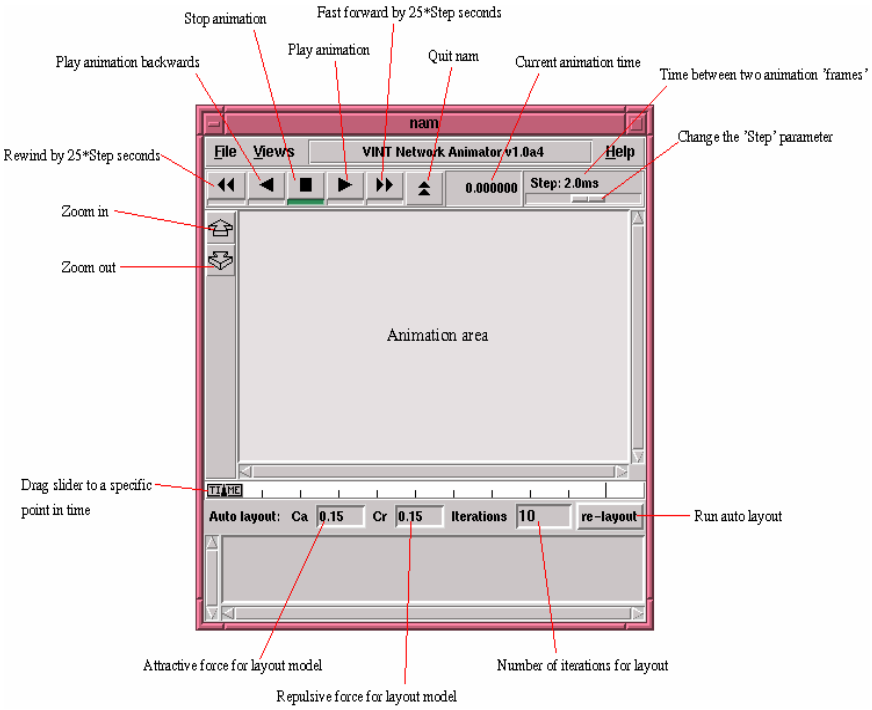
NS programi yazilirken OTCL ve C++ dilleri kullanilmistir. Iki dil kullanilmasinin sebebi NS'nin yapmasi gereken iki farkli isi olmasindandir: Bir yandan programin detayli simülasyonlari, baytlari, paket basliklarini ve büyük veri kümeleri üzerinde çalisan algoritmalari etkili bir sekilde isleyecek bir sistem programlama diline ihtiyaç vardir. Bu tür görevlerde yürütme hizi (run-time) çok önemlidir. Modelin degistirilmesi ve tekrar çalistirilmasi çok önemli degildir. Diger yandan simülasyon konfigürasyonlarini kolayca uygulayabilecek, simülasyonun degistirilmesini ve tekrar çalistirilmasini hizli bir sekilde yapabilecek bir dile ihtiyaç vardir. Bu durumda yürütme hizi önemli degildir, yineleme zamani (modelin degistirilmesi ve tekrar çalistirilmasi) daha önemlidir. NS bu iki ihtiyaci C++ ve OTCL dilleri ile karsilamaktadir. C++ hizli çalisir, fakat detayli protokol uygulamalarinda onu daha uygun hale getirecek degisiklikleri yapmak için yavastir. Otcl daha yavas çalisir, fakat çok çabuk degistirebilerek simülasyon konfigürasyonlari için ideal hale getirilmektedir. C++ oldugu gibi kodun degistirilmesinden sonra tekrar derlenmeye ihtiyaç duymaz. NS, iki dil üzerinde bulunan nesne ve degiskenleri birbirleri ile Tclcl yorumlayicisi (interpreter) yoluyla iliskilendirmektedir.

Sekil 3.11'de NS'nin genel mimarisi gösterilmistir. Buna göre NS Otcl içerisindeki Simulator nesnelerini kullanarak Tcl içerisinde simülasyonlar tasarlar ve çalistirir. Olay is düzenleyicisi ve ag bilesenlerinin bir çogu C++'da gerçeklestirilmistir ve Tclcl ile gerçeklestirilen baglanti ile Otcl'ye erisim saglanmistir.



Sekil 3.11 NS'nin Genel Yapisi

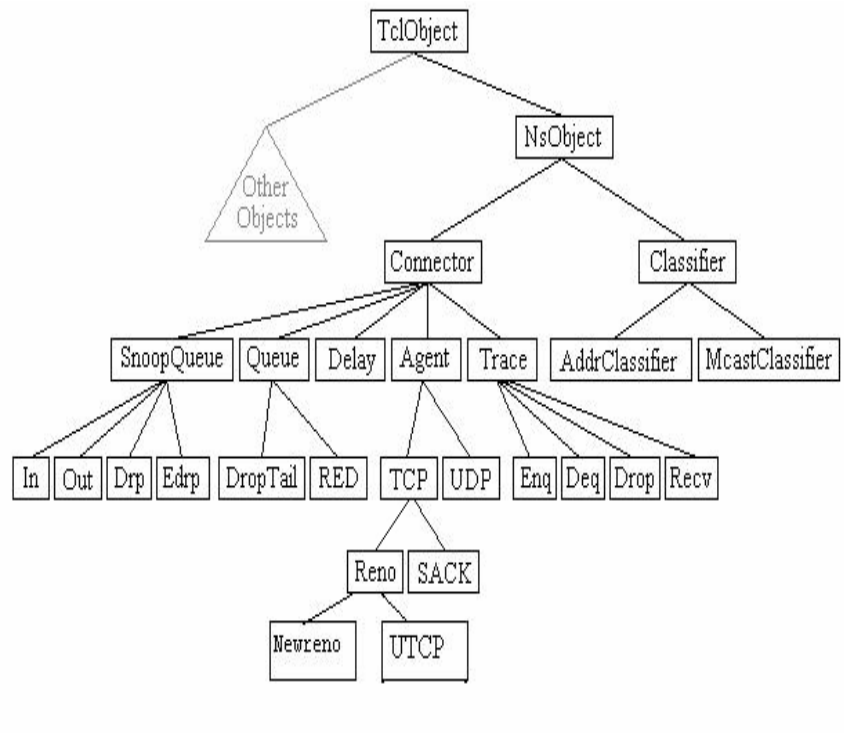
Simülasyon bittigi zaman, NS simülasyonun detaylarini içeren bir veya daha fazla metin tabanli dosya üretir. Bu dosyayi girdi olarak alan NAM (Network Animator) programi simülasyonu grafiksel olarak göstermektedir. Sekil 3.12'de NAM programinin yapisi gösterilmistir



Sekil 3.12 NAM Programinin Genel Yapisi.

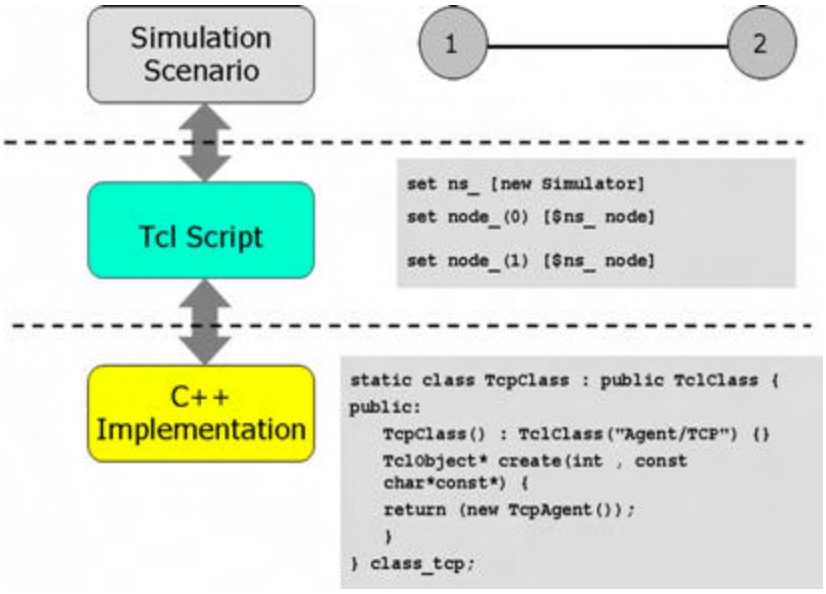
3.4.2. Simülasyonlar

Sekil 3.13 NS'nin sinif hiyerarsisini göstermektedir. Hiyerarsinin kökünde bütün OTcl kütüphane nesnelerini içeren TclObject sinifi vardir. TclObject sinifin bir alt sinifi olan NsObject ag bilesenlerini içermektedir. Ag bilesenleri C++ dilinde yazilarak olusturulmustur. Uygulamada Newreno TCP versiyonu degistirilerek UTCP isimli yeni bir TCP versiyonu olusturulmustur. Yaratmis oldugumuz yeni TCP versiyonunu Newreno versiyonu ile çesitli simülasyon senaryolarinda karsilastirilarak veri iletim hizindaki artis gözlenmistir.



Sekil 3.13 NS Nesne Hiyerarsisi

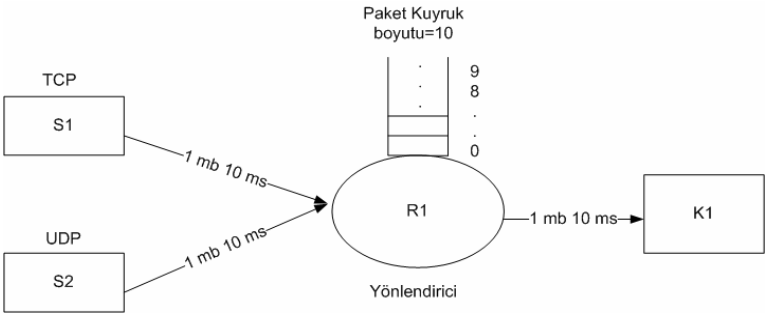
Simülasyon senaryolari Tcl script dilinde yazilarak olusturulmakta Tclcl baglantisi ile C++ ile yazilmis olan ag bilesenlerine ulasilmaktadir. Sekil 3.14'te basit bir simülasyon örnegi gösterilmistir.



Sekil 3.14 Simülasyon Senaryosu Örnegi

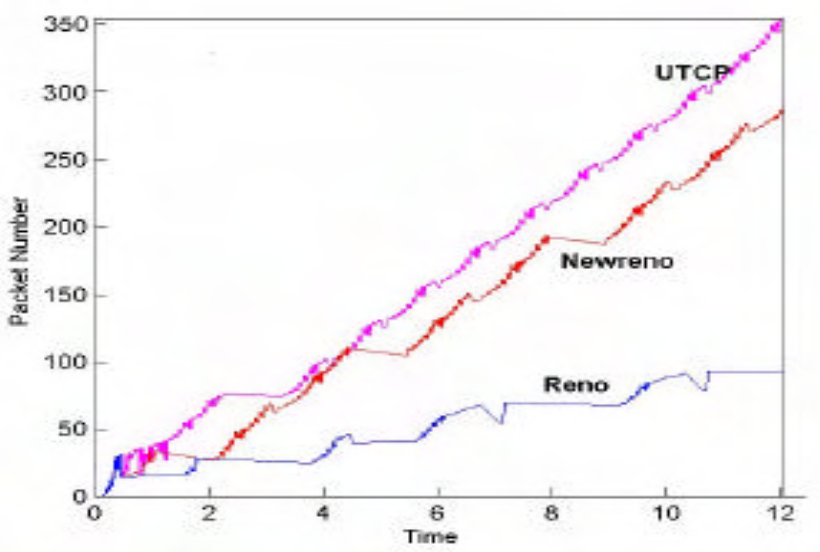
3.4.2.1 Simülasyon Senaryosu 1

Bu bölümde NS simülasyon programi kullanilarak UTCP, Newreno TCP ve Reno TCP'nin veri transferi performanslari karsilastirilmistir. Sekil 3.15'te olusturulmus olan simülasyon topolojisi gösterilmistir. Sekil 3.15'te S1 TCP göndericisini, S2 UDP göndericisini, R1 sinirli sayida ara bellegi olan R1 yönlendiricisini ve K1 alici yi göstermektedir. K1 alicisi aldigi her TCP paket için S1 göndericisine ACK paketleri göndermektedir. Sekil 3.15'te her bir baglantinin veri bant genisligi ve gecikme süreleri gösterilmistir. UDP üzerinde 0,98 Mbps sabit bit orani ile S2'den K1'e dogru trafik üreten CBR uygulamasi çalistirilmaktadir. TCP üzerinde ise S1'den K1'e dogru bütün olarak veri transferi yapan FTP uygulamasi çalistirilmistir. Sekil 3.15'te görüldügü gibi R1 ile K1 arasinda dar bogaz olusturularak paketlerin rastsal olarak düsürülmesi saglanmistir. R1 üzerinde en fazla 10 paket kuyrukta bekletilebilmektedir. Simülasyon topolojisi Ekler bölümünde gösterilen sim1.tcl betigi ile olusturulmustur.



Sekil 3.15 Simülasyon Senaryosu I

Bu simülasyonda Reno TCP, NewRenoTCP ve UTCP protokolleri üzerinde ayri ayri FTP uygulamasi çalistirilarak karsilastirilmalar yapilmistir. Bütün simülasyonlar 0 ile 12 saniye arasinda çalistirilmis ve zamana göre gönderilen paketlerin numaralari sekil 3.16'da gösterilmistir.



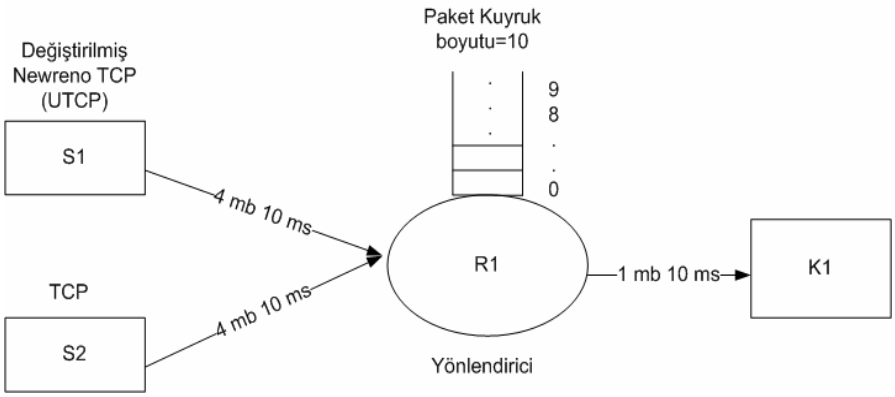
Sekil 3.16 UTCP, Reno Newreno TCP versiyonlarinin Gönderdikleri Paket Sayilarina Göre Karsilastirilmasi

12 saniye sonunda sekil 3.16'da görüldügü gibi yaklasik olarak Reno 80 paket, Newreno 280 paket, UTCP ise 360 paket göndermistir. UTCP diger TCP versiyonlari ile karsilastirildiginda %25 ile %450 arasinda performans artisi saglamaktadir. UTCP paketleri aldigi sira ile uygulama katmanina iletmesine

ragmen, paketleri siralama islemi uygulama katmanina asiri bir yük getirmeyecektir. Çünkü ayni zamanda aliciya birbirine yakin paketler iletilmektedir. Dolayisiyla bu birbirine yakin paketleri siralamak uygulama katmani için çok basit bir islemdir.

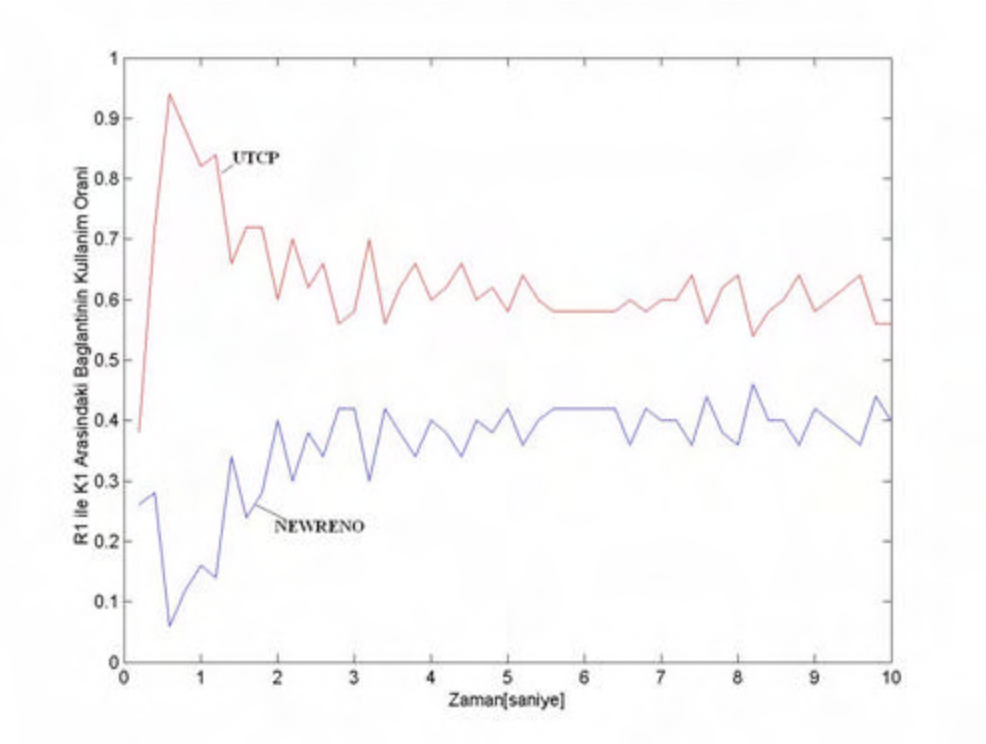
3.4.2.2 Simülasyon Senaryosu 2

Sekil 3.17'de görülen senaryoda önce UTCP ile Newreno TCP, daha sonra UTCP ile Reno TCP beraber çalistirilarak verimlilik ölçüleri karsilastirilmistir. Senaryoda her iki TCP protokolü üzerinde FTP uygulamasi çalistirilarak veri transferi gerçeklestirilmistir. Sekil 3.17'de görüldügü gibi R1 ile K1 arasinda darbogaz olusturularak paketlerin düsmesi saglanmistir. Yönlendirici üzerinde kuyruk boyutu 10 olarak belirlenmistir. Bu sayinin asilmasi durumunda paketler kuyruktan atilmaktadirlar. Simülasyon topolojisi Ekler bölümünde gösterilen sim2.tcl betigi ile olusturulmustur

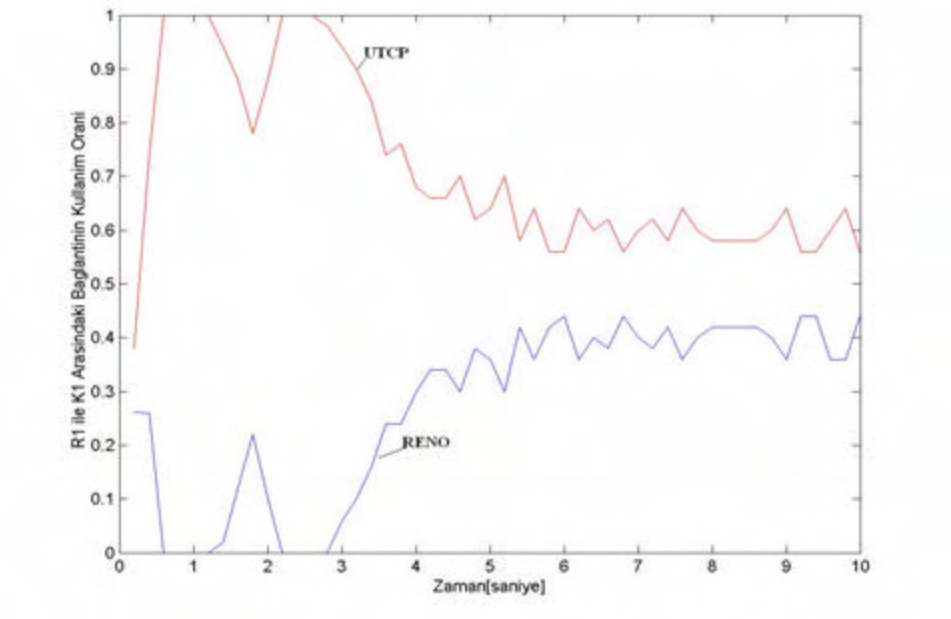


Sekil 3.17 Simülasyon Senaryosu II

Sekil 3.18 ve Sekil 3.19'da görüldügü gibi R1 ile K1 arasindaki baglantinin verimli kullanim oranlari karsilastirildiginda UTCP'nin Reno ve Newreno TCP ye göre üstünlügü açikça görülmektedir



Sekil 3.18 UTCP-Newreno TCP Verimlilik Degerlerinin Karsilastirilmasi



Sekil 3.19 UTCP-Reno TCP Verimlilik Degerlerinin Karsilastirilmasi

4. SONUÇ

Bu çalismada amaç TCP'nin performansinin veri transferi uygulamalari için gelistirilmesidir. Çalismada bunu geçeklestirmek için TCP'nin veri iletim mekanizmasi degistirilmistir. TCP'nin güvenilir ve sirali olan veri iletim hizmeti degistirilerek güvenilir fakat sirali olmayan veri iletim hizmeti gelistirilmis ve elde ettigimiz bu yeni protokol UTCP olarak isimlendirilmistir. UTCP protokolü TCP'nin kati bir sekilde veri iletim yapmasindan dolayi veri transferinde meydana gelen gecikmeleri azaltmayi hedeflemistir. TCP'de bu gecikmelere neden olan "head-of-line blocking" problemi ortadan kaldirilmistir. Verileri siralama islemi uygulama katmanina birakilmistir. Uygulama katmaninda uygun bir arabellek kullanilarak verileri kolaylikla siralamak mümkündür. Veriler siralandiktan sonra uygulama dosyasinin sonuna eklenecektir.

Çalismada güvenilir ve sirali olmayan veri iletimi yapan UTCP gelistirilirken TCP üzerinde yapilan degisikliklerden bahsedilmistir. Buna göre TCP protokolünde veri akis mekanizmasini saglayan kayan pencereler algoritmasi ve tekrar gönderimleri tetikleyen hizli tekrar gönderim algoritmalari degistirilmistir. Çalismamizda ilk önce agin veya alicinin kabul edeceginden fazla verinin gönderilmesini engelleyen tikaniklik kontrolü algoritmasi uygulanmamistir. Pencere boyutu sabit alinarak uygulamamiz denenmistir. Ikinci asamada TCP'nin tikaniklik kontrolü algoritmasi aynen alinarak uygulanmistir.

Gelistirmis oldugumuz UTCP protokolünü TCP'nin Reno ve Newreno versiyonlari ile çesitli ag senaryolari üzerinde karsilastirdik. Simülasyonlar için en yaygin kullanilan NS-2 programi kullanilmistir. NS-2 programinin simülasyonlar sonucu üretmis oldugu çiktilar kullanilarak matlab programi ile grafikler olusturulmustur. Elde ettigimiz simülasyon sonuçlarina baktigimizda UTCP'nin yaklasik olarak Newreno TCP'den %30, Reno TCP'den de %450 daha hizli veri iletimi yaptigi görülmektedir.

UTCP'yi gerçeklestirirken genis bant genisligi olan veya küçük verilerin transfer edildigi uygulamalarda kullanmayi düsünmedik. UTCP paket yogunlugu olan ve büyük hacimli veri transferi yapilan aglarda TCP'ye göre daha verimli çalismaktadir.

KAYNAKLAR

1. FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P. ve LEE, T.B., *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Internet Engineering Task Force (IETF), (1999).
2. POSTEL, J.ve REYNOLDS, J., *File Transfer Protocol (FTP)*, RFC 959, Internet Engineering Task Force (IETF), (1985).
3. POSTEL, J., *Simple Mail Transfer Protocol*, RFC 821, Internet Engineering Task Force (IETF), (1982).
4. POSTEL, J., *Transmission Control Protocol*, RFC 793, Internet Engineering Task Force (IETF), (1981).
5. POSTEL, J., *User Datagram Protocol*, RFC 768, Internet Engineering Task Force (IETF), (1980).
6. Flashget Web Site, http://www.amazesoft.com/index.html
7. STEWARD, R., *Stream Control Transmission Protocol*, RFC 2960, Internet Engineering Task Force (IETF), (2000).
8. HINDEN, R., SAX, J. ve VELTEN, D., *Reliable Data Protocol*, RFC 908, Internet Engineering Task Force (IETF), (1984).
9. CLARK, D., LAMBERT, M. ve ZHANG, L., *NETBLT: A bulk data transfer protocol*, RFC 998, Internet Engineering Task Force (IETF), (1987).
10. FALL, K. ve VARADHAN, K., *The ns Manual (Formerly ns Notes and Documentation)*, (2003).
11. KUROSE, J.F. ve ROSS, K.W., *Computer Networking, Second Edition*, (2003).
12. STEVENS, W.R., *TCP/IP Illustrated, Volume 1*: The Protocols, Reading, Massachusetts: Addison-Wesley, (1994).
13. STEVENS, W.R., *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, RFC 2001, Internet Engineering Task Force (IETF), (1997).
14. JACOBSON, V., *Congestion Avoidance and Control*, Proceedings of ACM SIGCOMM,Stanford, CA, pp. 314-329, (1988).

15. ALLMAN, M., PAXSON, V. ve STEVENS, W., *TCP Congestion Control*, RFC 2581, Internet Engineering Task Force (IETF), (1994).
16. HOE, J.C., *Improving the Start-up Behavior of a Congestion Control Scheme for TCP*. Proceedings of ACM SIGCOMM '96, (1996).
17. MATHIS, M., MAHDAVI, J., FLOYD, S. ve ROMANOW, A., *TCP Selective Acknowledgment Options*, RFC 2018, Internet Engineering Task Force (IETF), (1996).
18. FALL, K. ve FLOYD, S., *Simulation - Based Comparisons of Tahoe, Reno, and SACK TCP*, ACM computer Communication Review, **26(3)**, pp. 5-21, (1996).
19. MATHIS, M. ve MAHDAVI, J., *Forward Acknowledgment: Refining TCP Congestion Control*. Proceedings of ACM SIGCOMM '96, (1996).
20. PARTRIDGE, C., *Implementing the Reliable Data Protocol (RDP)*, USENIX Conf., Phoenix, Az., pp. 367-380, (1987).
21. Ns by example http://nile.wpi.edu/NS/index.htm

EKLER

Sim1.tcl

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        close $tf
        #Execute NAM on the trace file
        #exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

```
#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP/UTCP ]
$tcp set class_ 2
$tcp set packetSize_ 500
$ns attach-agent $n0 $tcp
set sink [new Agent/UTCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 500
$cbr set rate_ 0.99mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.0 "$cbr start"
$ns at 0.0 "$ftp start"
$ns at 12.0 "$ftp stop"
$ns at 12.0 "$cbr stop"

#Call the finish procedure after 12 seconds of simulation time
$ns at 12.0 "finish"

#Print CBR packet size and interval
```

```
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

Sim2.tcl

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open hw3_task2.nam w]
$ns namtrace-all $nf

#open the measurement output files
set f(2) [open hw3_task2_gpout2.tr w]



#necessary to remember the old bandwidth
set oldbw0 0
set oldbw00 0
#Define different colors for nam data flows
$ns color 1 Blue
$ns color 2 Red


#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
            #Close the trace file
        close $nf
            #Close the measurement files
        global f
              close $f(1)
              close $f(2)
#         close $f(3)
            #Execute nam on the trace file
        exec nam hw3_task2.nam &
        exec gnuplot hw3_task2.dem &
        exit 0
}
```

```
proc record1 {} {
       global f qmon fmon oldbw0 flowmon tcp
            #Get an instance of the simulator
              set ns [Simulator instance]
            #Set the time after which the procedure should be called again
       set time 0.2
            #Get the current time
       set now [$ns now]
      #How many bytes have been received by the traffic sinks?
       set parr [$flowmon set parrivals_]
       set pdep [$flowmon set pdepartures_]
       set pdro [$flowmon set pdrops_]
      set fcl [$flowmon classifier]
       set fids { 1 2 3 }
       foreach i $fids {
            set flow($i) [$fcl lookup auto 0 0 $i]
             if { $flow($i) != "" } {
                        #Calculate the throughput and write it to the files
                  puts $f($i) "$now \t [expr [$flow($i) set
bdepartures_]*8/$time/1000000] \t [expr [$flowmon set
bdepartures_]*8/$time/1000000] \t [expr ($parr - $pdep - $pdro)] \t [$tcp($i) set
cwnd_]"
                   #remember the old value
                  $flow($i) set barrivals_ 0
                  $flow($i) set bdepartures_ 0
                  $flow($i) set bdrops_ 0
                         }
       }
            #Reset the bytes_ values on the traffic sinks
       $flowmon set barrivals_ 0
       $flowmon set bdepartures_ 0
            #Re-schedule the procedure
       $ns at [expr $now+$time] "record1"
}


#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create links between the nodes
$ns duplex-link $n0 $n2 4Mb 10ms DropTail
$ns duplex-link $n1 $n2 4Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
$ns queue-limit $n2 $n3 10
```

```
#Flow Monitor
set linkn2n3 [$ns link $n2 $n3]
set flowmon [$ns makeflowmon Fid]
$ns attach-fmon $linkn2n3 $flowmon
#Monitor the queue for the link between node 2 and node 3
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setting for nam
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right


#4Mbps TCP traffic source
#Create a TCP agent and attach it to node n0
set tcp(1) [new Agent/TCP/UTCP]
$ns attach-agent $n0 $tcp(1)
$tcp(1) set fid_ 1
$tcp(1) set class_ 1
#$tcp(1) set rate_ 4Mb
# window_ * (packetsize_ + 40) / RTT
$tcp(1) set window_ 40
$tcp(1) set packetSize_ 460
#$tcp(1) set overhead_ 0.001
#Create a TCP sink agent and attach it to node n3
set sink [new Agent/UTCPSink]
$ns attach-agent $n3 $sink
#Connect both agents
$ns connect $tcp(1) $sink

# create an FTP source "application";
set ftp(1) [new Application/FTP]
#$ftp set maxpkts_ 1000
$ftp(1) attach-agent $tcp(1)


#1Mbps TCP traffic source (one)
#Create a TCP agent and attach it to node n0
set tcp(2) [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp(2)
$tcp(2) set fid_ 2
$tcp(2) set class_ 2
#$tcp1 set rate_ 4Mb
# window_ * (packetsize_ + 40) / RTT
$tcp(2) set window_ 40
$tcp(2) set packetSize_ 460
#$tcp(2) set overhead_ 0.001
```

```
#Create a TCP sink agent and attach it to node n3
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
#Connect both agents
$ns connect $tcp(2) $sink1

# create an FTP source "application";
set ftp(2) [new Application/FTP]
#$ftp(2) set maxpkts_ 1000
$ftp(2) attach-agent $tcp(2)


#Schedule events for all the flows
$ns at 0.0 "record1"
$ns at 0.0 "$ftp(1) start"
$ns at 10.0 "$ftp(1) stop"
$ns at 0.0 "$ftp(2) start"
$ns at 10.0 "$ftp(2) stop"

#Call the finish procedure after 10 seconds of simulation time
$ns at 10.0 "finish"

#Run the simulation
$ns run
```