YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

FBIO: FUZZY TABANLI BIO AKTİF KUYRUK YÖNETİM MEKANİZMASI

Bilgisayar Mühendisi Faris ŞEN

FBE Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Yrd. Doç. Dr. A. Gökhan YAVUZ (Y.T.Ü.)

İSTANBUL, 2008

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

ABR	Avaible Bit Rate
AF	Assured Forwarding
AQM	Active Queue Management
ATM	Asynchronous Transfer Mode
BB	Bandwidth Broker
BIO	BLUE with IN and OUT
CAC	Connection Admission Control
CBQ	Class Based Queueing
CBR	Constant Bit Rate
CE	Congestion Experienced
COA	Center of Area
COG	Center of Gravity
cwnd	Congestion Window
DiffServ	Differentiated Services
DT	Drop Tail
ECN	Explicit Congestion Notification
ECT	ECN Capable Transport
EF	Expedited Forwarding
FIFO	First In First Out
FLC	Fuzzy Logic Control
FTP	File Transfer Protocol
GFR	Guaranteed Frame Rate
IEEE	Institute of Electrical and Electronics Engineers
IETF	nternet Engineering Task Force
IntServ	Integrated Services
NAM	Network Animator
nrt_VBR	Non real-time Variable Bit Rate
NS	Network Simulator
OTCL	Object TCL
PHB	Per-Hop Behaviour
QoS	Quality of Service
RED	Random Early Drop
RIO	RED with IN and OUT

RSVP	Resource Reservation Protocol
rt_VBR	Real-time Variable Bit Rate
RTT	Round Trip Time
rwnd	Receiver Window
ssthresh	Slow Start Threshold
TCL	Tool Command Language
TCP	Transmission Control Protocol
TOS	Type of Service
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
UPC	Usage Parameter Control
VC	Virtual Circuit
WFQ	Weighted Fair Queuing

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Hayatımın her anında yanımda olan ve her konuda destek veren aileme; çalışmalarım süresince gösterdiği sabır, anlayış ve desteği için nişanlıma; yüksek lisans eğitimi ve tez geliştirme süresince yardımlarını, bilgisini ve zamanını paylaşan danışman hocam Yrd. Doç. Dr. A. Gökhan YAVUZ'a ve bölüm başkanımız Prof. Oya KALIPSIZ'a teşekkürü borç bilirim.

ÖZET

Son yıllarda Internet kullanıcıları ve buna paralel olarak da Internet uygulamaları hızla artmaktadır. Artan bu kullanımla birlikte trafik yoğunluğu da gün geçtikçe büyümektedir. Teknolojideki hızlı gelişme ve yeni uygulamalar farklı ihtiyaçları da beraberinde getirmektedir. Ağ verimliliğini sağlamak ve gelişen servislere uygun, kaliteli hizmet sunmak temel sorunlardan biri haline gelmiştir. Bu doğrultuda ihtiyaçları karşılamak için yeni tıkanıklık kontrolü ve kuyruk yönetim mekanizmaları tasarlama gerekliliği ortaya çıkmaktadır.

Ağdaki tıkanıklığı önlemek için uzun süredir tıkanıklık kontrol mekanizmaları kullanılmaktadır. Bu mekanizmalardan en yaygın olanı aktif kuyruk yönetim (AQM) yapılarıdır. Temelde bu yapı, paketlerin gerektiğinde düşürülüp düşürülmeyeceğine karar vererek kuyruğun boyutunu yönetmektedir. Günümüzde çeşitli AQM algoritmaları mevcuttur. Fakat bunların çoğunluğu yüksek trafik ve farklı uygulamalar altında istenilen performansı verememektedirler. Bunun yanında zor düzenlenebilen parametrelerin, gecikme sürelerinin ve yaşanan paket kayıplarının iyileştirilmesi gerekmektedir.

Bu tez çalışmasında öncelikle mevcut tıkanıklık kontrol yapıları ve AQM algoritmaları incelenerek; temel sorunları ve bunların çözümleri üzerinde durulmuştur. Bunların ışığında kuyruk yapılarının tasarımında yeni eğilim olan alternatif tekniklerin kullanımı üzerinde yoğunlaşılmıştır. Bulanık mantık (fuzzy lojik) kontrolleri (FLC) problemlerin üstesinden gelebilmedeki başarısı ve tıkanıklık kontrolü modellemesindeki esnek yapısıyla bu tekniklerin en önemli olanlarındandır. Geliştirdiğimiz tasarım, bulanık mantık kontrollerini kullanarak AQM algoritmalarından yaygın olarak bilinen BIO (BLUE IN & OUT) yapısındaki problemleri iyileştirmeye yöneliktir. Bunun sonucunda da geliştirilen tasarımın başarımı ölçümlenerek geleneksel tasarımla karşılaştırılmıştır.

Tasarlanan aktif kuyruk yönetim yapısı NS-2 network simülatör üzerinde gerçeklenmiştir. Öncelikle bulanık mantık yapısına girecek (paket kaybı, kuyruk boyutu) ve çıkacak değişkenler (paket düşürme olasılığı) seçilmiş, bu değişkenlere ait üyelik fonksiyonları tasarlanmıştır. Sonrasında da IF-THEN kuralları oluşturularak giriş ve çıkış değişkenleriyle ilişkilendirilmiştir. Tasarımda Single-Tone fuzzifier ve Center Of Gravity (COG) defuzzifier metotları kullanılmıştır. Benzetimlerle yapılan ölçümlemeler sonucunda bulanık mantık tasarımının geleneksel tasarıma göre daha az paket kaybı, daha az gecikme, daha az kuyruk boyutu ve daha fazla verim sağladığı görülmüştür.

Anahtar Kelimeler: QoS, Kaliteli Hizmet, Fuzzy Lojik Kontrol, Bulanık Mantık Köntrolü, Kuyruk Yönetimi, Tıkanıklık Kontrolü, AQM, NS-2, BIO (BLUE IN & OUT)

ABSTRACT

During past few years, Internet usage and number of applications are growing rapidly. Due to this usage growth, also amount of traffic have increased. There are different requirements along with technological improvements and new applications. To provide network throughput and quality of service to new applications has become one of the most important problems. It is necessary to design effective congestion control and queue management due to new requirements.

For along time, congestion control mechanisms are using to prevent congestion in the networks. Active queue management (AQM) algortihms are the most common methods for congestion control. These work basicly on managing queue size by choosing which packet to drop or not. Nowadays, many AQM algorithms exist. According to this, most of these algortihms do not work well under high traffic and different types of applications. Besides, hard to configure parameters, long delay times and most packet drops are issues to have improvements.

In our thesis, existing congestion control mechanisms and active queue management algorithms are observed firstly and basic problems and solutions are investigated. This led to a new trend in using alternative techniques. Fuzzy Logic Controllers, which have ability to cope with the aforementioned problems and provide more flexibility in modeling congestion controllers as well, are most important ones. In our approach, we implement FLC into mostly knowing AQM algorithm named BIO to improve its performance. After this, our design tested and compered with traditional design.

Our active queue management design are simulated on ns-2 network simulator tool. Firstly, we choose variables that input and output for fuzzy logic controller and design membership functions of these variables. After that, IF-THEN fuzzy rules are built and associated with variables. Single-tone fuzzifier and Center Of Gravity (COG) defuzzifier methods were used in the design. As a result of measurement by simulations, it was shown that the proposed fuzzy design achieves less packet loss, short delay times, short queue sizes and high throughput compared with traditional one.

Keywords : QoS, Quality Of Service, Fuzzy Logic Control, Queue Management, Congestion Control, AQM, NS-2, BIO (BLUE IN & OUT)

1. GİRİŞ

Günümüzde Internet hızlı bir şekilde büyürken, üzerindeki uygulamaların da sayısı ve çeşitliliği artmaktadır. Bu artış beraberinde farklı topolojiler, protokoller ve trafik yapıları ortaya çıkarmaktadır. Bunların her biri farklı ihtiyaçlar ve QoS beklentisi doğurmaktadır. Mevcut Internet mimarisi aynı anda tüm bu uygulamalara istedikleri kaliteli servisi sunacak şekilde tasarlanmamıştır. Bu nedenle tıkanıklıklara karşı trafiği kontrol eden, gecikmeyi ve paket kaybını azaltan, bunun yanında kaliteli hizmeti sağlayacak yapıların tasarımı önem kazanmaktadır.

Tıkanıklık kontrolü çözümlerinde, tasarlanan yapılar arasından en yaygın olanı aktif kuyruk yöntemleridir (AQM). Bu yapılar kuyruğun durumunu göre karar verip gelen paketlerin düşürülmesini veya iletilmesini sağlar. Böylece kuyruk boyutunu yönetirler. Bu yapıların tasarımlarından kaynaklanan, yüksek trafik ve farklı uygulamalar altında istenilen performansı gösterememek gibi sorunlar bulunmaktadır. Aynı zamanda zor düzenlenebilen parametrelerin, gecikme sürelerinin ve yaşanan paket kayıplarının iyileştirilmesi bu alanda geliştirme yapılan konulardır.

Son dönemlerde fuzzy lojik kontrolleri birçok alanda kullanılmaktadır. Özellikle bilgi teknolojilerinde sınıflandırma amaçlı birçok uygulamada başarılı sonuçlar vermektedir. Aktif kuyruk yönetim (AQM) mekanizmalarının tasarımında kullanılan alternatif tekniklerde yeni eğilim fuzzy lojik kontrolleridir. Fuzzy lojik tasarımı esnek yapısı ve sorunlara sağladığı çözümlerle AQM tasarımında başarılı sonuçlar vermektedir.

Bu çalışmada amaçlanan fuzzy lojik yapısını kullanarak yaygın olarak bilinen BIO (Blue With IN & OUT) AQM yapısının performansını arttırmak ve benzer yöntemlerle karşılaştırmaktır. Bu nedenle öncelikle trafik kontrolü ve hizmet kalitesi (QoS) yapıları 2. bölümde anlatılmaktadır. Mevcut AQM yapıları ve tıkanıklık kontrolüyle ilgili bilgiler 3. bölümde verilmiştir. 4. bölümde fuzzy lojik yapısının tasarım aşaması, 5. bölümde de uygulamanın gerçeklenmesi ve performans ölçümleri yer almaktadır. Son olarak da 6. bölümde sonuçlar karşılaştırılarak veriler yorumlanmış ve sistemin başarımı özetlenmiştir.

2. IP AĞLARINDA TRAFİK KONTROLÜ VE HİZMET KALİTESİ

Internet üzerinde artan uygulamalarla beraber bu uygulamaların beklentileri de farklılaşmaktadır. Ağ üzerindeki trafik, kullanıcıların ve ağ üzerindeki uygulamaların çoğalmasıyla birlikte artmaktadır. Uygulamaların ihtiyaçlarını karşılamak için sadece bandgenişliğini arttırmak yeterli olmamaktadır. Yeni uygulamalar yeni hizmet tiplerine ihtiyaç duymaktadır. Buna karşın günümüzdeki internet mimarisi bu isteklere cevap verememektedir.

Bu sebeple uygulamaların QoS isteklerini karşılayacak yapılara ihtiyaç duyulmaktadır.

2.1 Hizmet Kalitesi (QoS)

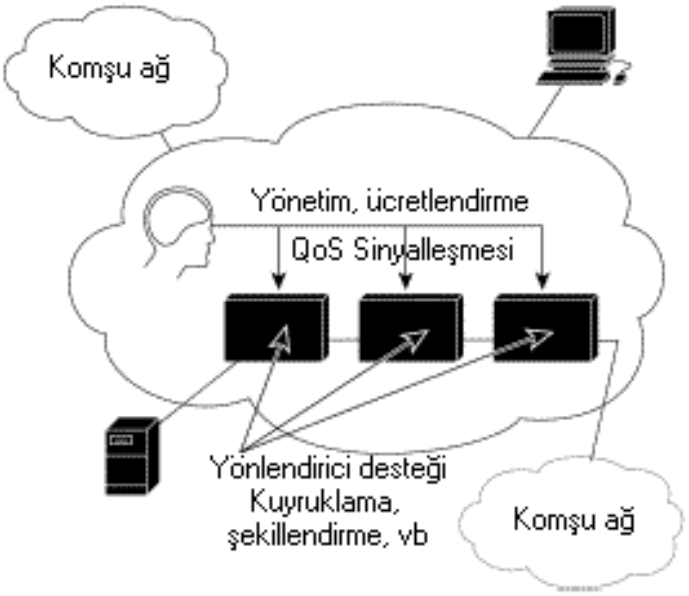
Kaliteli hizmet, uygulamaların ihtiyaçlarına özgü hizmet alabilmelerini sağlayabilmek amacıyla trafik ve hizmet tiplerinin ayırt edilebilmesidir. Bunun sağlanması için ağdaki tüm düğümlerin ve tüm iletişim katmanlarının destek vermesi gereklidir. QoS bandgenişliği sağlamaktan ziyade bu kapasitenin ihtiyaçlara göre uygun kullanımını amaçlar. QoS; farklı uygulamalara, farklı kullanıcılara veya farklı veri akışlarına farklı öncelikler vererek belli bir performansı veri akışlarına garanti edebilmektedir.

Hizmetin kalitesi bazı önemli parametrelere göre belirlenir. Bunlar gecikme (delay), gecikme değişimi (jitter), bandgenişliği, paket kaybı (packet dropping) ve güvenilirliktir.

Gecikme ve gecikme değişiminin artması, zaman kısıdı bulunan özellikle gerçek zamanlı uygulamalarda performansı etkiler. Bandgenişliği farklı uygulamaların kaynak paylaşımı sebebiyle sağlıklı çalışabilmesi için önemlidir. Güvenilirlikte ise hatalı bildirimler, TCP tarafından tekrar gönderilirken, UDP kullanan multimedya uygulamalarında verinin hatalı gönderimi anlamına gelmektedir. Hata oranının olabildiğince düşük tutulması gereklidir (Okuroğlu B. ve Oktuğ S. ,2003).

2.2 QoS Mimarisi

QoS mimarisi aşağıdaki şekilde de görüleceği gibi 3 ana bölümden oluşmaktadır. Bunlardan ilki yönlendiricilerin (router) farklı tipte hizmetler verebilmesidir. İkincisi uçtan-uca (end-to-end) QoS desteği verebilmek için geliştirilen sinyalleşme teknikleridir. Son olarak da bu trafiği kontrol etmek ve yönetmek için tasarlanan yapılardır. Genel QoS mimarisi Şekil 2.1'de görülmektedir.



Şekil 2.1 Genel QoS Mimarisi [1]

2.3 Uygulamaların İhtiyaçları

Günümüzdeki uygulamalar amaçları doğrultusunda farklı ihtiyaçlara gereksinim duyarlar. Örneğin gerçek zamanlı uygulamalar verinin düzenli ve doğru şekilde aktarımını talep ederler. Bu uygulamalara en güzel örnek eşzamanlı multimedya iletişimidir, yani görüntü ve sesin iki uç arasında eş zamanlı olarak gösterimi yapılan uygulamalardır. Bu iletişim sırasındaki gecikmeler o verinin kullanılamaz hale gelmesine sebep olabilmektedir. Gecikmelerin sabit olması durumunda bu gecikmeye göre gösterim yapılabilmekte ve bu süre zarfı dışında gelen paketler işlem görememektedirler.

Bu tarz uygulamalarda daha önce de bahsedildiği gibi gecikme ve doğruluk hizmet kalitesini belirlemektedir. Gerçek zamanlı uygulamalar bu parametrelere olan hassasiyetlerine göre ikiye ayrılmaktadırlar. Toleranssız uygulamalar için bu hizmetler çok önemli iken toleranslı olanlar için bir miktar gecikme ve hata göz ardı edilebilirdir.

Belirtilen ihtiyaçların yanında aynı hattın farklı tipteki uygulamalar için de kullanılması çoğunlukla gerekmektedir. Ayrıca hattın aynı bağlantı üzerinde farklı protokollerce belirli bir oranla paylaştırılması istenebilir. Bunun yanında aynı protokol içindeki farklı uygulamalara farklı kaynaklar ayrılması istenebilir (Okuroğlu B. ve Oktuğ S. ,2003).

2.4 QoS Uygulama Modelleri

Kaliteli hizmet sunmanın birçok yöntemi vardır. Temelde hizmet kalitesi modeli olarak best-effort, intserv ve diffserv modellerini inceleyeceğiz.

2.4.1 Best-Effort

Mevcut internet yapısı hizmet kalitesi bakımından best – effort modeliyle çalışmaktadır. Bu yapıda verinin iletimi ile ilgili herhangi bir garanti verilmemektedir. Mümkün olan kapasitede ve gecikmeyle o anki trafikte veri hedefe ulaştırılmaya çalışılır. Internet'in trafiği uç noktalarda kontrol etme düşüncesi sebebiyle bu şekilde çalışmaktadır. Burada temel düşünce veri transferinin protokoller tarafından ağ üzerinde yönetmesinden ziyade uç noktalarda yönetmesine dayanır. Ağ, yavaş olup sıkışıklığa karşı hassas olmasa da uç noktalardaki yapılar bu dezavantajları telafi edebilirler (Zhi Li, 2004).

2.4.2 ATM (Asynchronous Transfer Mode)

ATM teknolojisi QoS ihtiyaçları düşünülerek dizayn edilmiş ilk ağ teknolojilerindendir. ATM çoğunlukla yüksek hızlı omurga ağlarda kullanılmaktadır. Daha da önemlisi, sonrasında tasarlanan QoS mimarileri için önemli bir etki yaratmıştır.

ATM, evrimleşmiş anahtarlama tekniğine, trafik kontrol mekanizmalarına, sanal devre (Virtual Circuit, VC) oluşturma yeteneğine ve bunlar için çeşitli QoS seçeneklerine sahiptir. Ancak temelde kullanım amacı hızlı anahtarlama yeteneğinden faydalanılmak istenmesidir.

ATM teknolojisinin temel bazı özellikleri bulunmaktadır. Bunlardan ilki bağlantı tabanlı olmasıdır. Verinin taşınmasından önce tüm kaynaklar sinyal yordamları aracılığı ile ağ'a bağlantı isteklerini belirtirler. Bu isteklerde seçilen servis tipi, trafik parametreleri ve beklenen QoS talepleri yer almaktadır. Bağlantı sadece ağ üzerinde uygun şartlar var ise kurulmaktadır. Şartlar her iki taraf için de (hem ağ hem de kullanıcı) kabul edilmelidir. İkinci olarak ATM teknolojisi verileri 53 byte (5 byte başlık ve 48 byte bilgi) sabit büyüklüğünde hücreler halinde iletmektedir. Küçük boyutu sayesinde düşük gecikme sunmakta ve iletimde yardımcı olmaktadır. Bu sayede multimedya ve ses gibi verileri taşıyan real-time uygulamaların basit iletim teknikleriyle taşınarak gecikmenin olabildiğince düşük olması sağlanmaktadır (Zhi Li, 2004).

2.4.2.1 ATM Hizmet Sınıfları

ATM, çoklu servis modeli sayesinde real time uygulamalar ve TCP akışlarının da bulunduğu farklı tipteki trafiklerin eşzamanlı olarak aktarılmasını amaçlamaktadır. ATM forum tarafından tanımlanan başlıca QoS hizmet sınıfları aşağıda belirtilmektedir (Zhi Li, 2004).

- **CBR (Constant Bit Rate):** CBR hizmeti özellikle gerçek zamanlı uygulamalarda sabit veri miktarı ve düşük gecikme ihtiyaçlarına çözüm olmaktadır. Bu hizmet, video konferans ve uzaktan eğitim gibi uygulamalarda kullanılmaktadır.

- **rt-VBR (Real-Time Variable Bit Rate):** Gerçek zamanlı uygulamalarda zamanla değişen veri miktarlarına çözüm olmaktadır. Bir bakıma yoğun trafik oluşturan kaynaklarda kullanılmaktadır. Bu hizmette önemli olan düşük gecikme ve gecikmedeki değişimin azlığıdır.

- **nrt-VBR (Non-real-Time Variable Bit Rate):** Daha çok gerçek zamanlı olmayan uygulamalarda kullanılmaktadır. Beklenen trafik akışlarına göre düşük gecikme ve minimum hücre kaybını sağlamaya çalışır. Bu hizmetin kullanıldığı uygulamalara örnek olarak havayolu rezervasyonu, bankacılık işlemleri ve süreç izleme işlemleri verilebilir.

- **UBR (Unspecified Bit Rate):** UBR, gerçek zamanlı olmayan uygulamalarda gecikmeyi ve bazı kayıpları tolere edebilen TCP benzeri trafiklerde kullanılmaktadır. UBR için belli bir QoS garanti edilmemektedir. Bu nedenle best-effort yapısı gibi düşünülebilir. Veri, resim ve dosya transferleri bu uygulamalara örnek verilebilir.

- **ABR (Available Bit Rate):** Yine gerçek zamanlı olmayan uygulamalarda beklenen en çok ve en az veri hızı miktarlarını göre hizmet vermektedir. Diğerlerinden ayıran özelliği açık bildirimle kaynaklardaki kalan kapasiteyi kontrol ederek adil bir hizmet sunabilmesidir.

- **GFR (Guaranteed Frame Rate):** GFR hizmeti, gerçek zamanlı olmayan uygulamalarda, garanti isteği bulunulduğunda en çok ve en az hız limitleri ile en çok burst (patlama) ve frame (çerçeve) boyutlarını belirtir. Dinamik olarak kullanılabilen ek ağ kapasitesine erişebilmektedir. GFR temel olarak TCP/IP paketlerinin ATM üzerinden taşınması için tasarlanmıştır. Resim, veri ve dosya transferlerini bu uygulamaya örnek verebiliriz.

ATM servis kategorileri ve ilişkilendirildikleri QoS garantileri Çizelge 2.1 de özetlenmiştir.

Çizelge 2.1: ATM servis kategorileri ve QoS garantileri (Zhi Li, 2004)

QoS / Servisler	CBR	rt-VBR	nrt-VBR	ABR	UBR	GFR
Kayıp	Evet	Evet	Evet	Evet/Hayır	Hayır	Evet/Hayır
Gecikme	Evet	Evet	Hayır	Hayır	Hayır	Hayır
Gecikme Varyansı	Evet	Evet	Hayır	Hayır	Hayır	Hayır
Band Genişliği	Evet	Evet	Evet	Evet/Hayır	Hayır	Evet/Hayır

2.4.2.2 ATM Ağlarındaki QoS Yapıları

Her bir servis kategorisi için trafik yönetimi ve kontrolü farklı şekilde yapılandırılmıştır. Bununla beraber uç noktaları ve ağı tıkanıklıktan korumak, aynı zamanda ağ kaynaklarını efektif kullanmak amacıyla tanımlanmış genel özellikler de bulunmaktadır (Zhi Li, 2004).

- **Bağlantı Kabul Kontrolü (CAC - Connection Admission Control):** Bağlantı kabul kontrolü, bağlantının kurulması aşamasında isteğin kabul edilip edilemeyeceğine karar vermek amacıyla yapılan işlemleri içermektedir. Ağ tarafından yüksek trafik yüklerini önleyen ilk adımdır. Bağlantı isteklerinde, seçilen hizmet kategorisi, trafik bilgisi ve bandgenişliği, gecikme ve gecikme değişimi gibi istenen ve bu istek doğrultusunda kabul edilebilir QoS parametreleri belirtilmektedir.
- **Geri Bildirim Kontrolü (Feedback Controls):** Geribildirim kontrolü, ağ ve uç sistemler tarafından ATM üzerinde bulunan trafiğin ayarlanması için yapılan işlemlerdir. Buna örnek olarak ABR servis kategorisindeki ABR akış kontrolü verilebilir.
- **Kullanım Parametre Kontrolü (UPC - Usage Parameter Control):** Kullanım parametresi kontrolü, trafiğin izlenerek gerektiğinde kullanıcılara trafik sözleşmesinin yaptırımlarını uygulatmaktadır.
- **Hücre Kaybı Öncelik Kontrolü (Cell Loss Priority Contol):** ATM başlığındaki cell

loss priority bitini işaretleyerek gerektiğinde, öncelikler doğrultusunda paketlerin düşürülmesini sağlar.

- **Trafik Şekillendirmesi (Traffic Shaping):** Trafik akışlarını düzenleyerek ağdaki kümelenmeyi önlemek ve bağlantının uygunluğunu sağlamak için kullanılmaktadır.
- **Ağ Kaynak Yönetimi (Network Resource Management):** Hücre çizelgeleme ve kaynak tedarik etmek amacıyla yapılan işlemlerden sorumludur.
- **Çerçeve Düşürülmesi (Frame Discard):** Tıkanıklığın oluştuğu ağlarda hücrenin bağlı bulunduğu frame yapısının düşürülmesini sağlamak amacıyla tanımlanmıştır.

ATM'in QoS yapısı çok çeşitli durumları kapsadığından ve bunların için çözüm sağlamayı amaçladığından uygulamada zorluklara neden olabilecek kadar karmaşıktır. Bu nedenle çok yaygın kullanımı olamamış ve QoS yeteneklerinden yeterince yararlanılamamıştır. Sadece belirli yeteneklerine (ABR, UBR) yoğunlaşılmıştır. ABR'in kullanımı karmaşıkken, UBR'in başarımı ise diğer bağlantı tiplerindeki trafikten etkilenmektedir.

Alt seviyelerde ATM kullanılsa da, uçtan uca iletim TCP ile sağlanmaktadır. Uçtan uca tüm yapının ATM olmaması nedeniyle istenilen seviyede QoS hizmetleri tam anlamıyla sağlanamamaktadır (Okuroğlu B. ve Oktuğ S. ,2003).

2.4.3 IntServ (Integrated Services)

Tümleşik hizmetler (Integrated Services), IETF tarafından IP ağlarında uçtan uca QoS sunmak için geliştirilmiştir. Bu çalışma iki temel ihtiyaçtan doğmuştur. Bunlardan ilki belirli paket kayıplarının ve gecikmelerin garanti edilerek gerçek zamanlı uygulamaların desteklenmesidir. Diğer ihtiyaç ise ağ kapasitesinin farklı trafik sınıflarına göre paylaşımının kontrol edilebilmesidir. Bu iki ihtiyacın ışığında IETF tarafından tümleşik hizmet modeli içinde belirtilen, garantili hizmet (quaranteed QoS) ve kontrollü hizmet (controlled-load QoS) isminde iki yeni servis kullanıma sunulmuştur. ATM'e benzer olarak, trafik yönetim ve kontrol parametrelerini taşıyarak rezervasyon yapılmasına olanak sağlayacak sinyalleşme protokolü gereklidir. IntServ yapısında bu mekanizma için Kaynak Rezervasyon Protokolü (Resource Reservation Procokol - RSVP) kullanılmaktadır. RSVP, sadece IntServ ile kullanılmak üzere değil çeşitli QoS protokolleri ile birlikte kullanılmak üzere tasarlanmıştır. Bu nedenle RSVP, aktarılan mesajın içyapısını tanımlamaz. Yani RSVP sadece sinyalleşme protokolüdür ve QoS kontrol bilgisi sadece sinyalin içeriğidir. Bunun yanında IntServ bazı trafik kontrol mekanizmalarını da sunmaktadır (Zhi Li, 2004).

2.4.3.1 IntServ Servis Modelleri

IntServ içinde garantili hizmet ve kontrollü hizmet modelleri tanımlanmıştır.

- **Garantili Hizmet:** Bu servis, rezerve edilmiş kapasite için ve uçtan uca paket gecikmesinde kesin sınırlar isteyen, gecikme bakımından hassas gerçek zamanlı uygulamalar için tasarlanmıştır. Bu yapıda trafik; kova hızı *r*, kova derinliği *b*, minimum veri birimi *m* ve en çok paket büyüklüğü *M* parametreleriyle verilen jetonlu kova modeline (token bucket(r,b)) göre şekil almaktadır. Uçtan uca gecikme sınırı sıvı modeline uygun olarak hesaplanmaktadır. Garantili hizmet ATM'deki rt-VBR yapısının karşılığı olarak düşünülebilir. Farkı garantili hizmetin Internet tarafından desteklenmesidir (Zhi Li, 2004).

- **Kontrollü Hizmet:** Bu servis ise adaptif gerçek zamanlı uygulamaların ihtiyaçları için tasarlanmıştır. Kontrollü hizmet, bir trafik akışına ağın yüklü olmadığı durumlarda alacağı hizmet kalitesini sunar ve bunun için kabul kontrol algoritmalarını kullanır.

2.4.3.2 RSVP

RSVP, ağ üzerinde kaynak rezervasyonu için düğümler tarafından kullanılabilen ve IntServ'de varsayılan sinyalleşme protokolüdür. Bu protokolün başlıca özellikleri aşağıda belirtilmiştir.

- Alıcı tarafında başlatılma (Receiver-initiated). Alıcı akış için kaynak rezervasyonunu başlatır ve devamlılığını sağlar. Bu özellik ayrıca uygun multicast yapıları için tasarlanmıştır.

- Tek yönlülük. RSVP sinyalleri her akışa göre bağımsızdır.

- Soft State. Genel olarak RSVP, veriyi gönderen ve alıcısı arasındaki düğümlere QoS isteklerinin iletilmesini ve bu hizmetin sunulabilmesi için gerekli olan bilgilerin yönlendiricilerde tutulmasını sağlamaktadır. Bu durum bilgileri "Soft State"dir, yani periyodik mesajlarla tazelenmediğinde otomatik olarak yok edilmektedir. RSVP ayrıca dinamik QoS istekleri de sağlamaktadır.

RSVP yapısı çoğu uygulamada temel IP birimini daha iyi desteklemektedir (Zhi Li, 2004).

2.4.3.3 IntServ Trafik Kontrol Mekanizmaları

IP ağlarında uçtan uca QoS desteğini sağlamak için öncelikle IntServ kabul kontrol yapısının

uygulanması gereklidir. Sonrasında da her akış trafik kontrol mekanizmalarıyla denetlenmektedir.

- **Paket Çizelgeleme (Packet Scheduling):** Çeşitli kuyruklar ve çizelgeleme yapıları ile değişik paket akışlarının yönlendirilmesini sağlar. Paketlerin değişik seviyelerde hizmet almasının sağlandığı bu bileşen paketlerin kuyruklandığı çıkış birimlerinde koşmaktadır ve ikinci seviye işlem görmektedir.

- **Paket Sıralama (Packet Queuing):** Gelen paketlerin iletilmesine veya tıkanıklık nedeniyle düşürülmesine karar verir. Düşürmenin yanında tıkanıklık oluştuğunu belirtmek amacıyla ECN alanını işaretleme özelliğine de sahiptir.

- **Paket Sınıflandırma(Packet Classification):** Akışları alacakları hizmet sınıflarına göre ayırır. Akışların sınıflandırılmasında paket başlıklarındaki kaynak ve hedef adresleri ile port numaraları gibi alanların kontrol edilmesi kullanılabilir.

- **Kabul Kontrol (Admission Control):** Yeni bir akışın önceki garantileri etkilemeden ağa kabul edilip edilemeyeceğine karar veren yapıdır. Bu yapı akışın geçeceği her yönlendiricide olmalıdır. Kabul kontrol algoritmalarında geleneksel yaklaşım, hizmet verilen tüm akışlara ilişkin bilgilerin saklanarak en kötü duruma göre yapılan hesaplama sonucunda karar verilmesidir. Diğer bir yaklaşım ise en kötü durumu baz almak yerine akışların kullanım istatistiklerinin temel alınarak hesaplama yapılmasıdır.

Akış bazlı olması nedeniyle IntServ modelinde de ölçeklenebilirlik problemi bulunmaktadır (Zhi Li, 2004).

2.4.4 DiffServ (Differentiated Services - DS)

Farklılaştırılmış hizmetler, ölçeklenebilirlik sorunları nedeniyle tümleşik hizmetlere alternatif olarak ortaya çıkmıştır. DiffServ'un basit yapısı, ölçeklenebilirliği ve sınıf bazlı trafik yönetim mekanizmasıyla modern IP ağlarda IntServ yapısından daha kolay uygulanabilir olmuştur[5].

DiffServ, temel olarak IntServ yapısındaki akış bazlı farklılaştırmadan farklı olarak trafik sınıflandırması prensibine göre işlem görmektedir. Ağ üzerindeki her yönlendiricinin farklılaştırılacak trafiğe göre düzenlenmesi gereklidir. DiffServ mekanizmasında sinyalleşme protokolünün uyumlu hale getirilmesine ihtiyaç yoktur.

2.4.4.1 DiffServ Mimarisi

Günümüzde ağ yapısı genellikle küçük ağların daha büyük ağlardan hizmet alması şeklinde olmaktadır. DiffServ mimarisi, trafiğin uç noktalarda sınıflandırılarak farklı hizmetler alacak şekilde yönlendirildiği bir uygulama modelidir. Bu yapıda ağın merkezindeki yönlendiriciler karmaşık işlem yapmadan sadece IP başlığındaki bilgiye göre hizmet verirler.

DiffServ alanları (domain), sınırları açıkça belirtilmiş, tüm düğümlerin DiffServ desteklediği ve tek elden yönetilen bir yapıdır. Bu alanları DiffServ destekleyen ve desteklemeyen alanlara bağlayan yönlendiricilere sınır yönlendiricileri, diğer iç kısımda kalan yönlendiricilere de iç yönlendiriciler denilmektedir(Okuroğlu B. ve Oktuğ S. ,2003).

Tüm bu yönlendiriciler paketlere farklılaştırılmış hizmet verebilirler. Ancak sınır yönlendiricileri buna ek olarak paket sınıflandırılması, trafik denetleme ve şekillendirilmesi de yapabilmektedirler. Sınırdaki yönlendiricilerden gönderici olan yönlendirici, yapılan anlaşmaya uyum sağlamak için trafik şekillendirmesi (traffic shaping) yaparken, alıcı olan yönlendirici gelen paketlerin yapılan anlaşmaya uyumluluklarını (traffic policing) denetler. Tüm iç ve sınır yönlendiricileri paket sınıflandırması yapmaktadır. Sınır yönlendiricileri paketlerin IP adresi port numarası gibi birçok alanına bakarak sınıflandırma yapar ve sonucunda IP başlığındaki ilgili DiffServ alanını işaretleyerek paketin nasıl bir hizmet alacağını belirtir. İç alanlardaki yönlendiriciler ise sadece bu alana bakarak sınıflandırma yaparlar. Daha az işlem gerektiren bu yöntem yoğun olan merkezi yönlendiricilerde daha iyi performans sağlamaktadır.

2.4.4.2 Per-Hop Behavior

Farklılaştırılmış hizmet düğümlerinin akış gruplarına verdikleri yönlendirme hizmetinin niteliğine düğüm davranışı (Per-Hop Behavior - PHB) denmektedir. Örnek olarak PHB, DiffServ grubunun bandgenişliğinin bir kısmını kullanmasını taahhüt edebilmektedir. Bir PHB'nin uygulanması çeşitli kuyruk ve kapasite yönetim algoritmaları ile yapılmaktadır.

PHB'ler kaynak paylaşımlarındaki öncelikler, gecikme ve kayıp gibi trafiğin karakteristik değerlerine göre tanımlanabilmektedir. Paket başlıklarındaki DiffServ alanına göre paketlerin alacakları PHB belirlenir (Okuroğlu B. ve Oktuğ S. ,2003). Günümüzde kullanımı genel olarak benimsenmiş PHB'ler garanti edilen (Assured Forwarding - AF) PHB ve çabuklaştırılmış (Expedited Forwarding - EF) PHB'dir.

- **Garanti Edilen Düğüm Davranışı (AF PHB) :** AF PHB grubu farklı seviyelerde yönlendirme kaynağı sunabilmektedir. Her birinde 3 farklı öncelik bulunan 4 AF sınıfı mevcuttur. Bu öncelikler tıkanıklık durumunda paketlerin düşürülme önceliğini belirlemektedir. Her AF sınıfı için yönlendiricilerde kaynak ayrılmaktadır. Yönlendiricide kalan fazladan kaynakları da kullanabilirler. Paketlerin alacakları hizmetler de AF sınıfı için ayrılan kaynak, doluluk oranı ve paket düşürülme seviyesine göre belirlenir. AF PHB, patlamalı trafikten kaynaklanan kısa vadeli sıkışıklıklarda paketleri kuyruklarda bekleterek tolare ederken, uzun suren sıkışıklıklarda paketleri düşürmektedir.
- **Çabuklaştırılmış Düğüm Davranışı (EF PHB) :** EF uçtan uca düşük gecikmeli, düşük gecikme değişimli ve garantili bir hizmet sunar. Karşılaşılabilecek kayıp ve gecikmeleri en aza indirebilmek için çok kısa kuyruklar kullanılmalı veya hiç kuyruk kullanılmamalıdır. Bu nedenle EF hizmeti sunmak için çıkış hızı giriş hızından fazla olmalıdır.

2.4.4.3 Bandwidth Broker

Hizmeti alan taraf, sunulan hizmeti kendi içinde paylaştırmada iki yaklaşımı benimseyebilir. Düğümler alacağı hizmeti kendisi belirleyerek DiffServ alanını kendisi doldurabilir veya bandwidth Broker (BB) bandgenişliği simsarı tarafından yapılabilir[5].

Tüm düğümler, ağın mevcut kullanımı ve politikasını bilemediğinden BB kullanımı daha uygundur. BB yönetimi kolaylaştırmakta ve durum bilgilerini her yönlendirici üzerinde tutmak yerine merkezi olarak tutulmasını sağlamaktadır. Bu yapı temelde hizmeti paylaştırarak yönlendiricileri konfigüre eder.

2.4.5 IntServ ve DiffServ Karşılaştırılması

InterServ yapısı, internet yapısını aynı ATM ağlarına benzer şekilde düğümlerde akışların durum bilgilerini tutacak şekilde tasarlanmıştır. Gerçek zamanlı ve farklı tipte uygulama akışlarının normal trafikle aynı anda taşımayı amaçlamaktadır. Bu nedenle daha önce belirtilen değişikliğin kaçınılmaz olduğu savunulmaktadır. Sinyalleşme protokolleri sayesinde her akış ağdan farklı seviyede hizmet isteğinde bulunabilmektedir.

Bunun yanında tutulan bilgilerin yüksek işlem gören merkezi düğümlerde boyutları anormal artabilmekte ve ölçekleme sorunlarına neden olmaktadır. Aynı zamanda her düğümde gelen paketlerin, sınıflandırılması ve profil uygunluğunun denetlenmesi gerekmektedir. Bu da

yönlendiricilerde yüksek işlem gücü gereksinimi doğurmaktadır. Tüm bunlar yönlendiricilerde büyük değişiklikler yapılmasını gerektirmektedir.

DiffServ ise tüm bu gereksinimleri ve karmaşıklığı daha az trafiğin işlendiği ağ sınırlarına iterek daha az çaba sarfedilmesini sağlamaktadır. Tutulması gereken bilgilerin de BB gibi merkezi yerlerde tutulması amaçlanmıştır. QoS imkânları birleştirilmektedir. Bunun yanı sıra düğümlerde durum bilgisi tutulmadığından gecikme gibi parametreler için uçtan uca net bir garanti verilememektedir. Tutulması gereken bu durum bilgileri yönlendiriciler yerine paket başlığındaki bir alanda tutulmaktadır. DiffServ yapısının sunduğu QoS hizmeti tek yönlüdür (Okuroğlu B. ve Oktuğ S. ,2003).

3. TIKANIKLIK DENETİMİ VE KUYRUK YÖNETİMİ

Günümüzdeki Internet yapısı, IP protokolü üzerinde bağlantısız bir şekilde esnek bir yapıya sahiptir. Fakat bu mimarinin artılarının yanında eksikleri de bulunmaktadır. IP ağları mimariden kaynaklanan basitlik nedeniyle yoğun trafik altında beklenen performansı sağlayamamaktadır.

Buna çözüm olarak tıkanıklık denetim yapıları (congestion control) geliştirildi. Bunun yanında tıkanıklık denetimi için ağ sınırlarında yapılanlar yeterli olamamaktadır. Yapılan değişikliklere ek olarak ağ üzerindeki düğümlerde de kuyruk yönetimi yapıları geliştirilmiştir.

3.1 Tıkanıklık

Bilgisayar ağlarında sistemler arasındaki veri trafiği arttıkça ağın performansı düşer. Kaynak üzerindeki talep kapasiteyi aştığında tıkanıklık meydana gelir. Bilgisayar ağındaki tıkanıklığın oluşması için birkaç sebep sayılabilir. Bunlardan biri yönlendiricilerdeki yetersiz hafıza olmasıdır. Hafızanın yetersizliğinden dolayı yönlendirici, gelen veri paketlerinin artması durumunda veri trafiğini karşılayamaz ve tıkanıklık oluşur. Tıkanıklığı etkileyen bir başka etmen de yönlendiricilerin işlemci hızlarıdır. İşlemci hızının yavaş olması yönlendiricilerde kuyruk (queue) oluşmasına ve iletimin de yavaşlamasına sebep olur. Tıkanıklığın diğer bir sebebi de hatların bandgenişliğidir. Veri paketi trafiğinin fazla olduğu hatların bandgenişlikleri tıkanıklığı etkileyen faktörlerden biridir. Toplam iletim gecikmesini arttıran, paketlerin düşürülmesi nedeni ile boş yere kaynak harcanmasına neden olan tıkanıklık, gerekli denetim mekanizmalarının bulunmaması durumunda ağın çalışmasını tamamen engelleyebilir.

3.2 Tıkanıklık Denetimi Yapıları

Dört adet tıkanıklık denetimi algoritması vardır. Bunlar yavaş başlangıç (Slow Start), tıkanıklık önleme (Congestion Avoidance), hızlı tekrar iletim (Fast Retransmit) ve hızlı toparlama (Fast Recovery) isimleri ile tanımlanmıştır. TCP protokolünün bu algoritmalardan daha agresif olmaması gerektiği gibi, TCP göndericisinin, bu algoritmaların izin verdiğinden daha muhafazakâr olması, bazı durumlarda daha faydalı olabilmektedir[5].

TCP göndericisi tarafından ağ ortamına bırakılacak verinin miktarını kontrol etmek için yavaş başlangıç (Slow Start) ve tıkanıklık önleme (Congestion Avoidance) kullanılmak zorundadır. Bu algoritmalar kullanılarak bağlantıya bırakılan trafik kısıtlanabilir. Bu algoritmaları gerçekleştirmek için TCP protokolünün her bağlantı ifadesine iki değişken eklenir.

Bunlardan ilki olan Congestion Window (cwnd) yani tıkanıklık penceresidir, gönderici tarafında belirtilen bir limittir. Bu limit, göndericinin ACK almadan önce ağda ileteceği veri miktarıyla ilgilidir. İkincisi ise alıcı tarafında bir limit olan Advertised Window (rwnd) yani ilan edilmiş pencere olarak bilinir. Aktarımı tamamlanmamış verinin miktarıyla ilgilidir. Cwnd ve rwnd veri iletişimine hükmederler.

Bir başka durum değişkeni ise yavaş başlangıç eşik değeridir (Slow Start Thresold - ssthresh). Yavaş başlangıç veya tıkanıklık önleme algoritmalarının veri iletişimini kontrol etmek amacıyla kullanılmaktadırlar.

3.2.1 Yavaş Başlangıc (Slow Start)

Yavaş başlangıç fazında tıkanıklık penceresi, her bir alındı mesajı (ACK) ulaştığında bir segment artmaktadır. Bu durum tıkanıklık penceresinin exponansiyel yani üstsel büyümesine sebep olur. Yavaş başlangıç yöntemi, yeni bağlantı kurulduğunda ve zaman aşımından dolayı yeniden iletim yapıldığında kullanılır. Congestion window zaman aşımı oluncaya kadar veya eşik değerine (ssthresh) ulaşılana kadar artırılır. Eğer zaman aşımı olursa, ssthresh gönderilen verinin yarısına düşürülür. Tıkanıklık penceresi 1 segment boyuna düşürülür ve yavaş başlangıç fazı tekrar başlar.

3.2.2 Tıkanıklık Önleme (Congestion Avoidance)

Diğer bir algoritma ise tıkanıklığın oluştuğunda alınacak önlem üzerinedir. Tıkanıklık oluştuğunda yol üzerindeki düğümlerde paketler geldikleri hızda ilerleyememekte ve kuyruklarda beklemektedirler. Tıkanıklık önleme fazında, tıkanıklık penceresi her bir round trip time da bir segment artar buda tıkanıklık penceresinin lineer olarak artımı anlamına gelmektedir.

3.2.3 Hızlı Tekrar İletim (Fast Retransmit)

Fast retransmit ve Fast recovery algoritmaları bazı durumlarda iletim sayacı sona ermeden evvel TCP'nin veri kaybı tespit etmesini ve hata kurtarma gerçekleştirmesine izin verir. Bu algoritmalar kısmen erken kayıp tespitinden ve yeniden iletim yapmaktan, kısmen de zaman aşımından sonra iletim hızının azalmamasından TCP performansını artırmaktadır.

Eğer segment düzensiz olarak ulaşırsa, alıcı sıralı olarak ulaşan en son segment için alındı

gönderir. Bu segment daha önce alındısı gönderildiği için bu ikinci alındı duplicate acknowledgement (dupack) olarak anılır. Akabinde 3 duplicate paket daha gelirse, gönderdiği segmentin alıcıya ulaşmadığını düşünüp segmenti yeniden gönderir. Segment karşıya ulaştığında veri transferi kaldığı yerden devam eder, buda fast recovery sayesinde olur. Kaybolan segment tekrar gönderildikten ve alındısı (ACK) alındıktan sonra yavaş başlangıç moduna geri dönmeden veri alışverişinin kalınan yerden devam etmesine olanak sağlayan yapıdır.

3.2.4 Hızlı Toparlama (Fast Recovery)

Fast Recovery algoritmasında ssthresh, cwnd'nin yarısı yapılır. Kaybolan segment yeniden transfer edilir. Cwnd = ssthresh+3 yapılır. Böylece cwnd hedefe ulaşan TCP segmenti sayısı kadar büyümüş olur. Kaynak, aldığı her tekrar eden alındı için cwnd değerini bir büyütür. Tekrar eden alındı alınması durumunda haberleşme kanalından bir segment daha çıktığını bildirir. Eğer tıkanıklık penceresi izin verirse yeni bir segment gönderebilir. Kaynak yeni veriyi onaylayan bir alındı alırsa, bu alındı yeniden transfer edilen segmenti de kapsar. Cwnd değeri ssthresh yapılır. Tıkanıklık penceresinin boyu 1 değil, segmentin kaybolduğu anlaşıldığı andaki değerin yarısı olur. Böylece TCP, debisini paketin kaybolduğu andakinin yarısı yapmış olur.

3.3 Temel Kuyruk Yapıları

Ağlarda yönlendiricilerin trafik akışlarına farklı hizmet vermek için kullandıkları en önemli ve temel yol kuyruk yönetimi ve paket çizelgeleme yapılarıdır. Farklı tipte kuyruklar mevcut olmakla birlikte öngörülen ağ özelliklerine ve istenilen hizmet yapısına göre yönlendiricilerde bu yapılar kullanılmaktadır[4].

3.3.1 FIFO

FIFO en basit yapılı kuyruktur. Paketlerin birbirlerine göre öncelikleri bulunmamaktadır. İlk giren ilk çıkar mantığıyla çalışır. Bu tip kuyruklar tüm paketlere eş hizmet verilen yapılarda kullanılmaktadır. Ağda tıkanıklık yaşanmadığı durumlarda FIFO kuyruk yapısı yönlendiricilerde çok az işlem gücü gerektirdiğinden işlevsel olabilmektedir. Ancak bu haliyle akışlara farklılaştırılmış hizmetler sunmaları mümkün değildir. Bunun yanında uç düğümlerde tıkanıklık konusunda yardımcı olamamaktadır. Basitliği nedeniyle tercih edilse de tıkanıklı desteği sebebiyle ileride açıklanacak aktif kuyruk yönetim mekanizmaları daha çok

kullanılmaktadır.

3.3.2 Priority Queuing

Paketlerin farklı öncelikli kuyruklara yerleştirilerek hizmet verildiği bir yapıdır. Burada öncelikli hizmet alması istenen paketler belirlenerek kuyrukta daha önlere yerleştirilir ve öncelikli paketler bitmeden diğer paketler gönderilmez. Yoğun işlem gücü gereksinimleri sebebiyle pratikte oldukça verimsizdirler. Bunun yanında daha düşük öncelikli paketlerin hizmet alamaması da olasıdır.

3.3.3 CBQ (Class Based Queuing)

Sınıflandırılmış kuyruk yapıları, öncelikli kuyruklara benzemektedirler. Bu yapıda farklı önceliklere farklı çıkış kuyrukları bulunmaktadır. Her defasında kuyruklardan alınabilecek veri miktarı parametrelerle belirlenebilmekte ve kuyruklara belirli ağırlıklara göre hizmet verilebilmektedir. Böylelikle düşük öncelikli akışların hizmet alamaması engellenmiş olmaktadır. Paketlerin hangi sınıfa dahil edileceğini belirlemek için bir dizi kontrol işlemi gerektirdiğinden yönlendiricilerde ek işlem gücü oluşturmaktadır. Bu dezavantajı yaygınlaşmasını engellemektedir[4].

3.3.4 WFQ (Weighted Fair Queuing)

Ağırlıklı adil kuyruklar, daha az veri miktarına sahip akışların daha öncelikli hizmet alabilmesine olanak sağlayan bir yapısı vardır. Bu sayede yüksek veri ileten akışların ağ kaynaklarını tüketerek diğer akışların hizmet almalarını engellemesinin önüne geçilmiş olunur. Ölçeklendirme problemleri bulunmaktadır.

3.3.5 Weighted Round Robin ve Deficit Round Robin

Bu algoritma içerisinde sırasıyla her kuyruktan bizim belirlediğimiz miktardaki farklı sayılarda paketler sırayla gönderilir ve başa döner. Temel yaklaşım paketlerin IP başlığındaki TOS (Type of Service) alanına göre kuyruklara ayırmak ve bu kuyruklara ağırlıklı round robin yapısıyla hizmet vermektir (Okuroğlu B. ve Oktuğ S. ,2003).

Round-Robin algoritmasının değişik bir versiyonu da eksik ağırlıklı round robin algoritmasıdır ve bir akışın paket uzunlukları nedeni ile bir adımda kullanamadığı hizmeti bir sonraki adımda sunmaktadır.

3.4 Kuyruk Yönetim Yapıları (QM)

Ağ yapısında tıkanıklığın oluştuğu düğümler, tıkanıklıkla ilgili en erken bilgiye sahip olabilecek yapılardır. Bu nedenler her yönlendiricide de tıkanıklık denetimi yapısına ihtiyaç bulunmaktadır.

Bu yapılar tıkanıklığı fark ederek paket kayıplarını azaltırlar ve protokollerden bağımsız QoS hizmeti verilebilmektedir. Bu sayede tıkanıklık denetim yapılarına sahip olmayan iletim protokollerinin diğer protokolleri mağdur etmesi engellenmektedir. Buna örnek olarak TCP ve UDP iletim protokolleri verilebilir. Tıkanıklık oluştuğunda TCP hızını keserken, UDP akışlarının hızı kesilmemekte ve hak ettiklerinden fazla iletim yapabilmektedirler.

3.5 Aktif Kuyruk Yönetim Yapıları (AQM)

Genelde aktif kuyruk yapıları, akışları kontrol ederek tıkanıklığı kontrol etmeye çalışırlar. Öncelikle tıkanıklığı ölçümlerler ve bu ölçümlemeye göre hareket ederler. Bu ölçümleme yöntemleri olarak iki farklı yöntem kullanılmaktadır. Bunlardan ilki kuyruk tabanlı olanlar, diğeri ise akış tabanlı olanlardır. Kuyruk tabanlı ölçümlemede esas önemli olan kuyruk boyutudur. Bu yapıda geriye yönelik bilgi tutulması gereklidir ve uyum sağlama aşamasında bir miktar gecikme olabilmektedir. Diğer yandan akış tabanlı yapıda ise ölçümleme akışın geliş hızına bağlıdır ve buna göre karar alır. Önceki bilgiye ihtiyaç duymamakla birlikte kolay uyum sağlayabilmektedir.

Aktif kuyruk yönetim mekanizmalarının sağladıkları faydaları şu şekilde sıralayabiliriz:

1. Yönlendiricilerde düşürülen paket sayısı azalır: Kuyruk boyutunun ortalamasını düşük tutarak doğal yoldan oluşan paket patlamalarını paket düşürmeden geçiştirilebilir. Böylece daha verimli bir iletim gerçekleştirilir.

2. Düşük gecikme sağlanır: Ortalama kuyruk boyutu kısalacağından uçtan uca gecikme süreleri kısalmaktadır.

3. Kilitlenme durumunun önüne geçilmiş olunur: Bazı akışların diğer akışların iletimini engellemesi durumu, kuyrukta devamlı boş yer kalacağından oluşmaz. Böylece adil olmayan hizmet verilmesinin önüne geçilmiş olunur.

3.5.1 Drop-Tail

Geleneksel kuyruk yönetim yapısı olan drop-tail, kuyruk dolduğunda kuyruktan düşürme yapmaktadır. Bu yapıda bir üst sınır belirlenir ve bu limit aşılmadığı sürece iletim devam eder. Fakat kuyruk boyu limit değerine ulaştığında gelen paketler düşürülmeye başlanır. Uzun süre kullanılan bu yapıların iki temel sorunu bulunmaktadır.

1. Kilitlenme: Senkronizasyon ve zamanlama problemleri nedeniyle bazı bağlantıların kuyruğu ele geçirerek, diğer bağlantılara hizmet verilmesinin önüne geçebilmesidir.

2. Kuyruk doluluğu: Drop-Tail, yapısı gereği uzun süre dolu kalabilmekte ve sadece paket düşürme işlemi ile hız kesilebilmektedir. Bu nedenle kuyruk yönetim algoritmalarının temel amacı kuyruk boyutunu kontrol ederek kararlı durumda kuyruğun boyutunun en üst seviyede olmamasını sağlamaktır.

DT yapısında yönetilme yapılmayan bir mekanizma mevcuttur. Fakat ağların ihtiyaçlarının artmasıyla ve akan trafiğin artmasıyla bunların kontrol edilip yönetilme ihtiyacı olmuştur. Bu anlamda geliştirilen ilk aktif kuyruk yönetim algoritması rastlantısal erken düşürme (RED – Random Early Drop) algoritmasıdır.

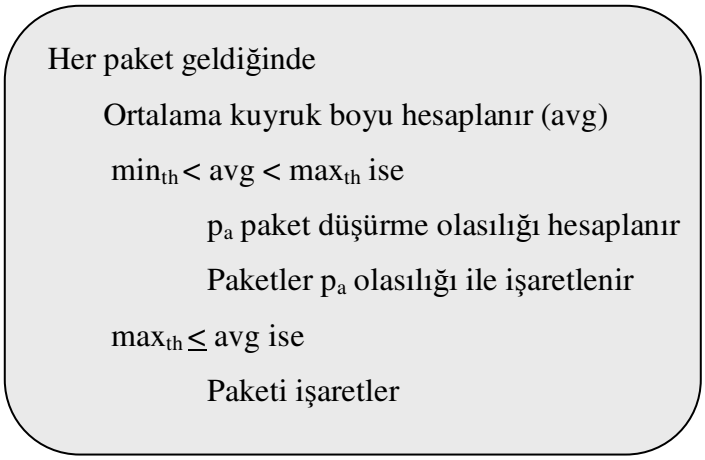
3.5.2 RED ve RIO (RED with IN and OUT)

Bu konuda yapılan çalışmalar sonucunda ilk ortaya atılan AQM algoritmalarından ve en popüler olanı RED (Random Early Detetection) yani rastlantısal erken algılama algoritmasıdır. RED, çıkış kuyruklarının ortalama boyunu denetleyerek, bu boyuta göre rastlantısal olarak tıkanıklıkla ilgili olabilecek değişimleri belirler. Uzun süreli tıkanıklıkta ortalama kuyruk boyutu artarak kaynak düğümlerin hızlarını kesmelerini, kısa süreli tıkanıklıklarda ise değişim çok küçük olacağından iletimde devamlılığı sağlamaktadır. Herhangi bir akışın tıkanıklıktan haberdar olma olasılığı, o akışın toplam trafikteki oranıyla doğru orantılıdır.

Kaynak düğümlerin tıkanıklıktan haberdar olmaları için temel iki yöntem mevcuttur. Bunlardan ilki ve en yaygın olanı paket düşürülmesidir. Diğeri ise ECN (Explicit Congestion Notification) yani açık tıkanıklık bildirini mekanizmasıdır. Kuyruk boyutu belirli limiti aştığında paketler ya düşürülmekte ya da işaretlenmektedirler.

RED kullanılan yönlendiricilerde her çıkış kuyruğu için sürekli ortalama kuyruk boyutu hesaplanmaktadır. Bu ortalama kuyruk boyutu, verilen iki parametre değeri arasında olduğu sürece belirli olasılıkla paketlerin işaretlenmesi ya da düşürülmesi sağlanır. Kuyruk boyu alt

eşik değerinin (min_{th}) altında ise paketler işaretlenmez. Üst limit değerine (max_{th}) ulaştığında ise gelen tüm paketler işaretlenir. RED algoritmasının işleyişi Şekil 3.1'de görülmektedir.

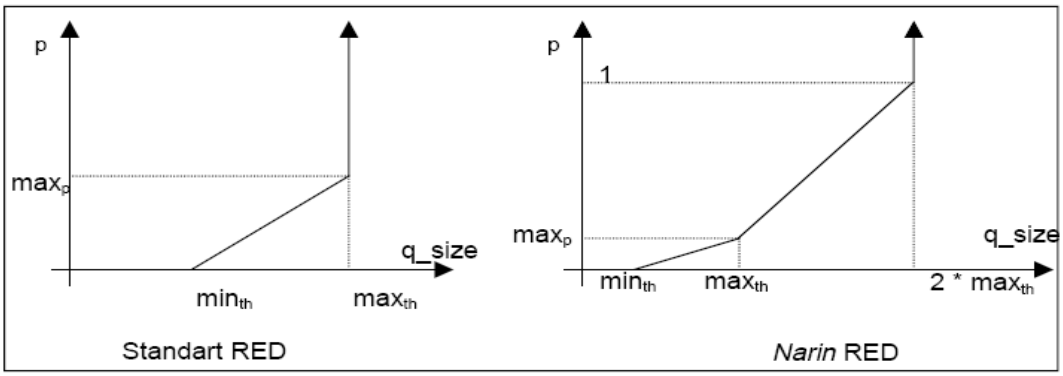


Şekil 3.1 RED algoritması

Ortalama kuyruk boyutu hesabı, her yeni paket geldiğinde hesaplandığından, hesaplamada hattın boş olduğu anlarda da hesaplama yapılır. Ortalama kuyruk boyu ağırlıklı olarak hem ortalama kuyruk boyunu hem de anlık kuyruk boyunu kullanarak hesaplanır. Bu hesaplamanın formülü denklem 3.1'de yer almaktadır.

$$avq = (1-w_q) * avg + w_q * q \tag{3.1}$$

Bunun sonucunda kuyruk boyundaki ani artışlar göz ardı edilir. Sadece uzun süreli tıkanıklıklara cevap verilmiş olunur. Böylece kuyruk boyundaki artışlar ortalama kuyruk boyuna olduğu gibi yansımaz. Parametrelerin birbirleriyle olan bağıntıları Şekil 3.2'te gösterilmiştir.

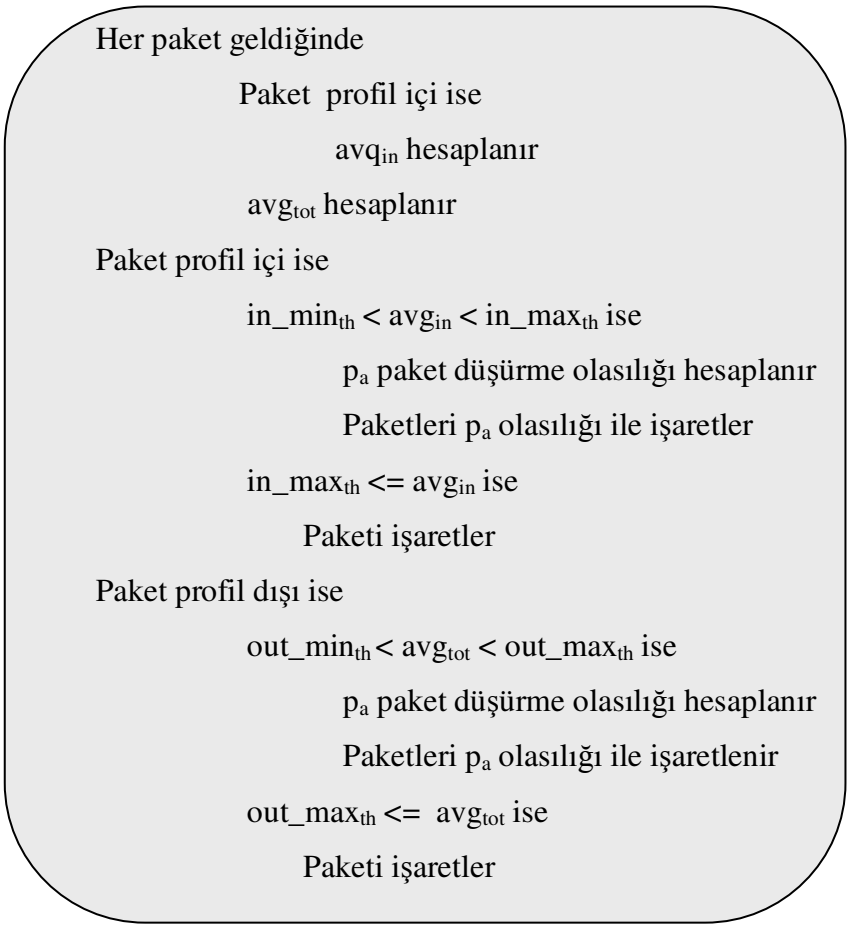


Şekil 3.2 RED parametreleri (Okuroğlu B. ve Oktuğ S. ,2003)

Tüm bunlara rağmen RED algoritmasının parametre seçimi çok kolay değildir. Parametreler yönlendirici üzerindeki trafiğin karakteristiğine ve aktif akış sayısı gibi etkenlere göre değişmektedir. Parametreler hedefler göz önüne alınarak seçilmeli ve şunlara dikkat edilmelidir.

1. Ağın verimi: Kuyruğun boş kalması ile hattın boşta olmasını engellemek için min_{th} değeri büyük seçilmelidir. Böylece kuyrukta sürekli iletecek veri olacaktır.
2. Gecikme süresi: Kuyruk boyunun az olması gecikmeyi azaltacağından min_{th} değeri küçük seçilmelidir.
3. Adil hizmet: Wq parametresi izin verilen patlamalı trafik oranını belirttiğinden küçük seçilmesi anlık kuyruk boyunun, ortalama kuyruk boyuna etkisini azaltır ve daha fazla patlamalı trafiği tolare etmeyi sağlar.

RIO algoritması ise iki ayrı RED algoritmasının IN ve OUT profilli paketleri için kullanılarak kuyruk boyutunun ayarlanması şeklinde çalışmaktadır. RIO algoritmasının akışı aşağıdaki Şekil 3.3'te yer almaktadır.



Şekil 3.3 RIO algoritması

3.5.3 BLUE ve BIO (BLUE with IN and OUT)

BLUE, diğer kuyruk yönetim algoritmalarının tersine kuyruk yönetimini paket düşürme ve hattın kullanım geçmişine dayanarak yapmaktadır. BLUE, paketleri işaretlemek için tek bir p_m olasılığı kullanmaktadır ve bu olasılık kuyruğun taşarak paketin düşmesi veya hattın boşta kalmasına göre değişmektedir. Kuyruktan sürekli paketler düşürülüyorsa mevcut olasılık artırılır. Hattın kullanıma boş kalması durumunda ise bu olasılık değeri azaltılır. BLUE tıkanıklığı kaynağa haber vermek için paket düşürür veya ECN ile paketleri işaretler (W.Feng,1999).

Bu yöntemdeki parametreler freeze_time, d_i ve d_d değerleridir. Freeze_time parametresi anlık değişimlerin göz ardı edilmesi ve değişimlere fazla tepki vermesini önlemeye yöneliktir. d_i ve d_d parametreleri ise olasılık değerinin artım ve azalım değerleridir.

Yapılan benzetimlerde d_i parametresi d_d parametresine göre oldukça büyük seçilmektedir. Bunun nedeni işaretleme olasılığının hem gereğinden fazla agresif hem de gereğinden fazla

muhafazakar olmasının önüne geçmektir. Paketler fazla işaretlendiğinde hat boş kalacak, az işaretlendiğinde ise kuyruk taşması sonucunda oluşacak toplu paket düşüşleri sayesinde kuyruk boyu sürekli salınacaktır. Bunun sonucunda yine hattın boş kalması olasıdır. BLUE algoritmasının akışı Şekil 3.4 'teki gibidir.

```
Paket kaybında (veya Q_len > L)
        if ( (now-last_update) > freeze_time )
                pm = pm + d_i;
                last_update = now;
Link boş kaldığında
        if ( (now-last_update) > freeze_time )
                pm = pm - d_d;
                last_update = now;
```

Şekil 3.4 BLUE algoritması

BIO (BLUE with IN and OUT) mekanizmasının temelinde BLUE algoritmasının IN ve OUT paketler için farklı parametreler kullanarak farklı düşürme olasılıkları hesaplama işlemi bulunmaktadır. Algoritmanın detayı Şekil 3.5'da görülmektedir.

BLUE konusunda yapılan farklı çalışmalar da bulunmaktadır. Bunlardan MBIO (Modified BLUE with IN and OUT) olasılık değerini artırmak ve düşürmek amacıyla kullandığı di ve dd değişkenlerini kuyruğun doluluk oranıyla orantılı olarak her seferinde belirleyip bu değerlerde ağırlıklandırmaktadır (Ming Jiang vd,2006). Bu konuda yapılan diğer bir çalışma ise DT_BLUE algoritmasıdır. Bu yapının çıkış noktası parametrelerin düzgün ve uygun olacak şekilde seçimindeki zorluğun giderilmesidir. Burada sınır değerler ve artım parametreleri uyumlu şekilde değişmektedir (Wei-yan Liu,2006).

Her paket geldiğinde
Paket profil içi ise
    Paket kaybında (veya $Q_{len} > maxth_{in}$)
        if ((now-last_in_update) > freeze_in_time)
            $pm_{in} = pm_{in} + di_{in}$;
            last_in_update = now;
    Link boş kaldığında
        if ((now-last_in_update) > freeze_in_time)
            $pm_{in} = pm_{in} - dd_{in}$;
            last_in_update = now;
    Paket pm_{in} olasılığı ile işaretlenir.
Paket profil dışı ise
    Paket kaybında (veya $Q_{len} > maxth_{out}$)
        if ((now-last_out_update) > freeze_out_time)
            $pm_{out} = pm_{out} + di_{out}$;
            last_out_update = now;
    Link boş kaldığında
        if ((now-last_out_update) > freeze_out_time)
            $pm_{out} = pm_{out} – dd_{out}$;
            last_out_update = now;
    Paket pm_{out} olasılığı ile işaretlenir.

Şekil 3.5 BIO algoritması

3.6 Açık Tıkanıklık Bildirimi (ECN – Explicit Congestion Notification)

Ağlarda oluşan tıkanıklık sonucunda paketler düşürülmekte ve iletimde istenen hizmet kalitesine ulaşılamamaktadır. Bu durumda tıkanıklığı algılamak ve olabildiğince az gecikme ve paket kaybıyla iletim yapabilmek önemli bir olgu olmaktadır. Aktif kuyruk yönetim yapıları tıkanıklığı uç düğümlere bildirmek için paketleri düşürebileceği gibi paketlerin belirli alanlarını işaretleyerek hatta tıkanıklık oluştuğunu belirtebilirler. Bu yapıya açık tıkanıklık bildirimi (ECN) ismi verilmektedir[5].

ECN gereksiz paket düşürmesinin engellenmesinin yanında tıkanıklığın çok daha erken bir şekilde kaynak tarafından algılanmasını sağlar. Bunun dışında paket düşürüldüğünde o paket için sarf edilen gücün kaybedilmesi önlenmiş olur. ECN kullanılmadığında tıkanıklık 3 çift paket alındısı ile ya da RTO süresinin aşılması ile algılanırdı. ECN ile ise en geç bir RTT sonrasında tıkanıklık bilgisi kaynak düğüme iletilir.

IP paket başlığında ECN desteği için iki bit bulunmaktadır. Birinci bit ECT (ECN Capable Transport) bitidir. Bu alan uç düğümlerin ECN desteklediğini belirtir. Diğer bit ise yönlendiriciler tarafından tıkanıklık oluştuğunu belirtmek amacıyla işaretlediği CE (Congestion Experienced) alanıdır. ECT ve CE için IP paket başlığındaki TOS alanının 6 ve 7 numaraları bitleri kullanılır.

3.7 Tıkanıklık Göstergeleri ve Kontrol Mimarileri

Tüm AQM tarımlarında kuyruk yönetimi için önemli üç temel konu vardır. Bunlar uygun tıkanıklık göstergelerini seçmek, kontrol yapısını oluşturmak ve işaretleme olasılığının hesaplanmasıdır.

Tıkanıklı göstergeleri AQM yapıları içinde tıkanıklığı belirlemekte kullanılan ve bu tıkanıklığa önlem almada karar vermeye yardımcı olacak parametrelerdir. En çok kullanılan tıkanıklık göstergeleri kuyruk boyutu, paket kayıp oranı, aktif akış sayısı, trafik hızı ve gecikme süresidir. Tıkanıklık göstergeleri ve kontrol prensipleri ile ilgili bilgileri özet halinde Çizelge 3.1 ‘de verilmiştir.

Çizelge 3.1: AQM Tıkanıklık Göstergeleri ve Kontrol Prensipleri (Zhi Li,2004)

AQM Yapısı	Tıkanıklık Göstergeleri	Kontrol Prensipleri
RED	Kuyruk boyu	Kuyruk boyunu belli aralıkta tutmak
SRED	Kuyruk boyu ve tahmini aktif akış sayısı	Kuyruk boyunu kararlı tutmak
ARED	Kuyruk boyu ve kuyruk uyumsuzluğu	Ortalama kuyruk gecikmesi sunmak
REM	Hat maliyeti	Hız uyumluluğunu yakalamak
PI	Kuyruk uyumsuzluğu ve kuyruk boyutundaki değişim	Hız uyumluluğunu yakalamak
BLUE	Paket kaybı ve hattın boşalması	Paket düşürmeyi önlemek ve hat verimliliğini yüksek tutmak
AVQ	Sanal kapasite	Hat verimini tanımlanmış değere yakın tutmak
GREEN1	Hız uyumsuzluğu	Hız uyumluluğunu yakalamak
GREEN2	Hız (RTT) ve aktif akış sayısı	TCP tıkanıklık kontrol davranışlarını yönlendiricilerde uygulamak
ADR	Teklif edilmiş yükleme ve hat doluluk oranı	Değişken hat kapasitesi, buffer alanı ve trafik yoğunluğu ile ölçeklenebilir olmak

3.8 Olasılık İşaretleme ve AQM Karşılaştırmaları

Her AQM algoritmasının kendine özgü tıkanıklık göstergeleri ve bunlara bağlı olarak da kontrol prensipleri bulunmaktadır. Bu prensipleri tıkanıklığı yönetmek için olasılık hesaplayarak uygularlar. Bu hesaplamalar doğal olarak tıkanıklık göstergelerine ve kontrol prensiplerine bağlıdır dolayısıyla her AQM'in kendine özgüdür.

4. FUZZY BIO TASARIMI

TCP protokolünün doğal dinamik yapısı ve karmaşıklığı, aktif kuyruk yönetim yapılarında gelişime ihtiyaç doğurmuştur. Fuzzy lojik yapılar (FL) birçok alanda kullanılmakta ve başarılı olmaktadır. Araştırmalar, FL gibi yapay zeka içeren teknolojilerin trafik kontrol mekanizmalarının içinde birçok alanda kullanılabileceğini ortaya çıkarmıştır. Bu bölümde ayrıntılı şekilde fuzzy lojik kontrolleri kullanarak aktif kuyruk yönetim mekanizması tasarımı ve geliştirilmesi incelenecektir. Öncelikle genel fuzzy lojik kontrolü (FLC) yapıları incelenmektedir. Sonrasında da tasarım adımları sırayla açıklanmakta ve tasarlanan yapı anlatılmaktadır.

Fuzzy BIO tasarımı, temel olarak bir fuzzy lojik kontrol (FLC) tasarımına dayanır. Bu FLC yönlendiricide tıkanıklığı önleyici olarak işlem görmektedir. Fuzzy lojik kontrol tasarımında iki farklı tasarım metodolojisi kullanılmaktadır. Bunlardan ilki deneme-yanılma (trial-and-error) yöntemidir, diğeri ise teorik (theoretical) yöntemdir. Tasarlanan FLC de ilk yöntem kullanılmıştır. Bu yöntemde IF-THEN kuralları tecrübe ve bilgiye dayanılarak oluşturulur ardından test edilir. Performansı yeterli görülmezse kurallar tekrar oluşturulur ve dizayn yenilenir. Bu döngü yeterli performans yakalanana kadar sürdürülür (S. Tagshavi Zargar vd.,2006).

4.1 Aktif Kuyruk Yönetimi İçin Fuzzy Kontrol Uygulaması

AQM tasarımlarının dayandığı en önemli iki nokta tıkanıklık göstergelerinin seçimi ve düşürme olasılığının hesaplanma yöntemidir. Çoğu AQM, kuyruk boyutunu veya trafik hızını baz almaktadır.

TCP/IP ağları, kuyruk yönetim yapılarıyla birlikte geri beslemeli kontrol sistemleri haline gelmektedir. Kontrol teorisinin aktif kuyruk yönetim yapılarında uygulanabilmesiyle daha esnek ve performanslı uygulamalar tasarlanabilir olmuştur. Klasik kontrol teorisi ile geliştirilen ilk yapılar PI yapılarıdır. Bu uygulama TCP dinamiklerine matematiksel modelin dahil edildiği öncü çalışmalardan olmuştur. Bununla beraber model geliştirme sürecinde bazı sınırlamalar mevcuttur. Örneğin verimin doğrusallığı sağlayan basit modeller kesin kontrol teknolojilerine gereksinim duymaktadırlar. Uzun süreli bağlantılar ve gecikmeden bağımsız sistemler için bazı tahminler yapılması gerekliliği vardır. Bunlara ek olarak bazı dinamik TCP yapıları göz ardı edilmektedir (slow start ve timeout). Tüm bu sebeplerden herhangi bir klasik

kontrol teorisi tabanlı bakış büyük olasılıkla performans ve süreklilik açısından istenen başarımı göstermeyecektir. Klasik kontrol teorisinin AQM tasarlama sürecindeki zayıflığı, doğasından gelen lineer olmayan ve karmaşık yapısındandır.

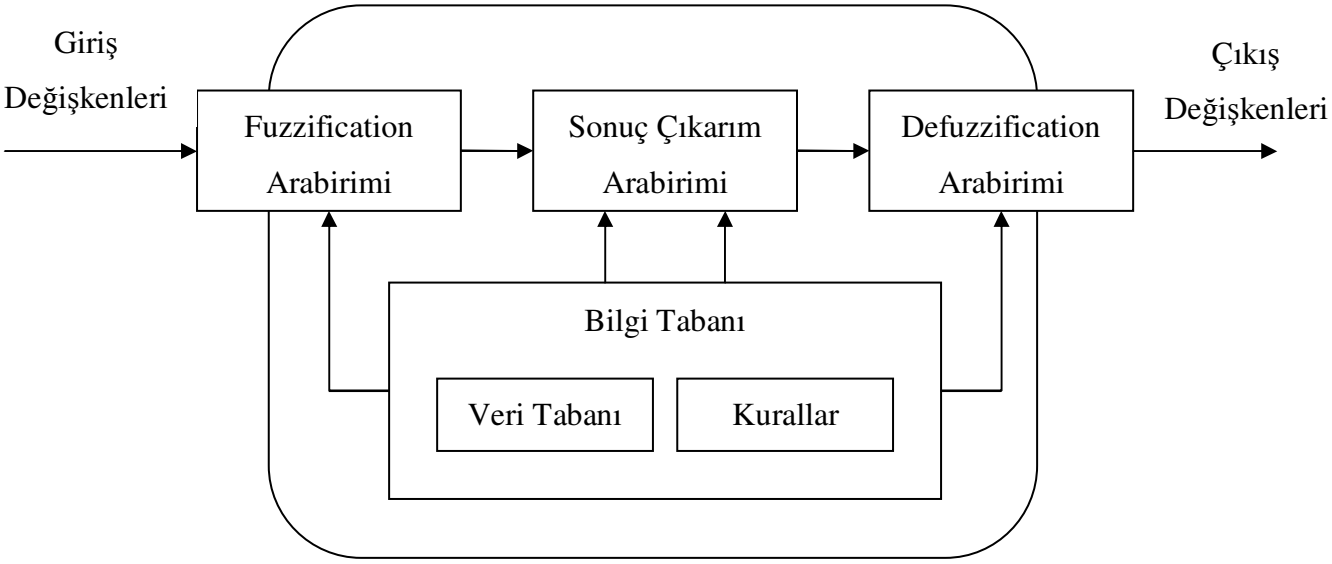
TCP/IP ağları şüphesiz karmaşık ve doğrusal olmayan sistemlerdir. Bunun da sebebi herhangi bir TCP dinamiğinin doğrusal olmaması, trafikteki farklılıklar ve ağın heterojen yapısından kaynaklanıyor olabilmektedir. Bu tarzdaki karmaşık ve doğrusal olmayan kontroller için fuzzy lojik (FL), alternatif kontrol çözümü sunmaktadır. Fuzzy lojik kontrol teorisi içinde doğrusal olmama durumu; kurallar, üyelik fonksiyonları ve sonuç çıkarma süreci ile kontrol altına alınabilmektedir. FL'nin arkasındaki fikir, insanın kavramsal çıkarım yapma süreci ile karar verme ve problem çözme yetisinin FL yapısı içine aktarılmasıdır. Bu sayede tecrübe ve uzman bilgisinin sorunları çözmedeki başarısının kullanılması sağlanmakta ve karşılaştırmalı ve önceden tahmine dayalı çözümlerin dezavantajlarından sıyrılmış olunur. FL yapılarını trafik ve tıkanıklık kontrolleri üzerinde kullanan birçok uygulama mevcuttur.

4.2 Genel Fuzzy Lojik Kontrolü (Generic Fuzzy Logic Control - FLC)

FLC'lerin arkasındaki düşünce, sürecin geleneksel hesaplama teknikleri için oldukça karmaşık olması durumunda veya bilgi kaynağının nitelikli olarak yorumlanabildiği, eksik olduğu ya da kesin olmadığı durumlarda uzman bilgiye dayalı sözel kontrol stratejisini otomatik kontrol stratejisine dönüştürerek karar vermeye yardımcı olmaktır. Genel FLC yapısı dört kısmı kapsamaktadır.

1. Fuzzification Arayüzü
2. Çıkarım Motoru
3. Bilgi Tabanı
4. Defuzzification Arayüzü

Şekil 4.1'de genel fuzzy lojik kontrol yapısı gösterilmektedir.



Şekil 4.1 Fuzzy lojik kontrol (FLC) tasarımı

- Fuzzification arabirimi ölçekleme ve fuzzification fonksiyonlarını içeren iki kısımdan oluşmaktadır.

  - Bu birim ölçekleme işlemini veya giriş değerinin normalizasyonunu gerçekleştirir. Giriş değerlerin normalize edilerek normalize edilmiş boyuta dönüştürülür. Normalize edilmemiş boyut kullanılacaksa bu ölçekleme fonksiyonuna gerek duyulmamaktadır.

  - Fuzzification fonksiyonunu gerçekleyerek giriş datasını fuzzy kümesine (F) dönüştürür. Fuzzy fonksiyonları üyelik fonksiyonlarını kullanmaktadır. Yaygın olarak kullanılan fuzzy fonksiyonu "single-ton fuzzification" 'dır. Bu fonksiyon denklem 4.1 de verilmiştir.

$$\mu_{F(u)} \begin{cases} 1, & u=u_0 \\ 0, & \text{otherwise} \end{cases} \qquad (4.1)$$

- Bilgi tabanı birimi, kuralları ve veritabanını içermektedir.

  - Kuralların bulunduğu bu birimde belli sayıda fuzzy kuralı yer almaktadır. Sözel

değişkenlerin "if-then" yapılarıyla kurulmuş kurallar çerçevesinde örneğin "*if x is A, then y is B"* şeklinde uzman bilgilerle oluşturulması prensibiyle çalışmaktadır. Bu sözel değişkenlerin oluşturulmasında kullanılan üyelik fonksiyonlarından en önemlileri triangle (üçgen) ve trapezoid (yamuk) şeklindeki fonksiyonlardır. Bu fonksiyonların yaygın kullanılmasındaki en önemli etken az hesaplama ihtiyacına sahip olmalarıdır.

- Veri tabanı, fuzzy kuralları içinde, ölçekleme, normalizasyon ve denormalizasyon işlemlerinde kullanılan fuzzy kümesinin üyelik fonksiyonlarını tanımlar.

- Sonuç çıkarma birimi fuzzy çıkarım işlemlerini gerçekleştirir. Burada her giriş değeri için fuzzy kümesine karşılık gelen değerler bulunur ve buradaki kurallara uygulanarak çıkış olarak bir fuzzy kümesi ortaya çıkartılır. Minimum değeri alma ve bu değerleri çarpma işlemleri, fuzzy değerlerinden anlam çıkarmak için kullanılan en önemli iki kuraldır.

- Defuzzification birimi, fuzzification biriminin gerçekleştirdiği işlemlerin tam tersini yapmaktadır.

  - Defuzzification fonksiyonu kontrol çıkış değerinin sözel değerden sayısal değerdeki karşılığına dönüştürmesi işlemini gerçekleştirmektedir. Bu konuda farklı birçok defuzzification stratejileri mevcuttur. Center of Area (COA) metodu en çok kullanılan fonksiyondur.

$$y_0 = \frac{\int \mu_{B_0}(y)\, y\, dy}{\mu_{B_0}(y)\, dy} \qquad (4.2)$$

  - Normalize bir domain kullanılıyorsa bu çıkış değeri için denormalizasyon işlemi yapılmaktadır.

Tüm bu yapıların oluşturduğu FLC'ler birçok fuzzy lojik kontrol uygulamasında başarılı şekilde uygulanmaktadır.

Öncelikler giriş değerleri üyelik fonksiyonları üzerinde karşılık gelen değerlerine dönüştürülür. Ardından her değer için karşılık gelen sözel değişkenler hesaplanır.

Her değer için kurallar kümesi içinde minimum değere karşılık gelen değer hesaplanır. Her kural için çıkan çıkış değerleri birleştirilerek sayısal değere dönüştürülmektedir. Elde edilen bu değer kontrol değeri olarak AQM de kullanılmaktadır.

4.3 Fuzzy BIO (BLUE With IN And OUT) Tasarım Adımları

Fuzzy BIO kontrol yapısı, ağ omurgasındaki kuyrukları, geri beslemeli kapalı çevrim bir sistem içerisinde kuyruk yönetimiyle yürütecek şekilde dizayn edilmiştir. FBIO yapısı düşürme olasılığını hesaplayarak kuyruk yönetiminde kullanmaktadır. Bu doğrultuda en önemli karar hangi parametrelerin bize tıkanıklığı göstereceğine karar vermek ve bu doğrultuda FBIO yapısına girecek değişkenleri öngörmektir. Fuzzy BIO yapısı aşağıdaki Şekil 4.2'deki şemada açıklanmıştır.

Paket profil içi ise
    Paket kaybında (veya $Q_{len} > maxth_{in}$)
        if ((now-last_in_update) > freeze_in_time)
            paket kayıp değişkeni $plost_{in}$ güncellenir
            queue size değişkeni $qsize_{in}$ güncellenir.
    Link boş kaldığında
        if ((now-last_in_update) > freeze_in_time)
            paket kayıp değişkeni $plost_{in}$ güncellenir
            queue size değişkeni $qsize_{in}$ güncellenir.
    last_in_update = now;
Paket profil dışı ise
    Paket kaybında (veya $Q_{len} > maxth_{out}$)
        if ((now-last_out_update) > freeze_out_time)
            paket kayıp değişkeni $plost_{out}$ güncellenir
            queue size değişkeni $qsize_{out}$ güncellenir.
    Link boş kaldığında
        if ((now-last_out_update) > freeze_out_time)
            paket kayıp değişkeni $plost_{out}$ güncellenir
            queue size değişkeni $qsize_{out}$ güncellenir.
    last_out_update = now;

Her paket geldiğinde
    Paket profil içi ise
        $Plost_{in}$ ve $Qsize_{in}$ değişkenlerine göre pm_{in} hesaplanır
        Paket pmin olasılığı ile işaretlenir.
    Paket profil dışı ise
        $Plost_{out}$ ve $Qsize_{out}$ değişkenlerine göre pm_{out} hesaplanır
        Paket pmin olasılığı ile işaretlenir.

Şekil 4.2 Fuzzy BIO algoritması

Fuzzy BIO Tasarım adımları aşağıdaki şekildedir.

4.3.1 Input ve Output değişkenlerin seçimi.

Input değişkenler BIO mekanizmasının anahtar özelliklerini temsil edecek şekilde seçilmelidir. Output değişken ise algoritmayı etkileyen ve durumu ifade eden değişken olmalıdır. BIO algoritmasına bakıldığında input değişkenler olarak paket kayıpları (plost) ile normalize kuyruk boyutu(q_n) alınmıştır. Output değişken olarak da paket düşürme olasılı (p_m) seçilmiştir.

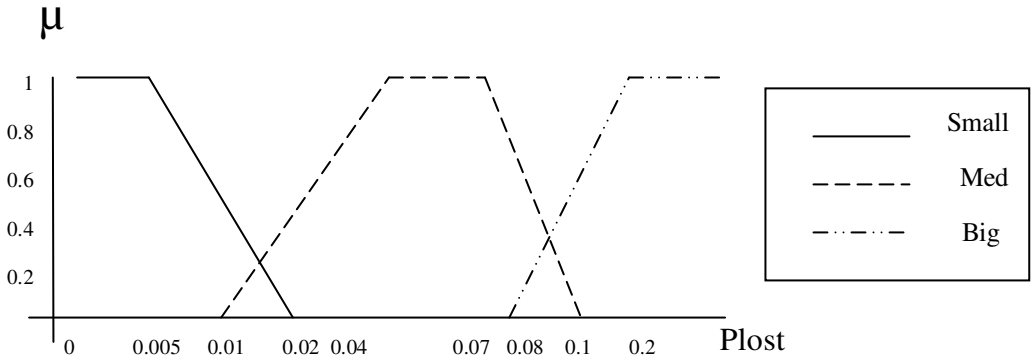
4.3.2 Input ve Output değişkenler için uygun üyelik fonksiyonların seçimi.

Hesaplama işlemindeki basitlikleri nedeniyle üçgen (triangle) ve trapezoid (trapezoidal) fonksiyonların her üç değişken için de kullanılması uygun görülmüştür. Bu doğrultuda fuzzy lojik kontrol yapısındaki giriş ve çıkış değişkenlerinin sözel kümeleri aşağıda verilmiştir.

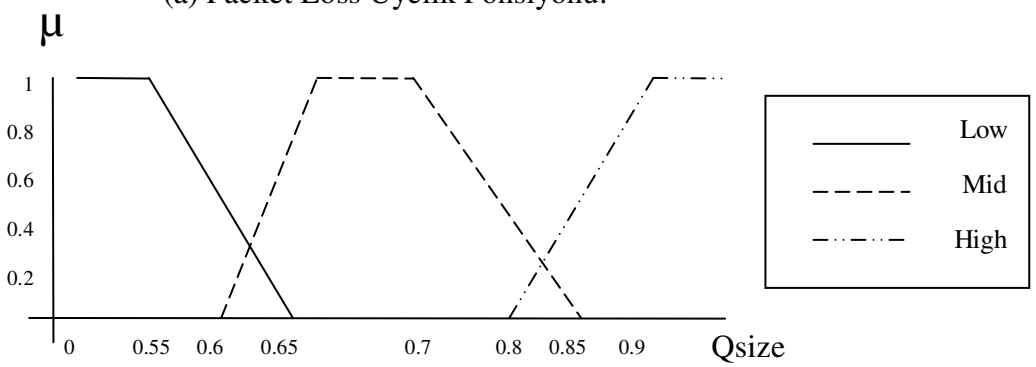
T(packet loss) = {small,medium,big}

T(normalized queue length) = {low,middle,high}

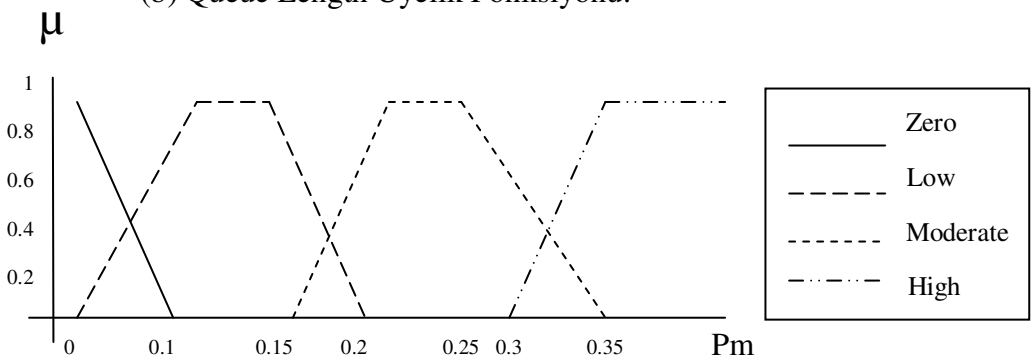
T(Pm) = {zero,low,moderate,high}



(a) Packet Loss Üyelik Fonsiyonu.



(b) Queue Length Üyelik Fonksiyonu.



(c) Drop Probability Üyelik Fonksiyonu.

Şekil 4.3 Değişkenlerin Üyelik Fonksiyonları

4.3.3 Fuzzy IF – THEN kurallarının tasarımı.

Bu kurallar, sistemin nasıl çalışması gerektiğine dair düşünceler neticesinde oluşturulmuştur. Bunu oluşturmada konuyla ilgili bilgi ve tecrübe önemlidir. Fuzzy IF – THEN kuralları iki adımı içermektedir. Öncelikle input değişken değerleri üyelik fonksiyonlarına sokularak bu değerlere karşılık gelen sözel değişkenler belirlenir. Sözel değişkenler de kurallar doğrultusuna sözel çıkış değişkenini oluşturur. Bu kuralların oluşumunda iki yöntem vardır. Birincisi deneme – yanılma, ikincisi de teorik yöntemdir. Deneme – yanılma yönteminde,

kuralların performansı yeterli görülene kadar tasarım döngüsü tekrarlanır. Teorik yöntemde ise bazı parametrelere (örneğin kapasite verimi gibi...) göre dizayn yapılır (M.H. Yaghmaee ve Hale Amintoosi, 2003). Kuralların ayrıntısı Şekil 4.4 'de gösterilmiştir.

Kural 1: if packet loss is small and normalized queue length is low then pm is zero;
Kural 2: if packet loss is small and normalized queue length is mid then pm is zero;
Kural 3: if packet loss is small and normalized queue length is high then pm is zero;
Kural 4: if packet loss is med and normalized queue length is low then pm is zero;
Kural 5: if packet loss is med and normalized queue length is mid then pm is zero;
Kural 6: if packet loss is med and normalized queue length is high then pm is moderate;
Kural 7: if packet loss is big and normalized queue length is low then pm is zero;
Kural 8: if packet loss is big and normalized queue length is mid then pm is low;
Kural 9: if packet loss is big and normalized queue length is high then pm is high;

Şekil 4.4 IF –THEN kuralları

4.3.4 Fuzzifier ve Defuzzifier Fonksiyonları

Fuzzifier fonksiyonu her input değişkeni için üyelik fonksiyonundaki alanı bulur. Defuzzifier fonksiyonu ise output değişkeninin değerini hesaplar. Bu araştırmada single-tone fuzzifier ve Center Of Gravity (COG) defuzzification metotları kullanılmıştır (M.H. Yaghmaee ve Hale Amintoosi, 2003).

4.3.5 Genel süreç

Öncelikle iki giriş değişkeni fuzzifier fonksiyonuna sokularak sözel (linguistic) değişkenler oluşturulur. Bu değişkenler kurallara bakılarak çıkış değerlere karar verilir. Çıkan değerler max-min kombinasyonuna bakılarak output değişkeninin hangi alana dahil olacağına karar verilir. Son olarak da bu değişken defuzzifier metoduyla output değişkenini oluşturur. Bu değişken paketin düşürme olasılığıdır. Çıkış değerini hesaplamada Şekil 4.7'deki COG defuzzifier fonksiyonu kullanılmaktadır.

$$Pm = \frac{\int Pm.\mu\,(Pm)\,dPm}{\int \mu(Pm)\,dPm} \qquad (4.3)$$

5. UYGULAMA VE PERFORMANS ÖLÇÜMLERİ

Bu bölümde öncelikle tasarımın gerçekleneceği NS-2 yapısı anlatılmakta ve ardından uygulamayla ilgili ölçümlemelere yer verilmektedir.

5.1 Network Simulatör NS-2

Bu çalışmamda simülasyon ve ölçümleme yapmak için network projelerinde yaygın olarak kullanılan NS-2 (Network Simulator) ağ benzetim aracı kullanılmıştır. NS-2 simulator programı network uygulamaları geliştirmek için tasarlanmış bir programdır. NS gerçek TCP uygulamalarını, yönlendirme (routing) işlemlerini, çoklu iletim (multicast) protokollerini ve kablosuz (wireless) ağ uygulamalarını gerçeklemek için yapılmış bir network benzetim aracıdır[2].

NS2 Programı bize bir network ortamı oluşturmamız ve oluşturulan network ortamının bağlantı analizlerini yapmamız ve bunu sanal ortamda görebilmemizi sağlayan bir simülator programıdır. Yapılmasındaki amaçlarından bir tanesi bir network oluşturmadan önce bağlantı şekillerini seçip nereye hangi düğümün (printer, router, host) konulacağına karar verilmesini sağlamaktır. Yani dinamik ve elverişli bir network oluşturmamız için bize yardımcı olmaktadır.

NS aracı C++ ve TCL dilleriyle oluşturulmuştur. NS yazılımının çekirdeği (kernel) simülasyonun yüksek işlem gerektiren temel yapıtaşlarını barındırdığından C++ programlama diliyle yapılmıştır. Diğer yandan; kullanıcı arayüzü, ağ konfigürasyonu ve trafik yapısı gibi kısımları esnek ve interaktif bir dil olan TCL ile yazılmıştır. Bu aracın içinde hali hazırda Drop Tail, RED, ARED, REM, PI ve BLUE gibi en bilinen kuyruk yönetim mekanizmalarının implementasyonları mevcuttur.

NS network programının olmazsa olmaz yardımcıları NAM (Network Animasyon Ortamı) ve TCL programlarıdır. TCL, NS'in derlendiği bir compiler programıdır ve NS'te veya TCL'de yazılan kodlar bu compiler tarafından derlenir. NS ile yazdığımız bir network animasyonunu görmemiz için ise NAM programı kullanılmaktadır. Bu program yazılan ve derlenen kodların animasyon görüntülerini göstermektedir.

5.1.1 NS-2 Yapısı

NS-2'nin temel işlevlerinin gerçekleştirildiği çekirdek kısmı C++ ile nesne tabanlı bir yapıda yazılmıştır. Bu aracı, kullanıcılar benzetim senaryosu yani OTCL (Object TCL) yazarak

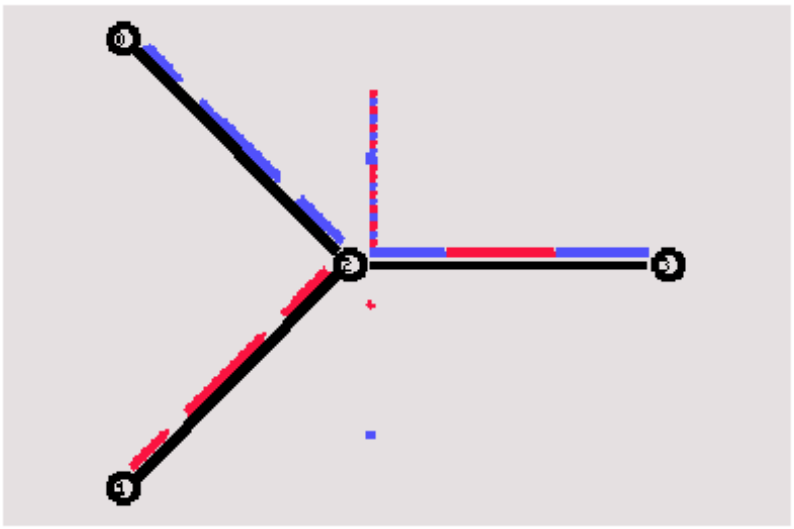
kullanmaktadırlar. Bunu gerçeklemek için C++ kodlarının OTCL arabirimi bulunmaktadır.

Benzetim, simülator isimli sınıf tarafından yöneltilmektedir. Temel işlevler ise Node, Link ve Agent sınıfları ve bunların alt sınıfları tarafından yapılmaktadır.

Node sınıfı ağdaki düğümü temsil etmektedir. Hem uç düğüm hem de yönlendirici olabilmektedir. Paket alıp vermek ve yönlendirme işlemlerini yapabilmektedir. Link sınıfı ise düğümleri birbirlerine bağlamaktadır. Tek yönlü, çift yönlü ve CBQ hatları kullanılabilmektedir. Düğümlerin çıkışlarında kullanılan kuyruklar Queue sınıfları ile temsil edilmektedir. DT, RED, FQ, SFQ, DRR, CBQ kuyruk tipleri standart olarak desteklenmektedirler.

Simülasyonu asıl yöneten sınıf ise Agent sınıfıdır. Bunu iletim seviyesi protokolleri işleten kısım olarak düşünebiliriz. İletim seviyesi protokollerini gerçekleyen bu sınıfların üzerinde ise çeşitli uygulamaları simule eden Application sınıfı yer almaktadır. Bu sınıfın telnet, ftp, traffic gibi alt sınıfları mevcuttur. Traffic sınıfı pareto dağılımına göre trafik üreten Traffic/Pareto, exponansiyel trafik üreten Traffic/Expoo ve sabit bir trafik üreten Traffic/CBR gibi alt sınıfları bulunmaktadır[5].

NS, benzetimleri ölçümlemek amacıyla veri toplamak ve bunların görüntülenmesini desteklenmekte, ayrıca bu amaca yönelik kuyruk ve akış izleme sınıfları bulunmaktadır. Bu sayede belirli zamanlarla verileri dosyalarda toplayabilmektedir. Toplanan verilerin izlenebilmesi ve analiz edilmesi için iki temel araç mevcuttur. Bunlardan ilki nam'dır (network animatör). Bu araç sayesinde simülasyon sürecinde tüm olanlar toplanan veriler sayesinde tekrar oynatılabilmektedir.

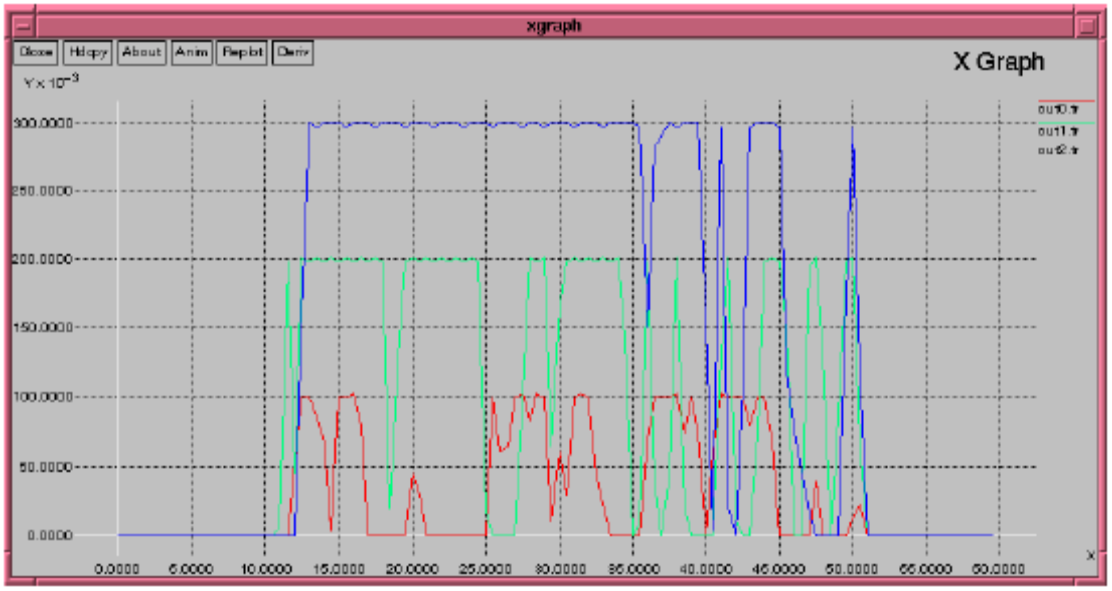


Şekil 5.1 NAM

Şekil 5.1'deki NAM görüntüsünde bulunan dairesel şekiller düğümleri ifade eder. Burada 0 ile 2 numaralı düğümler arasındaki akış mavi renkle, 1 ile 2 numaralı düğümler arasındaki akış ise kırmızı renkle ifade edilmektedir. 2. düğümde bellekte biriken ve her iki akıştan gelen paketler kareler ve dikey olarak sıralanmış şekilde gösterilmektedir. Düşen paketler ise aşağıya doğru hareket eden kareler olarak temsil edilmektedir.

Nam, benzetimin genel gidişini görmek ve bazı basit yapıların işleyişini kontrol etmek için uygun olsa da istatistiksel bilgiler için yeterli olmamaktadır. Ayrıca bilgi sayısı ve benzetim süresi arttıkça dosya boyutu çok büyüyebilmektedir.

Bu tür kuyruk boyutu, paket kayıpları gibi istatistiksel bilgileri görüntülemek amacıyla Xgraph aracı kullanılmaktadır. Verileri kıyaslama ve görme amaçlı kullanım için oldukça uygun bir araçtır.



Şekil 5.2 Xgraph

Şekil 5.2'de gösterilen grafikte 3 farklı yapının birbirlerine göre durumlarını farklı renklerdeki eğrilerle gösterilmektedir.

5.2 FBIO İçin NS-2 Gerçekleştirmesi

Çalışmalarımızda NS-2'nin 2.31 versiyonu kullanıldı. Öncelikle mevcut kuyruk yönetim yapılarının gerçekleştirmeleri incelendi. Sonrasında mevcut BLUE kodu BIO yapısına dönüştürülerek modifiye edildi. Bu aşamada RED ve RIO gerçekleştirmeleri dikkate alındı. Daha sonraki aşamada BIO algoritması fuzzy lojik yapısıyla bütünleştirilerek performans denemeleri yapıldı. Bu aşamadan önce özellikle default değerler ve obje tanımları NS-2 içindeki gerekli dosyalarda (ns-default.tcl ve make) eklendi [7]. Simülasyon işlemi içinde network topolojilerinin tasarlandığı ve istatistiksel bilgilerin toplanıp sonrasında analiz edildiği TCL dosyaları tasarlandı. Test aşamasında da FBIO yapısı BIO ve RIO yapıları ile karşılaştırılmıştır.

5.3 Trafik Modeli

Günümüzdeki trafik modellerine baktığımızda 3 temel model bulunmaktadır[5].

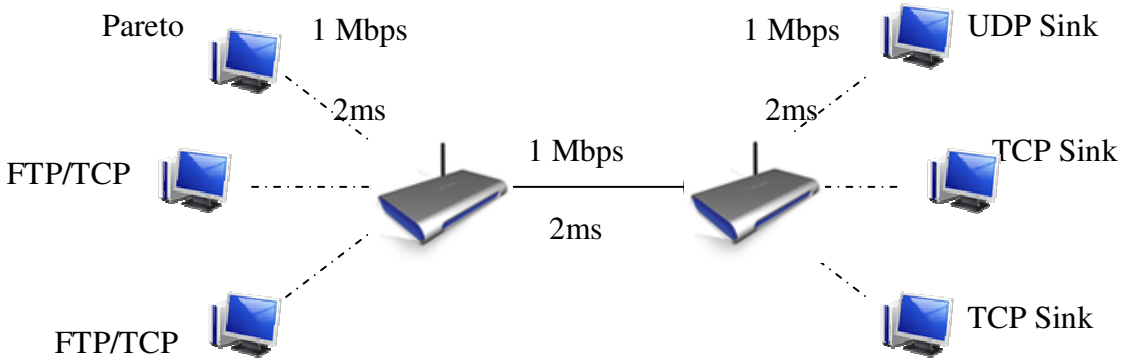
CBR: Cbr, sabit bit akışı gereksinimi olan video ve ses aktarımlarında kullanılır. Uçtan uca sabit bir band genişliği garanti eder. Sayısal telefon santrali olan pbxler bu hizmet sınıfı üzerinden birbirlerine aktarım yapabilir.

Pareto: Internet trafik verilerinin yapılan gözlemler sonucunda internetin pareto dağılımına uyduğu belirlendi. Bir hattı ele aldığımızda da bu hattan geçen çok sayıda ve her biri pareto dağılımına uyan yapıtaşlarının toplam etkisine yeni bir hattan geçen network trafiğine bakıldığında da bu trafiğin self similar veya long range dependent olduğu ortaya çıktı (yani kendini tekrar eden). Bu durumda verilerin istatistiksel yapısı (örneğin dağılımı, varyansı gibi) birbirine benzemektedir. Bu nedenle ileriye dönük trafik tahminleri yapma ve bu tahminlere göre bant genişliği ayarlama, kaynakları verimli kullanma şansı vermektedir.

Exponansiyel: Kablosuz ağlarda yapılan çalışmalara göre kablolular kadar self similar olmadığı ortaya çıkmıştır. Bunun sebebi ise kablosuz ağlarda o kadar büyük çapta videoların ve resimlerin dönüp dolaşmaması, bunun teknik olarak daha zor olması. Bu yüzden kablosuz ağlarda trafiğin yapıtaşlarının büyüklüğünün dağılımı pareto değil de exponansiyel dağılıma daha çok uyuyor ve kendini tekrar etme oranı daha az olmaktadır.

5.4 Network Topolojisi

Araştırmam doğrultusunda, kaynak ve hedef düğümleri birbirlerine bağlayan iki yönlendirici arasındaki omurga hattının performansını ölçümlemek için Şekil 5.3'de gösterilen basit bir ağ topolojisi kullanılmıştır. Simülasyon için oluşturulan topolojide, internetin trafik modeli olan Pareto, FTP/TCP ve UDP akışları bulunmaktadır.



Şekil 5.3 Simulasyon Ağ Topolojisi

5.5 Performans Metrikleri

Kullanıcı ve ağ açısından QoS performansını analiz etmek ve performansın gelişimini ölçümlemek amacıyla iki çeşit ölçüm metrikleri mevcuttur. Bunlar kullanıcı merkezli ve ağ merkezli parametrelerdir.

- **Kullanıcı Merkezli**

- Kullanıcı Verimi: Hedefin tekrar paketleri hariç aldığı paket sayısının zamana bölümüdür.

- Cevap Süresi: İsteğe verilen cevaba kadar geçen süre. İstek sonrası ilk paketin serverdan çıkan son paketin istemci tarafından alınma zamanına kadar geçen süredir.

- **Ağ Merkezli**

- Ağ Verimi: Tüm hedef düğümlerin tekrarlanan paketler hariç aldıkları paket sayısının geçen süreye oranıdır.

- Hattın Kullanılabilirliği: Transfer hızının hattın kapasitesine oranıdır.

- Paket Kayıp Oranı: Toplam düşen veri miktarının toplam aktarılan veri miktarına oranıdır.

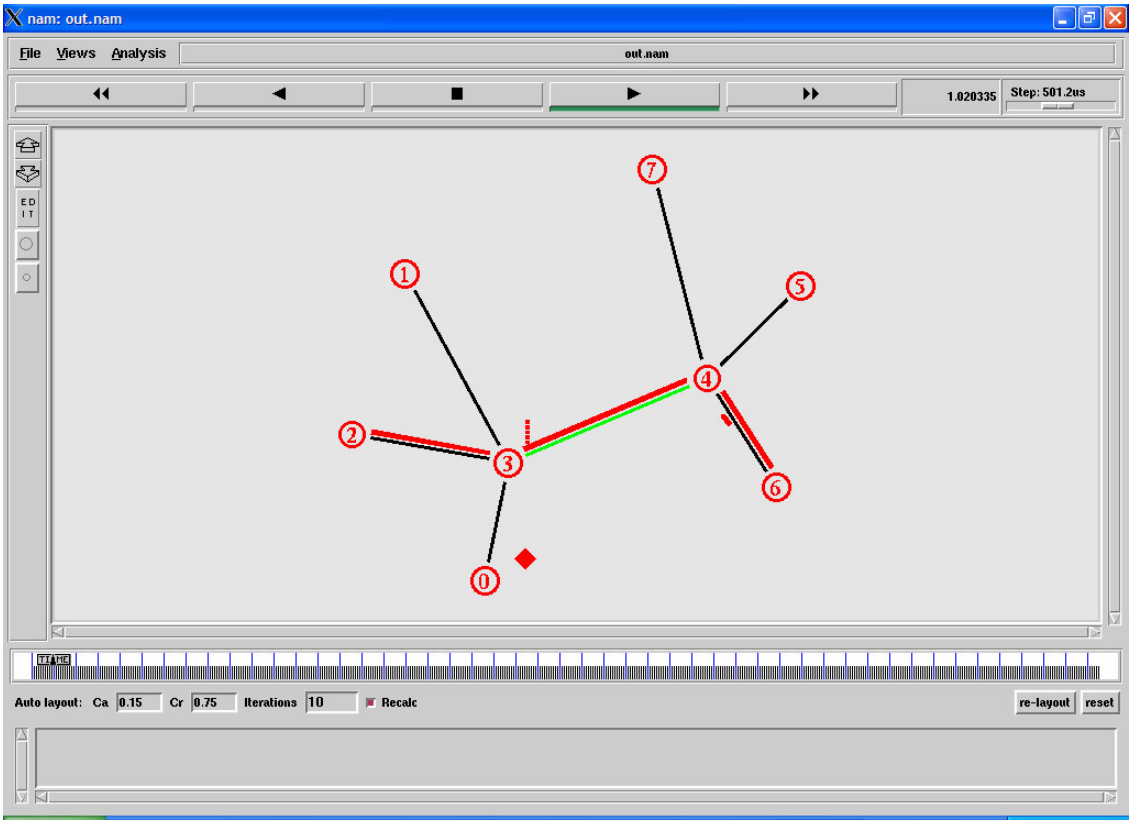
5.6 Performans Ölçümleri

Performans ölçümlerinde Fuzzy BIO tasarımı benzer algoritmalarla (örnek olarak BIO ve RIO kuyruk yapıları) aynı senaryolar üzerinde ölçümlenerek karşılaştırma yapılmıştır. Bu karşılaştırmalarda her bir aktif kuyruk yönetim algortiması için kuyruk boyutu (queue size), paket kaybı (packet loss), kuyruk gecikmesi (queue delay) ve hattın verimi (throughput) ölçümlemeleri yapılmış ve bu sonuçlar karşılaştırılarak yorumlanmıştır.

5.6.1 Test Senaryosu – 1

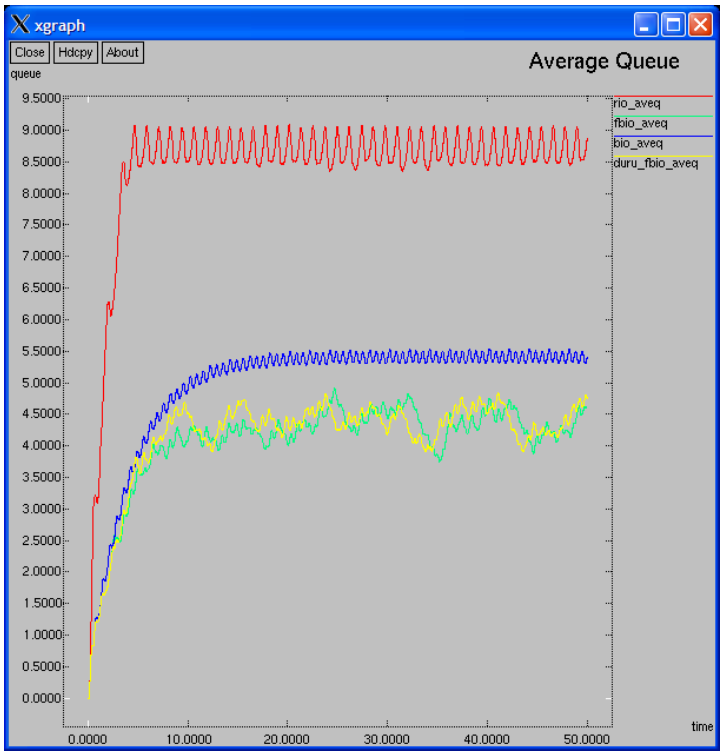
İlk test senaryomuzda, ağ topolojisindeki tüm düğümler arasındaki hatların kapasiteleri 1 Mbps ve gecikmeleri 2ms olarak belirlenmiştir. Senaryo gereğince Pareto trafiği oluşturan n0 ve ftp trafik oluşturan n1 ile n2 düğümlerimiz mevcuttur. Bunların dışında hedef düğümlerimiz n5 ,n6, n7 yer almaktadır. Trafik oluşturan tüm düğümler 0.0 zamanında trafik

yaratmaya başlarlar. Buffer boyutu olarak 20 paket verilmiştir.



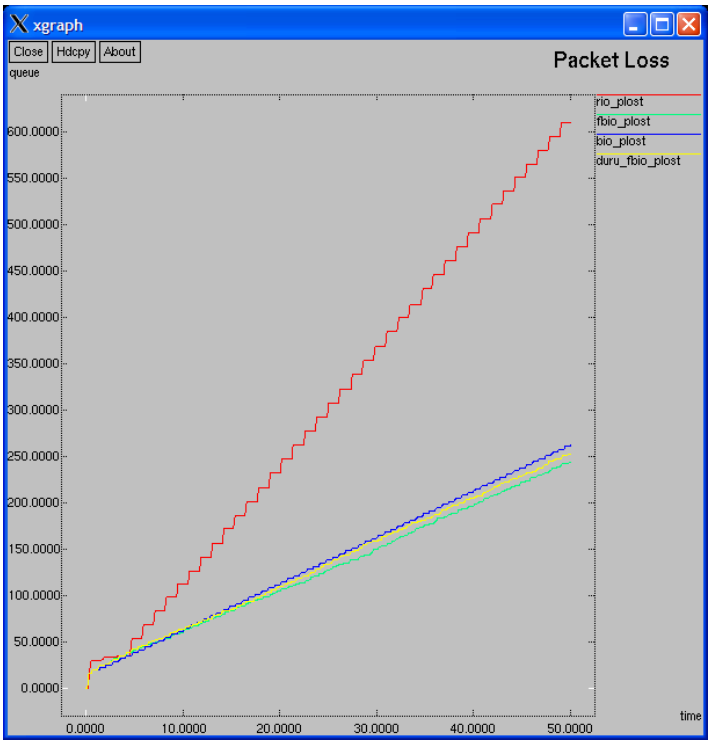
Şekil 5.4 Test senaryosu NAM görünümü

Şekil 5.4'te görünen NAM görüntüsünde yeşil ile görünen hat yönlendiriciler arasındaki hattı göstermektedir. Sayıların bulunduğu kırmızı dairelerin her biri bir düğümü göstermektedir. Kırmızı çizgilerin bulunduğu düğüm bağlantıları akışın mevcutta bulunduğu hattı göstermektedir. Aksi yönde 4 ve 6 numaralı düğümler arasındaki hatta görülen kısa kırmızı çizgi ise ACK mesajlarını ifade etmektedir. 3. düğüm üzerinde bulunan dikey sıralı kareler bufferdaki bekleyen paketleri temsil eder. Aşağıya doğru hareket etmekte olan büyük kare şekil ise düşürülen paketi ifade etmektedir.



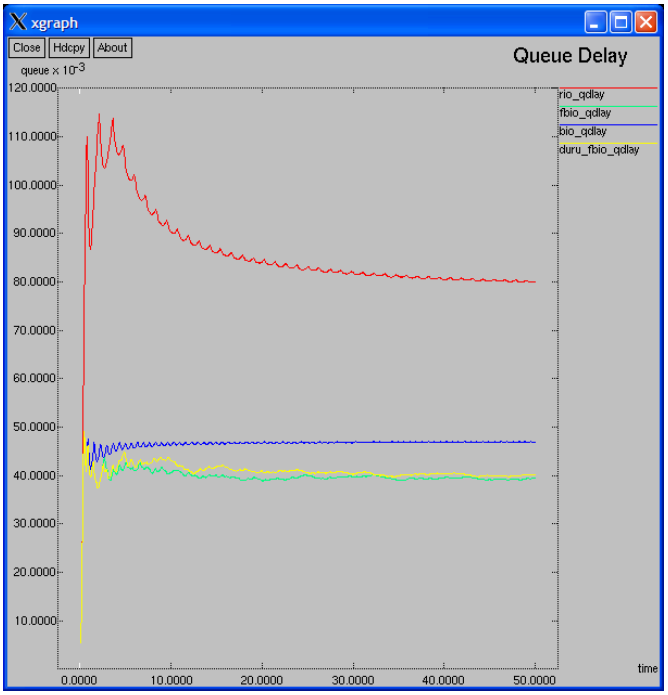
Şekil 5.5 Ortalama Kuyruk Boyutları (Average Queue Sizes)

Simülasyon sonucundaki kırmızı renkli değerler RIO, mavi renkli değerler geleneksel BIO, yeşil renkli değerler bizim geliştirdiğimiz FBIO algoritmasını ve sarı renkli eğri ise FBIO olgaritmasının duru halini temsil etmektedir. Duru hal bulanık mantık yapısındaki üyelik fonksiyonlarındaki bölgelerin birbirleriyle iç içe olmayıp sınır halinde bulundukları durumdur. Görüldüğü üzere Fuzzy BIO algoritması geleneksel algoritmaya göre daha iyi sonuç vermektedir. Bakıldığında diğer aktif kuyruk yönetim mekanizmalarına göre daha az kuyruk kapasitesi yani bellek gereksinimi sunmaktadır. Bu sayede yönlendiricilerde hafıza kapasitesi daha az ve verimli kullanılabilir. Algoritmanın duru halinin de çok yakın değerler aldığı görülmektedir.



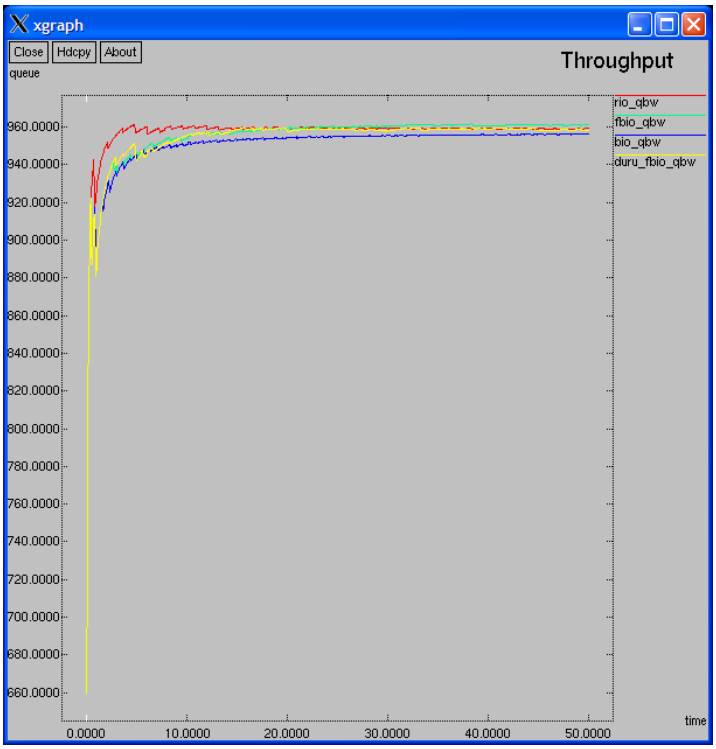
Şekil 5.6 Paket Kayıp Büyüklükleri (Packet Loss)

Simülasyon sonuçlarına göre RIO algoritması başlangıçta daha az paket düşürmüş olmasına rağmen, sonrasında diğerlerini bir hayli geçmiştir. Burada yine Fuzzy BIO yapısının diğer yapılara bakış daha az paket kaybı yarattığını görmekteyiz.



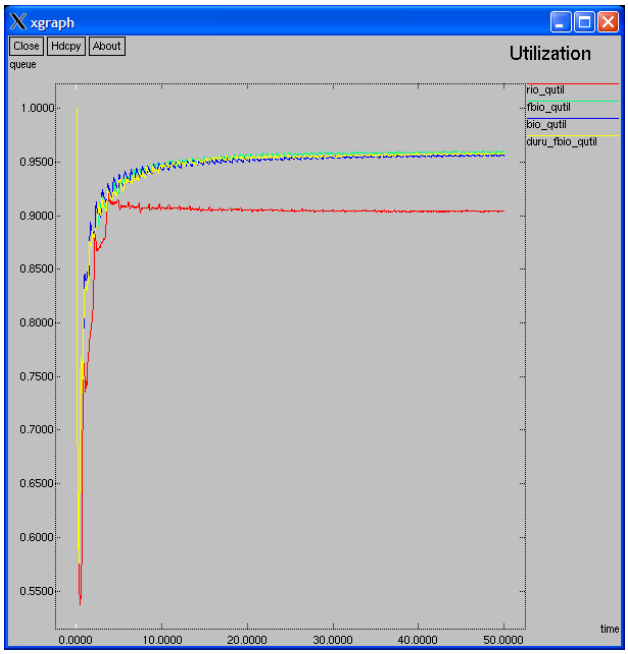
Şekil 5.7 Gecikme Süreleri (Queue Delay)

Gecikme sürelerine bakıldığında RIO ve BIO arasında oldukça fark oluştuğu görülmektedir. Fuzzy BIO yaklaşık 40 ms gecikirken BIO 50 ms, RIO ise 105 ms gecikmektedir. Bu durumda yine Fuzzy BIO yapısının bir üstünlüğü bulunmaktadır.



Şekil 5.8 Hattın Verimi (Throughput)

Hattın verimliliğini belirlemek amacıyla yapılan bu ölçümlemede her üç aktif kuyruk yapısının da birbirlerine çok yakın oldukları belirlenmiştir.

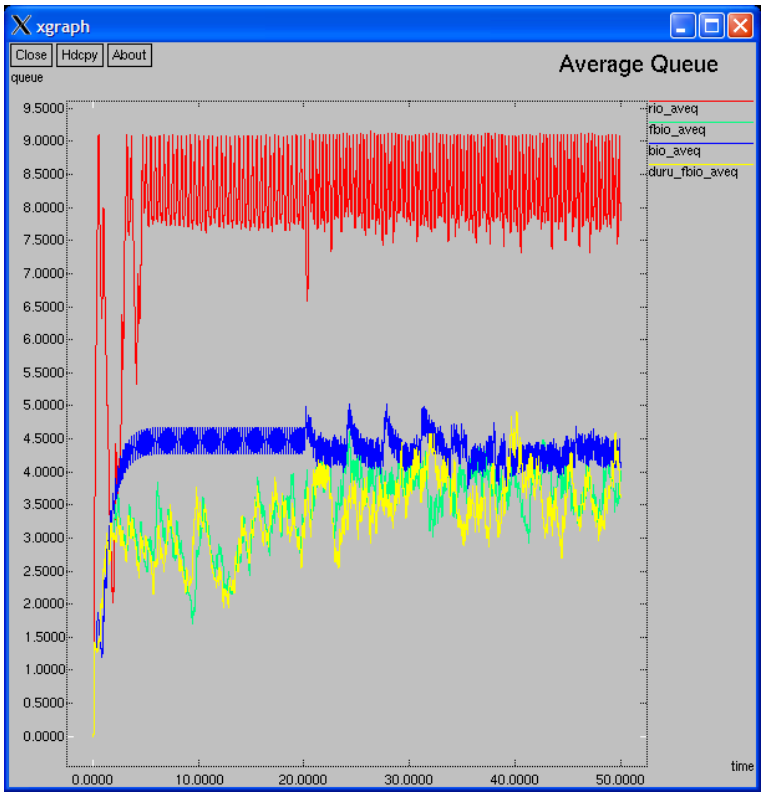


Şekil 5.9 Kullanılabilirlik (Utilization)

Hattın kullanılabilirlik parametresine bakıldığında ise paket kaybıyla orantılı olarak FBIO algoritmasının diğerlerine yakın olmakla birlikte daha iyi sonuç verdiği görülmektedir.

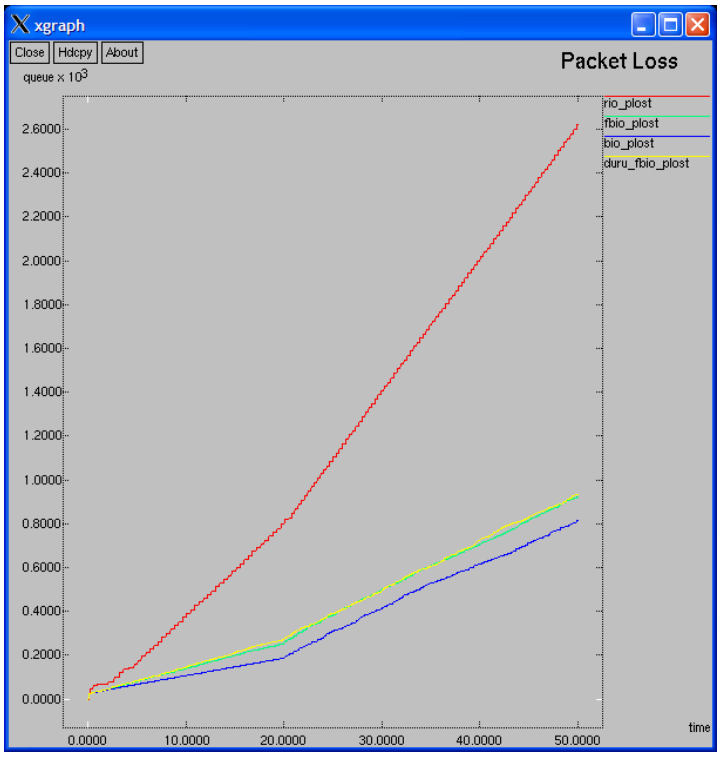
5.6.2 Test Senaryosu – 2

İkinci test senaryomuzda, ağ topolojisindeki tüm düğümler arasındaki hatların kapasiteleri 5 Mbps ve gecikmeleri 1ms olarak belirlenmiştir. Senaryo gereğince Pareto trafiği oluşturan n0 ve ftp trafik oluşturan n1 ile n2 düğümlerimiz mevcuttur. Bunların dışında hedef düğümlerimiz n5, n6, n7 yer almaktadır. Trafik oluşturan tüm düğümler 0.0 zamanında trafik yaratmaya başlarlar. Buffer boyutu olarak 20 paket verilmiştir.



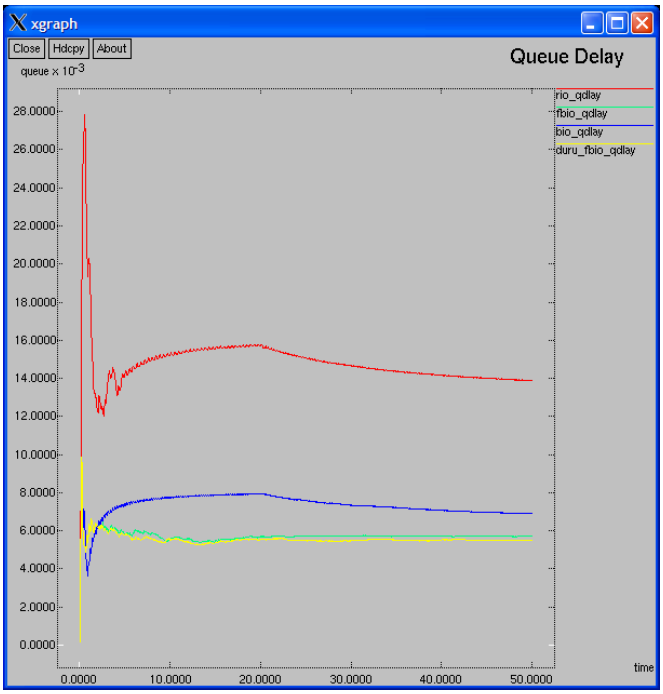
Şekil 5.10 Ortalama Kuyruk Boyutları (Average Queue Sizes)

Simülasyon sonucundaki ölçümlemelere göre Fuzzy BIO algoritmasının geleneksel BIO algoritmasına göre yine daha küçük kuyruk boyutuna ihtiyaç duyduğu ortaya çıkmıştır. Burada Fuzzy BIO'nun kuyruk boyunu yönetmedeki başarımı diğerlerinden ayırt edici bir üstünlük kurmaktadır. Bu sayede yönlendiricilerdeki buffer ihtiyacı daha aşağılara çekilebilinir. Bakıldığında FBIO da ortalama kuyruk 3.5 paket olurken, geleneksel BIO 4.5 ve RIO 8.5 paket olmaktadır. Duru halin ise hemen hemen aynı olduğu görülmektedir.



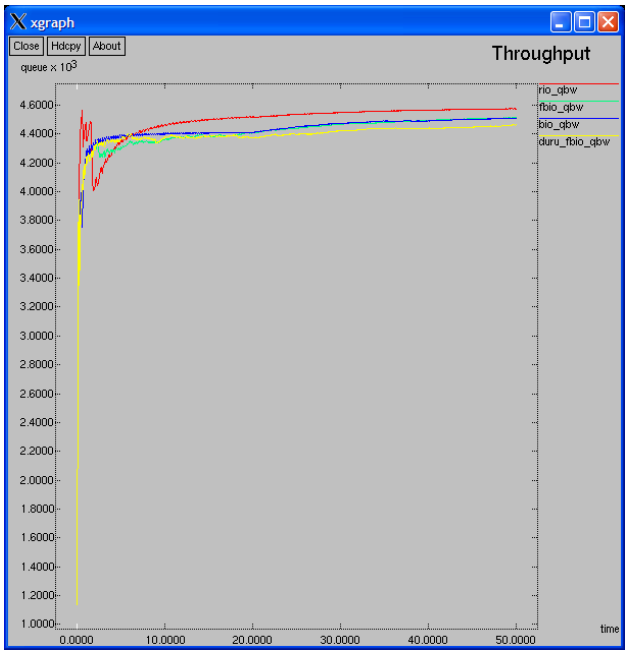
Şekil 5.11 Paket Kayıp Büyüklükleri (Packet Loss)

Simülasyon sonuçlarına göre RIO algoritması açık şekilde daha fazla paket düşürdüğü ve daha az güvenilir olduğu anlaşılmıştır. Burada yine Fuzzy BIO ile geleneksel BIO yapısının diğer yapılara bakış daha az paket kaybı yaptığını görmekteyiz.



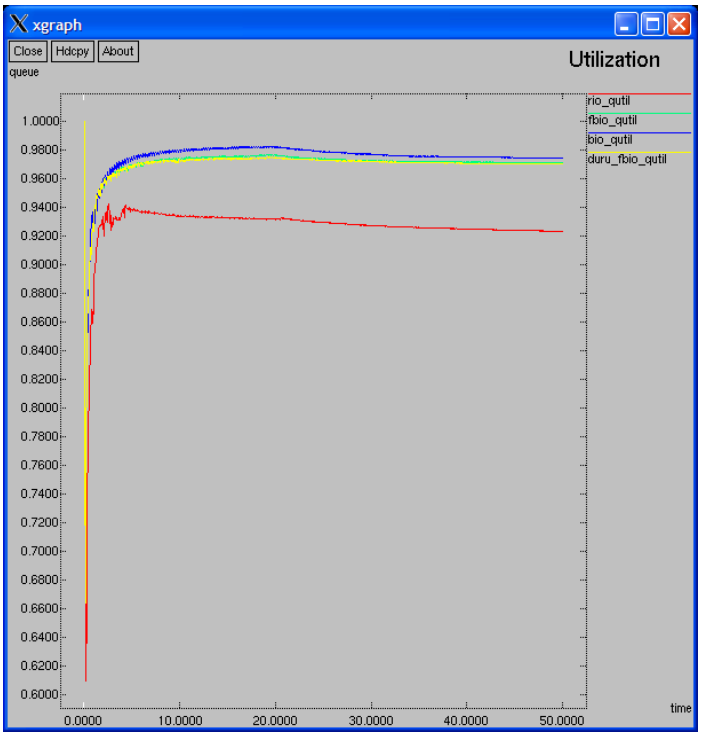
Şekil 5.12 Gecikme Süreleri (Queue Delay)

Gecikme sürelerine bakıldığında RIO ve BIO arasında yine oldukça fark oluştuğu görülmektedir. Fuzzy BIO yaklaşık 6 ms gecikirken BIO 8 ms, RIO başlangıçta daha fazla olmasına rağmen sonradan toparlanmakta ve 14 ms gecikmektedir. Bu durumda yine Fuzzy BIO yapısının daha az gecikme yarattığını görmekteyiz. Duru halin ise biraz daha iyi sonuç verdiği gözlenmektedir.



Şekil 5.13 Hattın Verimi (Throughput)

Hattın verimliliğini belirlemek amacıyla yapılan bu ölçümlemede her üç aktif kuyruk yapısının da birbirlerine çok yakın oldukları belirlenmiştir. Hızları yaklaşık 4.4Mbps olmaktadır.



Şekil 5.14 Kullanılabilirlik (Utilization)

Kullanılabilirlik metriğine baktığımızda yine RIO'nun diğerlerinden daha kötü sonuçlar verdiği açıkça görülmektedir. Fakat diğer iki algoritma yine birbirlerine çok yakın değerlerde seyretmektedirler. Kullanılabilirlik parametresi Fuzzy BIO ve geleneksel BIO için yaklaşık %98 olurken, RIO için %92 olmaktadır.

6. SONUÇ

Gün geçtikçe artan yeni Internet servisleri ve bunların doğrultusunda ortaya çıkan servis kalitesi ihtiyaçlarına bakıldığında günümüzdeki kuyruk denetim mekanizmaları yeterli olamamaktadırlar. Bu nedenle geri beslemeli kontrol sistemleri için oldukça başarılı olan Fuzzy lojik yapısını yeni kuyruk tasarımında uyguladık. Sonuçlar karşılaştırıldığında FBIO tasarımının geleneksel BIO ve RIO kuyruk yapılarına oranla daha iyi sonuçlar verdiği görülmektedir. Özellik kuyruk boyutu ve kuyruk gecikmesindeki sonuçlar yapının genel başarısı ile ilgili bilgi vermektedir. FBIO tasarımı diğer kuyruk yönetim yapılarına özellikle geleneksel BIO karşısında daha az paket kaybı, daha az gecikme, daha az kuyruk boyutu ve yüksek verim göstermektedir.

Yapılan tez çalışmasında aktif kuyruk denetim yapıları detaylı incelenerek yapılan benzetimlerle performansları incelenmiştir. Bu incelemeler sonucunda özellikle kolay uygulanabilirliği, az karmaşık yapısı ve performansı nedeniyle BLUE algoritmasının ve onun farklılaştırılmış hizmetler sunabilmek amacıyla geliştirilmiş hali olan BIO algoritmasının daha iyi performans verebilmesi amacıyla geliştirilmesi düşünülmüştür. Bu doğrultuda da gün geçtikçe daha fazla kullanılan fuzzy lojik kontrol yapısının BIO algoritmasına entegre edilmesine karar verildi. Benzetimlerde ve birçok uygulamada kullanılan ve USC/ISI, XEROX PARC, LBLN, UCB kurumlarından oluşan bir grup tarafından ortaklaşa olarak C++ ve OTCL dillerinde geliştirilen ve kaynak kod olarak dağıtılan NS ağ benzetim aracı kullanılmıştır.

Oluşturulan aktif kuyruk mekanizması, kurulan ağ topolojisi üzerinde belirlenen senaryolar uyarınca test edilmiş, yapılan benzetim ve analizler sonucunda aşağıdaki sonuçlara varılmıştır.

1. Fuzzy lojik yapısı geleneksel yapıya kıyasla daha esnek bir kontrol sağlamaktadır.
2. Geleneksel yöntemin aksine fuzzy lojik yapısında parametre seçimi daha kolay olmaktadır.
3. Analiz sonuçlarına bakıldığında fuzzy lojik yapısı paket düşürme, kuyruk boyu, verim ve gecikme parametrelerine bakıldığında daha iyi sonuçlar vermektedir.
4. Trafiğin yoğun biçimde profil dışı paketlerden oluştuğu durumlarda fuzzy'nin üstünlüğü beklendiği kadar fark yaratmamaktadır.
5. Akışlara adil şekilde davranma konusunda hala istenilen performans

sağlanamamaktadır.

Bu çalışma sonucunda fuzzy yapısının geleneksel yapıya oranla başarımının çoğu noktada daha esnek ve yüksek olduğu görülmüştür. Paket kaybı bakımından %7 - %15 arasında bir kazanç elde edilmekte, kuyruk boyu olarak %20-%22 civarında daha kısa kuyruk ihtiyacı oluşmaktadır. Hattın verimi bakımından da geleneksel yapı ile fuzzy lojik yapı arasındaki değerler birbirlerine çok yakın olmaktadırlar. Paket gecikmesi açısından ise %20-%25 civarında daha az bir gecikme süresi olmaktadır. Paket kayıp oranının da kullanılabilirlik parametresiyle orantılı olduğu görülmektedir. Yapılacak yeni çalışmalarda farklı trafik yapıları altında bu sistemin başarımı denenerek parametrelerin geliştirilmesi ve fuzzy yapısının daha iyi performans için ayarlanması sağlanabilir. Ayrıca daha esnek fuzzy kontrolleri, genetik algoritmalarla veya neuro-fuzzy yapıları kullanarak geliştirilen tasarımlar yapılarak başarımları incelenebilir.

KAYNAKLAR

Zhi Li, Zhongwei Zhang, ve Ron Addie, “A Circumspect Active Queue Management Scheme Based on Fuzzy Logic”, International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA), 2004.

W. Feng, ”Stochastic fair blue: a queue management algorithm for enforcing fairness”, IEEE INFOCOM, Nisan 2001.

M. H. Yaghmaee, “A modified random early detection algorithm: fuzzy logic based approach”, JCN, Journal of Communication Network, Eylül 2005.

S.Floyd, R.Gummadi, S.Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED", Technical Report, 2001

W.Feng, D.Kandlur, D.Saha, K.Shin, "BLUE: A New Class of Active Queue Management Algorithms", U.Michigan CSE-TR-387-99, Nisan 1999

C. Chrysostomou, A. Pitsillides, G. Hadjipollas, A. Sekercioglu ve M. Polycarpou, "Fuzzy Logic Congestion Control in TCP/IP Best-Effort Networks" , ATNAC'03, Aralık 2003

C. Chrysostomou1, A. Pitsillides, L. Rossides, M. Polycarpou, A. Sekercioglu, "Congestion Control in Differentiated Services Networks using Fuzzy-RED", IFAC, Kasım 2003

M.H. Yaghmaee,Hale Amintoosi, "A Fuzzy Based Active Queue Management Algorithm", SPECTS2003, Temmuz 2003

Ming Jiang; Jian Wu; Chunming Wu, “MBIO: An New Active Queue Management Algorithm for DiffServ Network” Networking, Sensing and Control, 2006. ICNSC’06. IEEE International Conference, 2006

C.Y.Zhu ve O.Yang. “A Comparison of Active Queue Management Algorithms Using the OPNET Modeler.“, IEEE Communications Magazine, 2002

Okuroglu B., Oktug S., “BIO Revisited,” Proc. of IEEE 11th International Symposium on Software, Telecommunications and Computer Networks (SoftCOM 2003), Kasım 2003.

S. Tagshavi Zargar; M. H. Yaghmaee; A. Milani Fard, “Fuzzy Proactive Queue Management Technique”, Annual India Conference, 2006

Ming Jiang; Qin Chen,” TLA: A Traffic Load Adaptive Congestion Control Algorithm for TCP/AQM Networks”, Intelligent Systems Design and Applications, 2006.

Wei-yan Liu, Shun-yi Zhang, Mu Zhang, Tao Liu, "A DT-based Adaptive BLUE Algorithm,", Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), 2006

Ns-2. Network Simulator. http://www.isi.edu/nsnam/ns.

INTERNET KAYNAKLARI

[1] Cisco Documentation, Quality of Service (QoS) Networking, http://www.cisco.com/en/US/docs/internetworking/technology/handbook/QoS.html

[2] Ns-2. Network Simulator. http://www.isi.edu/nsnam/ns.

[3] Integrated Services Charter, http://www.ietf.org/html.charters/intservcharter.html.

[4] Qualiy of Service http://www.networkerfactory.net/index.php/category/qos/

[5] Wikipedia, http://en.wikipedia.org/

[6] Introduction to ns-2 http://www.cs.odu.edu/~mweigle/research/netsim/intro/

[7] LBNL Network Simulator http://www-nrg.ee.lbl.gov/ns/

[8] BLUE AQM http://www.thefengs.com/wuchang/blue/

[9] Running_Ns_and_Nam_Under_Windows_9x/2000/XP_Using_Cygwin

http://nsnam.isi.edu/nsnam/index.php/Running_Ns_and_Nam_Under_Windows_9x/2000/XP_Using_Cygwin

[10] NS2 Network Simulator Under Windows http://www.usq.edu.au/users/leis/notes/software/ns2win.html

[11] Getting Started With ns2 http://evanjones.ca/ns2.html

[12] NS Simulator Course for Beginners

http://www-sop.inria.fr/mistral/personnel/Eitan.Altman/ns.htm

[13] Ns by Example http://nile.wpi.edu/NS/

EKLER

Ek 1 ns-default.tcl

```
# ------------------------------------------------
# Fuzzy BIO (Blue IN & OUT) AQM kuyruk algoritması
# ------------------------------------------------

Queue/FBIO set ptc_ 25.0
Queue/FBIO set prob_ 0.0
Queue/FBIO set in_prob_ 0.0
Queue/FBIO set out_prob_ 0.0
Queue/FBIO set thresh_ 10
Queue/FBIO set maxthresh_ 30
Queue/FBIO set in_thresh_ 250
Queue/FBIO set in_maxthresh_ 500
Queue/FBIO set out_thresh_ 150
Queue/FBIO set out_maxthresh_ 300
Queue/FBIO set q_weight_ 0.002
Queue/FBIO set linterm_ 10
Queue/FBIO set in_linterm_ 10
Queue/FBIO set bytes_ false
Queue/FBIO set queue_in_bytes_ false
Queue/FBIO set mean_pktsize_ 1000
Queue/FBIO set wait_ true
Queue/FBIO set setbit_ false
Queue/FBIO set gentle_ false
Queue/FBIO set drop_tail_ true
Queue/FBIO set drop_front_ false
Queue/FBIO set drop_rand_ false
Queue/FBIO set doubleq_ false
Queue/FBIO set ns1_compat_ false
Queue/FBIO set dqthresh_ 50
Queue/FBIO set ave_ 0.0
Queue/FBIO set curq_ 0
Queue/FBIO set totplost_ 0
Queue/FBIO set in_curq_ 0
Queue/FBIO set which_ 0
Queue/FBIO set decrement_ 0.002
Queue/FBIO set increment_ 0.02
Queue/FBIO set decrement_in_ 0.001
Queue/FBIO set increment_in_ 0.002
Queue/FBIO set decrement_out_ 0.002
Queue/FBIO set increment_out_ 0.01
Queue/FBIO set ihold-time_ 0.1
Queue/FBIO set dhold-time_ 0.1
Queue/FBIO set bbw_ 100000

Queue/FBIO set fbio_step_number 10

# ------------------------------------------------
# ------------------------------------------------
```

Ek 2 test-1.tcl

```
# try FBIO
# Initialize simulation
set ns [new Simulator]

# Create trace file
set nf [open out.nam w]
set tf [open fbio.tr w]
$ns namtrace-all $nf
$ns trace-all $tf

# Set analyz files
set qsize [open qsize_fbio.tr w]
set qbw [open qbw_fbio.tr w]
set qlost [open qlost_fbio.tr w]
set qdlay [open qdlay_fbio.tr w]
set qutil [open qutil_fbio.tr w]

# Define 8 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

# Topology
#   Pareto                                      UDP sink
#          n0                                  n5
#           \                                 /
#            \                               /
#             \                             /
# FTP/TCP n1 --n3 ----------------------n4 --- n6 (TCP Sink)
#             /                           \
#            /                             \
#           /                               \
#         n2 (Ftp/TCP)                       n7 (TCP Sink)
#

#set FBIO
#out packets are droped before any in packets
#flow with flowid_ 0 has prrority over other flows
#Queue/FBIO set debug_ true
Queue/FBIO set in_thresh_ 10
Queue/FBIO set in_maxthresh_ 20
Queue/FBIO set out_thresh_ 3
Queue/FBIO set out_maxthresh_ 9
Queue/FBIO set in_linterm_ 50
Queue/FBIO set linterm_ 200
Queue/FBIO set mean_pktsize_ 500


# Create the topology
$ns duplex-link $n0 $n3 1Mb 2ms DropTail
$ns duplex-link $n1 $n3 1Mb 2ms DropTail
$ns duplex-link $n2 $n3 1Mb 2ms DropTail

$ns duplex-link $n3 $n4  1Mb 2ms FBIO
```

```
$ns duplex-link $n4 $n5 1Mb 2ms DropTail
$ns duplex-link $n4 $n6 1Mb 2ms DropTail
$ns duplex-link $n4 $n7 1Mb 2ms DropTail

#--------------------------------------#
# QUEUE MONITOR                        #
#--------------------------------------#
$ns duplex-link-op $n3 $n4 queuePos 0.5
set qf_size [open queue.size w]
set qmon_size [$ns monitor-queue $n3 $n4 $qf_size 0.05]

set samples [new Samples]
$qmon_size set-delay-samples $samples

#set bytesInt_ [$qmon_size get-bytes-integrator]
#set pktsInt_ [$qmon_size get-pkts-integrator]

# Specify the queue limit
$ns queue-limit $n3 $n4 20

# Configure FBIO queue parameters here
set fbioq [[$ns link $n3 $n4] queue]

# Tracing a queue
set tchan_ [open all.q w]
$fbioq trace curq_
$fbioq trace ave_
$fbioq trace totplost_
$fbioq attach $tchan_


# Define the Pareto source at n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set p [new Application/Traffic/Pareto]
$p set packetSize_ 512
$p set burst_time_ 20ms
$p set idle_time_  80ms
$p set rate_       4M
$p attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n5 $null0
$ns connect $udp0 $null0
$ns at 0.0 "$p start"


# Define TCP Source at n1
set tcp1 [new Agent/TCP]
#$tcp1 set fid_ 1
$tcp1 set window_ 50
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set tcp_sink1 [new Agent/TCPSink]
$ns attach-agent $n6 $tcp_sink1
$ns connect $tcp1 $tcp_sink1
$ns at 0.0 "$ftp1 start"

# make token bucket limiter for flow 0
# Fill rate 100000 Bps, or 100 packets per second.
```

```
set link1 [$ns link $n1 $n3]
set tcm1 [$ns maketbtagger Fid]
$ns attach-tagger $link1 $tcm1
set fcl1 [$tcm1 classifier]; # flow classifier
$fcl1 set-flowrate 0 100000 10000 1
#target_rate_ (fill rate, in Bps),
#bucket_depth_,
#tbucket_ (current bucket size, in bytes)


# Define TCP source 2 at n2
set tcp2 [new Agent/TCP]
#$tcp2 set fid_ 2
$tcp2 set window_ 50
$ns attach-agent $n2 $tcp2
set tcp_sink2 [new Agent/TCPSink]
$ns attach-agent $n7 $tcp_sink2
$ns connect $tcp2 $tcp_sink2
set ftp2 [$tcp2 attach-source FTP]
$ns at 0.0 "$ftp2 start"

# make token bucket limiter for flow 1
# Fill rate 100000 Bps, or 1000 packets per second.
set link2 [$ns link $n2 $n3]
set tcm2 [$ns maketbtagger Fid]
$ns attach-tagger $link2 $tcm2
set fcl2 [$tcm2 classifier]; # flow classifier
$fcl2 set-flowrate 0 100000 10000 1

$ns at 0.0 "record"
$ns at 50.0 "finish"

# Procedure finish
proc finish {} {
   global ns tf nf tchan_ qsize qbw qlost qdlay qutil qmon_size
   $ns flush-trace
   close $nf
   close $tf
   close $qsize
   close $qbw
   close $qlost
   #exec nam out.nam &
   set awkCode {
          {
            if ($1 == "Q" && NF>2) {
            print $2, $3 >> "fbiotemp.q";
            set end $2
            }
            else if ($1 == "a" && NF>2)
            print $2, $3 >> "fbiotemp.a";
            else if ($1 == "t" && NF>2)
            print $2, $3 >> "fbiotemp.t";

          }
        }
    set f [open fbiotemp.queue w]
    puts $f "TitleText: FBIO"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
    close $tchan_
        }
```

```
        exec rm -f fbiotemp.q fbiotemp.a fbiotemp.t
        exec touch fbiotemp.a fbiotemp.q fbiotemp.t

        exec awk $awkCode all.q

        puts $f \"queue
        exec cat fbiotemp.q >@ $f
        #puts $f \n\"ave_queue
        #exec cat fbiotemp.a >@ $f
        #puts $f \n\"tot_plost
        #exec cat fbiotemp.t >@ $f
        close $f
        exec xgraph -bb -tk -x time -y queue fbiotemp.queue &

        exec xgraph qsize_fbio.tr  -t "Queuesize" &
        exec xgraph qbw_fbio.tr  -t "Throughput" &
        exec xgraph qlost_fbio.tr  -t "Lost" &
        exec xgraph qdlay_fbio.tr  -t "Queue Delay" &
        exec xgraph qutil_fbio.tr  -t "Utilization" &



        puts "Link statistics for link n1->n2:"
        puts "parrivals_=[$qmon_size set parrivals_], pdrops_=[$qmon_size
set pdrops_]"
        puts "pdepartures_=[$qmon_size set pdepartures_]"
        puts "barrivals_=[$qmon_size set barrivals_], bdrops_=[$qmon_size
set bdrops_]"
        puts "bdepartures_=[$qmon_size set bdepartures_]"


   exit 0
}

# Record tanımı her kayıtta yapılacak isler

set old_departure 0

proc record {} {
      global ns qmon_size qmon_bw qmon_lost qsize qbw qlost qdlay qutil
old_departure
      set ns [Simulator instance]
      set time 0.05
      set now [$ns now]

      $qmon_size instvar size_ pkts_ barrivals_ bdepartures_ parrivals_
pdepartures_ bdrops_ pdrops_ bytesInt_ pktsInt_
      puts $qsize "$now [$qmon_size set pkts_]"
      #if { $now !=0 } { puts $qsize "$now [expr [$qmon_size set
size_]/$now]" }
      #puts $qsize "$now [expr $parrivals_-$pdepartures_-$pdrops_]"

      #-----------------------------------------------
      set pktint [$qmon_size get-pkts-integrator]
      #puts $qsize "$now [$pktint set sum_]"
      #if { $now !=0 } { puts $qsize "$now [expr [$pktint set sum_]/$now]"
}

      set delaySamples [$qmon_size get-delay-samples]
      if { $now !=0 } {puts $qdlay "$now [$delaySamples mean]"}
      #-----------------------------------------------
```

```
      #puts $qbw "$now [expr ($bdepartures_ - $old_departure)*8/$time]"
      #set old_departure $bdepartures_
      #puts $qbw "$now [expr $bdepartures_*8/1024/$time]"

      if { $now !=0 } { puts $qbw "$now [expr $bdepartures_*8/1024/$now]" }
      #puts $qbw "$now [expr $bdepartures_*8/$time]  $bdepartures_"
      #set bdepartures_ 0

      if { $now !=0 } { puts $qutil "$now [expr
($bdepartures_*8/1024/$now)/($barrivals_*8/1024/$now)]" }

      #puts $qlost "$now $pdrops_  $bdrops_"
      puts $qlost "$now $pdrops_"
      #puts $qlost "$now [$qmon_size set pdrops_]"

      $ns at [expr $now+$time] "record"
}

$ns run
```

ÖZGEÇMİŞ

Doğum tarihi	20.12.1982	
Doğum yeri	Kırcaali / Bulgaristan	
Lise	1994-2001	Vefa Anadolu Lisesi
Lisans	2001-2005	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fak. Bilgisayar Mühendisliği Bölümü

Çalıştığı kurumlar

2005-2006	Novilink Bilgisayar Yazılım LTD.ŞTİ.
2006-Devam ediyor	Garanti Teknoloji A.Ş.