SAÜ. Fen Bil. Der. 17. Cilt, 3. Sayı, s. 437-446, 2013 SAU J. Sci. Vol 17, No 3, p. 437-446, 2013

Ns-3 ağ simülatöründe, kuyruk yönetim algoritmalarının performans analizi

Ünal Çavuşoğlu[1*] , M. Muhammed Öztürk[1] , Uğur Özbek[2] , Ahmet Zengin[1]

[1*] Sakarya Üniversitesi, Teknoloji Fakültesi, Bilgisayar Mühendisliği, Sakarya
[2] Sakarya Üniversitesi, Uzaktan Eğitim Merkezi, Sakarya

25.12.2012 Geliş/Received, 22.08.2013 Kabul/Accepted

ÖZET

Bu makalede bilgisayar ağlarında trafik akışının düzenlenmesinde önemli bir yere sahip olan kuyruk yönetimi konusu ele alınmıştır. Günümüzde yaygın olarak kullanılan kuyruk yönetim algoritmaları tanıtılarak kısaca çalışma prensipleri anlatılmıştır. Kuyruk yönetim algoritmalarının başarım karşılaştırması için NS-3 ağ simülatör programı kullanılmıştır. NS-3 ağ simülatörü üzerinde RED ve DROP TAIL kuyruk yönetim algoritmaları çeşitli topolojiler üzerinde test edilmiş ve performans değerlendirmesi yapılmıştır. Ayrıca elde edilen sonuçlar kuyruk yönetimi konusunda daha önce yapılmış olan çalışmalar ile karşılaştırılmıştır. Yapılan uygulamalar sonucu elde edilen veriler ve yapılmış olan çalışmalar değerlendirildiğinde, olasılıksal yöntemler ve eşik değerlerini kullanan RED algoritmasının DROP TAIL kuyruk algoritmasından daha başarılı olduğu tespit edilmiştir.

Anahtar Kelimeler: NS-3, RED, DROP TAIL, Kuyruk yönetim algoritmaları, bilgisayar ağ trafiği yönetimi

Performance analysis of queue management algorithms in Ns-3 network simulator

ABSTRACT

In this article, which has an important role in the regulation of the flow of traffic in computer networks is discussed queue management. Queue management algorithms are widely used today introduced the working principles are described briefly. NS-3 network simulator program was used for queue management algorithms comparison of performance. On NS-3 network simulator, DROP TAIL and RED queue management algorithms have been tested on various topologies and performance evaluations were made. Furthermore, the studies are compared with results obtained previously. The data obtained from these tests and studies those evaluated, RED algorithms that use probabilistic methods and threshold values have been found to be more successful.

Keywords: NS-3, RED, DROP TAIL, quque management algorithms, computer network traffic management

[*] Sorumlu Yazar / Corresponding Author

1. GİRİŞ (INTRODUCTION)

İnternet üzerinde yüksek düzeyli paket kayıp oranından kaçınmak önemlidir [1]. Bir paket kaynaktan hedefe ulaşmadan kaybolursa iletim için kullanılan kaynaklar boşa harcanmış olur. Bu kaynak kayıp düzeylerinin çok yüksek olması tıkanıklık çöküşlerine neden olur. Tıkanıklık ve paket kayıp sorunlarına çözüm olarak kuyruklama algoritmaları geliştirilmiştir. Bilgisayar ağlarında kuyruk yönetimi konusu oldukça önemli yer tutmakta, doğru algoritmanın seçimi bütün bir ağın başarımına doğrudan etki edebilmektedir [2]. Aktif kuyruk yönetim algoritmaları kuyruk üzerindeki işlemleri gerçekleştirirken, kuyruk uzunluğuna bakarak algılama gerçekleştirmektedirler. Aktif kuyruk yönetiminin amacı yönlendirici üzerindeki ortalama kuyruk uzunluğunu azaltmaktır [3]. Böylece paketlerdeki uçtan uca gecikmeler azaltılmış olur [4] ve ağ kaynakları verimli kullanılır. Kuyrukta oluşan birikme sistemler veya uç noktalar arasında tıkanıklığa yol açmadan önce bazı paketler düşürülerek tıkanıklığın önüne geçilmeye çalışılmaktadır. Kullanılan aktif kuyruk yönetim algoritmasının yapısına göre uç sistemler üzerinde tıkanıklığı engelleyecek tedbirler alınmasına yardımcı olunmaktadır. Aktif kuyruk yönetim algoritmalarının tasarımında bağlantı tıkanıklıklarının derecesinin belirlenmesi ve paket düşüşlerinin analiz, göz önüne alınması gereken konulardır [5].

Ağların deneysel olarak gözlemlenmesinin en kolay ve etkili yollarından biri, çok farklı alanlarda da kullanılan simülasyon yöntemidir. Bu yöntem ile ağdaki düğümler, bağlantılar ve ağ trafiği gerçek dünyaya benzer şekilde tasarlanarak farklı durumlar kolaylıkla denenebilmektedir. Gerçekte elde edilmesi maddi veya diğer kısıtlarla imkansız olabilecek durumlar da ağ simülatörlerinde kolaylıkla gözlenebilir. Özellikle donanım olarak bulunan düğümler üzerindeki cihazların içerisindeki kuyruk yönetim algoritmalarını değiştirerek deneyler yapmak çok maliyetli ve zahmetli olabileceğinden bu durumlar ağ simülatörleri ile test edilebilir. NS-3 ağ araştırmaları için altyapı sağlayan açık kaynaklı bir ağ simülatörüdür. Bu simülatör ile çeşitli kuyruk yönetim algoritmaları modellenebilmekte ve gerekli deneyler yapılabilmektedir [6].

Günümüzde kullanılan birçok kuyruk yönetim mekanizması vardır. Genelde hangi kuyruklamanın seçileceği sistem yöneticisi tarafından belirlenir, ancak yapılan çalışmalar bazı kuyruk yöntemlerinin diğerlerinden daha iyi olduğunu ortaya konmaktadır. Bu üstünlük belli servis sınıfları için iyi olabilirken bazıları için iyi bir yöntem de oluşturmayabilir. Ağ üzerindeki uygulamalara göre karar verilmelidir. Bu çalışmada NS-3 içerisinde bulunan DROP TAIL ve RED algoritmalarının özellikle paket kaybı değerleri üzerinden başarımları analiz edilecektir.

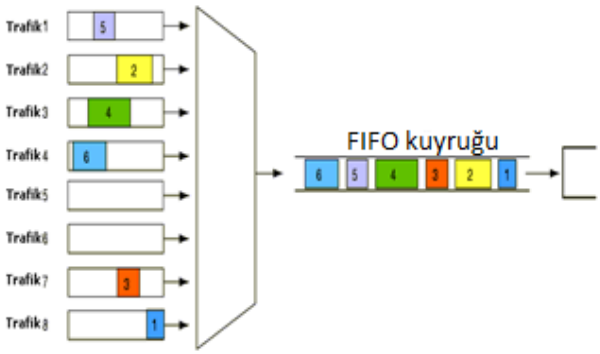
Makalede, giriş bölümünün ardından, 2. Bölümde kuyruk yönetim algoritmaları hakkında bilgi verilmiş, 3. Bölümde ağ simülasyon teknolojileri ve NS-3 ağ simülatörü tanıtılarak, NS-3 ağ simülatöründeki kuyruk yapısı açıklanmış, 4. Bölümde NS-3 ağ simülatörü üzerinde RED ve DROP TAIL kuyruk algoritmaları ile ilgili testler gerçekleştirilmiş, son bölümde ise yapılan testler sonucu kuyruk yönetim algoritmalarının karşılatırılması yapılmıştır.

2. KUYRUK YÖNETİM ALGORİTMALARI (QUEUE MANAGEMENT ALGORITHMS)

Günümüzde bilgisayar ağları üzerinde artan trafik yoğunluğunu rahatlatmak ve tıkanıklık sorunlarını azaltmak için birçok kuyruk algoritması geliştirilmiştir. Ses ve gerçek zamanlı uygulamaların bulunduğu ağlarda en yaygın olarak kullanılan kuyruk algoritmaları geleneksel ve düşük gecikmeli yöntemlerdir. Geleneksel kuyruk yönetim algoritmaları tüm paketlerde yer alan öncelik alanları ile paketleri sınıflara ayrılmış olan kuyruklara atar. Her kuyruk kendine atanan servis kadar önceliklidir. Dolayısıyla bu kuyrukta paket varsa her zaman öncelikli gönderilmeye hazırdır. Ayrıca bu kuyruk için sabit bir bant genişliği atanabilmektedir. Az gecikmeli kuyruk yönetim algoritmalarının en belirgin farkı ise en yüksek öncelikli paketi kuyruğa sokmadan direkt olarak gideceği yere iletilmesi için ağ cihazının iletim portuna yönlendirmesidir. Böylece ağ içinde en kritik uygulamada gecikmelerden dolayı oluşabilecek sorunların önüne geçilmeye çalışılmıştır. Diğer trafikler ise yine sınıflarına ve önceliklerine ayrılarak kuyruklandırılırlar .

2.1. İlk Gelen İlk Çıkar Algoritması (First In First Out Algorithm)

Bu yöntem ağ cihazlarının geleneksel olarak kullandığı bir yöntemdir. Ağ cihazlarına gelen paketler tek bir kuyruğa konulur ve kuyruğa daha önce gelen paket daha önce hizmet alır. Bu kuyruklama yöntemi gecikmeye duyarlı hizmetlerin iletilmesi için uygun değildir. Küçük boyutlu paketler büyük paketlerin ardında kalarak yüksek gecikme değerleriyle iletilirler. Ayrıca bu tek kuyruğu kullanan trafik kaynaklarından birisi daha yüksek iletim hızlarında paket yolladığında diğerlerinin aleyhinde olacak şekilde kuyruğu doldurabilir.



Şekil 1.İlk gelen ilk çıkar yöntemi (First in first out method) [7]

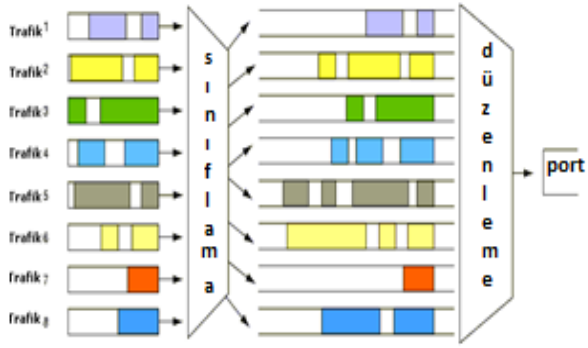
Şekil 1'de farklı sistemler tarafından gerçekleşen trafik akış diagramı üzerindeki gelen paketlerin işleme alınması işlemi gerçekleştirilmektedir. Gelen trafik hatları üzerinde paketler geliş önceliğine göre işleme alınmaktadır.

2.2. Adil Kuyruklama Algoritması (Fair Queueing Algorithm)

FIFO algoritmasında trafik kaynaklarındaki paketlerin işleme alınması için tek bir kuyruk yapısı bulunması işlemlerin sadece tek bir kuyruğa bağlı olarak yürütülmesi zorunluluğunu getirmektedir. Bu zorunluluğun giderilmesi için tek bir kuyruk yapısı yerine birçok kuyruk kullanılarak adil kuyruklama yapısı[8] John Nagle tarafından ortaya atılmıştır. Nagle'ın algoritması FIFO için basit bir çözüm getirmiştir. Algoritma pseudo kodu aşağıda görülmektedir.

1. Yeni bir veri varsa gönder
2. Pencere büyüklüğü >= maksimum bölüm büyüklüğü ve veri >maksimum bölüm büyüklüğü ise maksimum bölümün gönderimini tamamla.
3. Değilse ve kuyrukta onaylanmamış veri varsa kabul gelene kadar verileri kuyrukla.
4. Değilse verileri gönder.

Bu yapıda her bir trafik kaynağı için bir kuyruk yapısı kullanılması öne sürülmüştür. Kuyruk sayısı trafik kaynakları kadar olabileceği gibi daha az sayıda da olabilmektedir. Bu algoritma her bir trafik kaynağından sırasıyla paket çekilerek kuyruktaki paketlerin azaltılması esasına göre çalışmaktadır. Fakat adil kuyruklama yönteminin pratikte kullanımı sırasında sıkıntılar ortaya çıkmaktadır. Kuyruk sayısı trafik kaynaklarından az olduğu durumlarda trafik kaynakları arasında kuyruklardaki işlemleri gerçekleştirme için olasılık hesaplamaları kullanılarak dağıtım gerçekleştirilmektedir [9] .

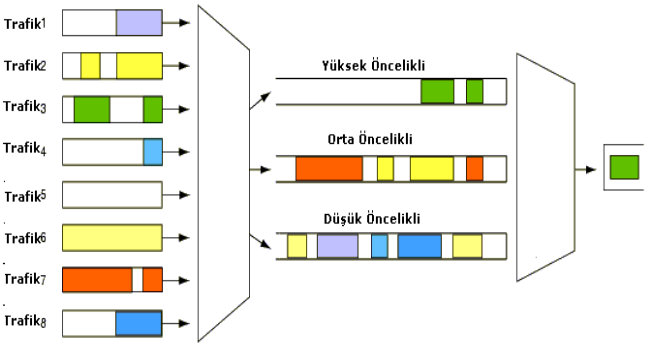


Şekil 2.Olasılıksal adil kuyruklama (Stochastic Fairness Queueing) (SFQ) [10]

Şekil 2'de gelen trafik hatları üzerinde gerçekleşen kuyruk yapılanması görülmektedir. Şekil 2'deki örnekte hat sayısı kadar kuyruk mevcuttur. Paketler kuyruk sayısı baz alınarak ilgili iş kuyruğuna sırası ile alınarak trafik kaynakları üzerinde işlem görmesi sağlanmaktadır.

2.3. Ağırlıklı Adil Kuyruklama Algoritması (Weighted Fair Queueing Algorithm)

Ağırlıklı adil kuyruklama algoritması olasılıksal adil kuyruklama algoritmasına ağırlık ve paket boyu algılama kavramlarının eklenmesiyle meydana gelen bir kuyruklama algoritmasıdır. Bu kuyruklama algoritmasının çalışma mantığı incelendiğinde, sistemde birden fazla kuyruğun bulunduğu görülmektedir. Kuyruklara kendi aralarında öncelikler ve ağırlıklar verilerek sistemin işleyişi sağlanmaktadır. Paketlerin kuyruğu terk etme zamanları göz önüne alınarak gerektiğinde adil kuyruklamanın sıralı mantığı terk edilir. Böylece kuyruğa gelen uzun bir paketin ardındaki kısa paketin iletim zamanını etkilemesinin önüne geçilmeye çalışılır. Gerek Sınıf Tabanlı Kuyruklama, gerekse de Ağırlıklı Adil Kuyruklama birer Öncelikli kuyruklama yöntemidir.



Şekil 3.Oncelikli kuyruklama (Priority Queuing) [11]

Şekil 3'te trafik kaynakları üzerindeki paketlerin, sistemde öncelik sırası esasına göre yapılandırılmış olan üç adet kuyruk üzerindeki paket ilerleyişleri görülmektedir. Yüksek önceliğe sahip olan kuyrukta bulunan paketler işlemini tamamladıktan sonra sırasıyla diğer önceliğe sahip kuyruktaki paketlerin iletimi gerçekleştirilecektir.

2.4. Kuyruk Düşürme Algoritması (DROP TAIL Algorithm)

Yönlendirici kuyruk uzunluklarını yönetmenin geleneksel tekniği, her kuyruk için maksimum uzunluğunu ayarlayarak, kuyruk maksimum uzunluğa erişinceye kadar kuyruğa gelen paketleri dahil ederek bundan sonra gelen paketleri reddeder. Kuyruk üzerinden bir iletim gerçekleştiğinde kuyrukta yer açıldığından dolayı gelen paketler kuyruğun sonuna eklenir. Bu yöntem "DROP TAIL" olarak bilinir. Çok iyi bilinen ve kullanılan bir algoritma olmasının yanında, iki önemli sakıncası vardır[12]. Bu sakıncalardan ilki kuyruğun bir veya birkaç bağlantı tarafından kontrol altına alınması ve diğer bağlantıların yapılmasının engellenmesidir. Kuyruğu etkisi altına alan bu bağlantılar başka bir iletimin gerçekleşmesini engellerler ve diğer uygulamalar için senkronizasyon ve zaman aşımı gibi sonuçlar ortaya çıkarırlar. Ortaya çıkan diğer bir problem ise kuyruk yapısı dolduğunda kuyruğun dolu olduğunun terminallere iletimi için gönderilen kuyruk dolu sinyalinin iletimidir. Bu sinyalin en önemli etkisi düzgün iletildiği durumda kuyruk boyutunu azaltmasıdır.

Kısaca DROP TAIL'de etkin olarak yönetim yoktur. Ağ sistemlerinin kullanımı arttıkça ağ cihazları üzerindeki kuyruk yapısının boyutları büyümekte ve yönetimi çok daha zor ve karmaşık bir hal almaktadır [13]. Veri iletimini yönetmek için ilk gerçek aktif kuyruk yönetim algoritması olan RED geliştirilmiştir.

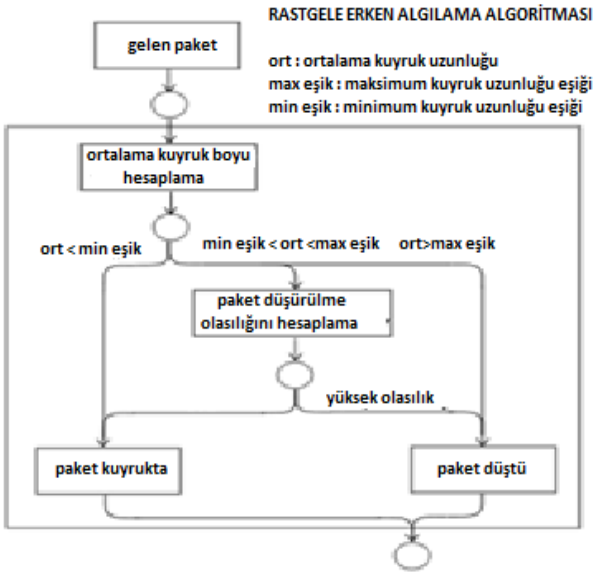
2.5. Rastgele Erken Algılama Algoritması (Random Early Detection Algorithm)

RED algoritması [14] ağ kuyruğundaki tıkanıklıkları tespit eden ve ağ trafik yükleme seviyesini ortalama kuyruk büyüklüğünü kullanarak ölçen bir algoritmadır. Ağ üzerinde bir sıkışıklık durumu ortaya çıktığında ortalama paket gecikmesi, gecikmedeki değişme miktarı ve iletim hızındaki değişme artacaktır. Dolayısıyla ağ üzerinde ortalama gecikme ve iletim hızı açısından daha iyi bir başarım elde edilmesi isteniyorsa ağ sıkışıklığı azaltılmalıdır. RED algoritması kuyruk yönetim algoritmaları içerisinde en çok tercih edilen algoritmalardan birisidir. RED algoritmasının çalışma mantığı kısaca şu şekildedir:

Ağ üzerindeki sıkışıklığı algılamak için, her bir paket geldiğinde öncelikle ortalama kuyruk boyu hesaplanır. Eğer hesaplanan ortalama kuyruk boyu belli bir sınırın üzerinde ise gelen paket hemen düşürülür. Belli bir sınırın altında ise paket bekletilmeksizin iletilir. Eğer yukarıda anılan iki sınır arasında bir boyutta ise ortalama kuyruk uzunluğuna göre hesaplanan bir olasılığa göre gelen paketin düşürülüp düşürülmeyeceğine karar verilir [15]. RED kuyruk yönetim algoritmasına ait akış diyagramı Şekil 4'te görülmektedir [16].

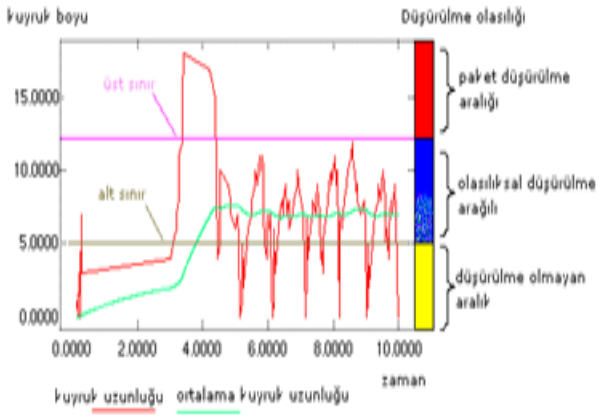
Şekil 5'teki grafikte, TCP bağlantı için anlık hız ve ortalama hız değerleri görülmektedir. Bir TCP bağlantı oluşturulduğunda örneğin http ile dosya indirdiğimizde TCP olabilecek en yüksek hızı deneyecektir. Bu denenmekte olan hız, yönlendirici sınırlarının üzerinde olacaktır. Bu durumda FIFO'dan gelen fazla paketin düşürülmesinden başka bir seçenek kalmamaktadır. TCP bağlantıda sistem paket kaybını gördüğünde, olabildiğinden daha hızlı aktarım yapmaya çalıştığını fark edecek fakat çok büyük paket kayıplarında ne kadar hızlı aktarım yapması gerektiğini tespit edemeyecektir.

Sonuç olarak hız sağlanabilecek değerin çok daha altına inecektir. Bir süre sonra TCP yeniden en yüksek değeri deneyecek ve aynı şey tekrar tekrar yaşanacaktır. İndirme hızlandırıcılarının genel olarak yaptıkları birden çok bağlantı açarak en az değerlerin görüldüğü yerlerde başka bir TCP bağlantının daha hızlı transfer yapmasını sağlayıp, bu değerlerin ortalamasını yükseltmek olur.



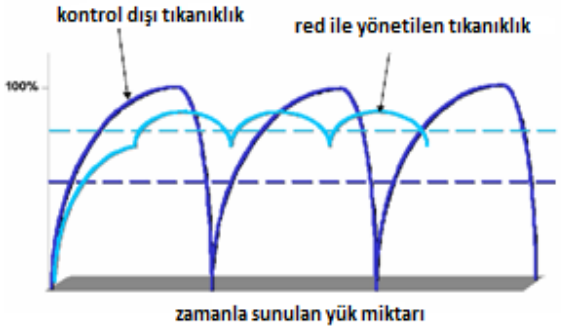
Şekil 4.RED algoritması akış şeması (RED algorithm flow diagram)

RED algoritmasının kuyruk yönetimi üzerinde oluşturmuş olduğu olumlu etki Şekil 5 ve Şekil 6'da görülmektedir.



Şekil 5.Rasgele erken algılama algoritması zamana bağlı kuyruk boyu değerleri (Random early detection algorithm, time-dependent queue length values) [17]

Şekil 6 RED algoritmasının bu faydasını açık olarak göstermektedir. Şekil 6'da kontrol edilmeyen trafiğin durumunu, RED uygulanmış bir bağlantının değerlerini ve kesik çizgilerle belirtilmiş ortalama değerlerini göstermektedir. RED algoritmasının yaptığı rastgele olarak önceden paketleri düşürmek olsa da, kayıplar kontrolsüz trafikteki kayıplardan çok daha az olduğu için verimi arttırmaktadır.



Şekil 6.RED zaman aşımı değerleri (RED time-out values)[18]

3. AĞ BENZETİM TEKNOLOJİLERİ (NETWORK SİMULATION TECHNOLOGIES)

3.1. Benzetim (simulasyon) Nedir? (What is simulation ?)

Benzetim, gerçek bir dünya süreci veya sisteminin işletilmesinin zaman üzerinde taklit edilmesidir [19]. Sistem objeleri arasında tanımlanmış ilişkileri içeren sistem veya süreçlerin bir modelidir. Benzetim gerçek bir sistemin taklit edilerek yapılmasıdır. Benzetim, taklit edilen gerçek bir olayın genelde bilgisayar yardımıyla modellenmesi, gerçek sistemin modelinin tasarımı ve bu model ile amacına yönelik olarak sistemin işletilmesi, sistemin davranışını anlayabilmek veya değişik stratejileri değerlendirebilmek için deneyler yürütülmesi sürecidir. Modelleme ve benzetim tasarımcılara, sistemi gerçekleştirmeden önce sistemin çalışırlığının görülmesini ve giriş parametrelerinin değişiminde sistemin göstereceği davranışın tespit edilmesini sağlar.

Benzetim sonuçlarının doğruluğu, modelin gerçek sisteme yakınlığı ile doğru orantılıdır. Bilgisayar benzetimi, sistemin yapısına bağlı olarak iki grupta toplanabilir:

- **Sürekli Olay (continuous-event)** olaylar zamana bağlı olarak bir süreklilik arz eder. Zaman doğrusal olarak arttıkça, prosesler de zaman içerisinde doğrudan değişirler.

- **Ayrık olay (discrete-event)** sayısal veri iletişim sistemleri ve bilgisayar ağları, mesajların üretimi ve dağıtımı gibi durumlar gerçekleştiğinde modüllerin çalıştırıldığı ve benzetim saatinin ilerlediği ayrık olay benzetim yöntemi ile modellenir.

3.2. NS-3 Ağ Benzetim Simülatörü (NS-3 network simulator)

Network Simulator 3 (NS-3) ayrık olaylı bir ağ simülatörüdür [20]. NS-3 genellikle multicast protokol ve yönlendirme simülasyonlarında aynı zamanda ad-hoc ağların araştırmasında da kullanılır. NS-3, yaygın ağ protokollerini destekler, kablolu ve kablosuz ağlar için simulasyon sonuçları sunar. Ayrıca sınırlı işlevsellik ağ emülatörü olarak kullanılabilir.

NS (Network Simulator), ağ simülasyonu oluşturmak ve gerçekleştirmek için ilk olarak 1989 yılında geliştirilmeye başlanmış, akademik araştırmalar için büyük öneme sahip açık kodlu bir simülatörüdür. NS kullanımı, 1995 yılında DARPA'nın sponsorluğunda ivme kazanmıştır ve günümüzde de simülatörün geliştirilmesi gönüllüler tarafından sürdürülmektedir.

NS ile, bir kablolu ya da kablosuz ağlarda istenilen miktarda düğümler ve bu düğümler arası linkler tanımlanabilir, yönlendirme algoritmaları ile çoklu gönderim protokolleri kullanılabilir ve ad-hoc network, WiFi, WiMax, vb. gibi bir takım popüler kablosuz ağ uygulamalarının modellemeleri ve simülasyonu gerçekleştirilebilir. NS simülatörü, ağ araştırma ve eğitimini destekleyerek protokol tasarımı, trafik araştırması sağlamakta, ücretsiz açık kaynak kodu ile karşılaştırmalı ve model paylaşımlı güvenilir bir deney ortamı sunmaktadır. NS programlama, yapılan çalışmalarda belli bir algoritmaya uygun olarak oluşturulmaktadır. Öncelikle olay programlayıcısı kurulur, izleme açılır ve kapanır, ağ modeli oluşturulur,

ilgili yönlendirme protokolü kurulur, sisteme hatalar girilir, iletim bağlantısı ile trafik oluşturulur ve uygulama gerçekleştirilerek veriler iletilir.

NS simülatörünün ikinci versiyonu NS-2 olarak adlandırılmaktadır [21]. NS, C++ tabanlı bir simülatördür ve Tcl dilinin objeye yönelik bir versiyonu olan OTcl ile bir simülasyon arayüzü vasıtasıyla ağ simülasyonu gerçekleştirmek mümkündür. C++, her paket işlemi için hızlı, detaylı bir kontrol sağlarken, Otcl ise mevcut C++ nesnelerini kullanarak simülasyon senaryo ayarları, periyodik veya tetiklemeli tahrik ve kolay yazılım imkanlarıyla öne çıkmaktadır.

NS-2, Linux platformunda çalışmaktadır ve Cygwin programı vasıtasıyla Microsoft Windows ortamında da simülasyonlar gerçekleştirilebilmektedir. NS-2'nin son sürümü 2.35, Haziran 2011'den itibaren kullanılmaya başlanmıştır. NS-2'nin ağ modellemesindeki en önemli dezavantajı düz evren modeli kullanmasıdır. Ancak, gerçek ağlar engebeli yeryüzü şekilleri, yükselti, yansıma, gecikme ve gölgeleme gibi etkiler nedeniyle idealden uzak bir görünüme sahiptir. Bu duruma çözüm olarak oluşturulan gölgeleme etkisi modelinin ise çok etkin çalışmadığı görülmüştür.

NS-3 simülatörü, NS simülatörünün yeni versiyonu olarak 1 Temmuz 2006 tarihinden itibaren geliştirilmeye başlanmıştır ve projenin tamamlanması için 4 yıllık bir süre öngörülmüştür[22]. Günümüzde tasarımı devam etmekte olan NS-3 simülatörü, University of Washington, Georgia Institute of Technology, ICSI ve INRIA gibi kurumlar tarafından geliştirilmeye devam etmektir. NS-3 simülatörü araştırma geliştirme ve akademik faliyetlerde kullanılmak üzere özellikle internet tabanlı sistemler için geliştirilen bir ayrık olay ağ simülatörü olarak tanımlanmaktadır. GNU GPLv2 (general public license) lisanslı yani bedelsiz ve açık kaynak kodlu bir yapıya sahiptir.
NS-3, C++ ve Python dilleri kullanılarak yazılabilmektedir. Yine NS-2 simülatöründe olduğu gibi Linux işletim sistemi üzerinde çalıştırılmaktadır ve Cygwin programı vasıtasıyla Windows tabanlı sistemlerde de kullanılabilmektedir. NS-3'te kod yapıları Doxygen isimli bir yazılım dökümantasyon program vasıtasıyla uygulanmaktadır. Şekil 7'de, 2006 yılından bugüne kadar artan kod satır sayısı gösterilmektedir.

Şekil 7.NS-3 kümülatif kod satır sayısı değişimi (The change of cumulative code lines number in NS-3) [23]

NS-3 simülatörünün 14 farklı sürümü mevcuttur. İlk stabil sürüm olan NS-3.1, Temmuz 2008'de kullanıma sunulmuştur. Bu sürümde simülatör çekirdeği, TCP/IPv4 trafiği, noktadan noktaya CSMA ve WiFi modelleri mevcuttur. İkinci sürüm NS-3.2, Eylül 2008'te kullanıma sunulmuş ve Python bağlayıcıları, gerçek zaman programlama, IEEE 802.1D, istatistik oluşturulmuştur.

NS-3.3, Aralık 2008'de hazırlanmış ve emülasyon özelliği, ilk IPv6 ve ICMP standartları simülatöre kazandırılmıştır. Nisan 2009'da devreye alınan NS-3.4 sürümünde ise simülatöre obje isimlendirme, yeni WiFi modelleri ve kuyruk listeleme fonksiyonları kazandırılmıştır. Temmuz 2009'da çıkartılan NS-3.5 sürümünde, IEEE 802.11e MAC EDCA, 802.11n A-MSDU, 802.11b PHY, Nakagami kayıp modeli, Gamma, Erlang ve Zipf rasgele değişken yapıları sisteme ilave edilmiştir. Ekim 2009'daki son NS-3.6 sürümünde ise, akış izleme, 5/10 MHz kanal ve WiFi fonksiyon eklemeleri, ICMP, IPv6 radvd, yeni test çerçevesi ve 802.11s mesh yapı ilaveleri gerçekleştirilmiştir.

NS 3.7 ve NS 3.12 sürümleri arasında IPv4 broadcast datagram 1 den 64'e kadar değiştirilmiştir. Topoloji başlıkları eklenmiş, IPv6 desteği geliştirilmiş. AODV uzaklık vektöründe RFC 3561 geliştirilmiştir. Wimax modelinde Point-to-Multipoint (PMP) mod ve WirelessMAN-OFDM PHY katmanıyla birlikte 802.16 özelliğinin MAC ve PHY gerçekleşimi sağlanmıştır. Kuyruk davranışları değiştirilmiştir. OFDM modülasyon tipi için Wifi hata oranı değiştirilmiştir. Wi-Fi iletim oranlarının adı ve yapısı değiştirilmiştir. C++'ın desteklediği 64 bitlik sayı yapısını da desteklemeye başlamıştır. En son çıkan sürüm olan NS-3.14 de hareketlilik modülü iki nesneyle birlikte eklenmiştir. Bu modül Model::GetRelativeSpeed() metodudur. Yeni

Ipv6AddressGenerator sınıfı ön ek ve arabirim kimliği tabanıyla birlikte ardışıl adres üretmek için eklenmiştir.

NS-3 kütüphanesi bir takım modüllerin birleşiminden oluşmaktadır. Bu modüller; core (çekirdek), simulator , common (ortak-yaygın kullanım alanı), node (düğüm) ve devices (araçlar) olarak sıralanabilir.

- Core: src/core dizini altında bulunmaktadır ve başka hiçbir modüle bağlı kalmadan kullanılabilen bir dizi hizmetler sunmaktadır. Bu hizmetlerin bir kısmı işletim sistemine bağlı olarak çalıştırılabilir.
- Simulator: src/simulator dizini altında bulunmaktadır ve olay zamanlamalı hizmetler sunmaktadır.
- Common: src/common dizini altında bulunmaktadır ve ağ simülasyonlarına özel hizmetler içermektedir.
- Node: src/node dizini altında bulunmaktadır. Başta ipv4 düğümleri olmak üzere her düğümlere ait arayüzlerin tanımlanmasından sorumludur.
- Devices: src/devices dizini altında bulunmaktadır ve fiziksel katman (MAC) seviyesindeki modelleri içerir.

3.2.1. NS-3' de Kuyruk Modülünün Çalışması (Operation of the tail module in NS-3)

NS'de kuyruk modülü multi-producer, multi-consumer kuyrukları gerçekleştirir. Özellikle bilgilerin çoklu iş parçaçıkları arasında güvenle değiştirilmesi gerektiğinde iş parçacığı programlama kullanılır.

Giriş noktası ve alımlardaki farkları üç tip kuyruk gerçekleştirir. Bir FIFO kuyruğunun görevi ilk eklenenleri ilk olarak almaktır. Bir LIFO kuyruğu, son olarak eklenen girişi ilk olarak (bir yığın gibi) işleme alır. Öncelikli kuyrukla girişler sıralanır ve (heapq modul kullanılarak) en düşük değerli giriş ilk alınır.
Kuyruk modülü aşağıdaki sınıfları ve durumları tanımlar:

class **Queue.Queue(***maxsize) :* Bir FIFO kuyruğunu ifade eder. Maxsize, kuyruğa yerleştirilebilir öğelerin sayısı en yüksek limitini ayarlayan bir tamsayıdır. Kuyruktaki öğeler tüketilene kadar, bir kez ulaşılan büyüklükler engellenecektir. Eğer maxsize sıfır veya sıfırdan daha az ise kuyruk boyutu sonsuzdur.

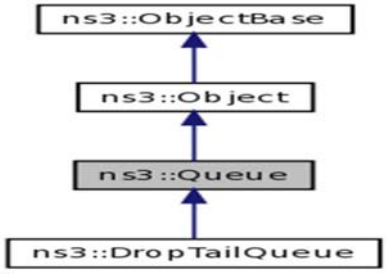
class **Queue.LifoQueue(***maxsize***) :** Bir LIFO kuyruğunu ifade eder. Maxsize, kuyruğa yerleştirilebilir öğelerin sayısı en yüksek sınırını ayarlayan tamsayı bir değerdir. Kuyruktaki öğeler tüketilene kadar, bir kez ulaşılan büyüklükler engellenecektir.

class **Queue.PriorityQueue(***maxsize***) :** Bir priority queue kuyruğunu düzenlemek için. Maxsize, kuyruğa yerleştirilebilir öğelerin sayısı en yüksek sınırını ayarlayan bir tamsayıdır. Kuyruktaki öğeler tüketilene kadar, bir kez ulaşılan büyüklükler engellenecektir.

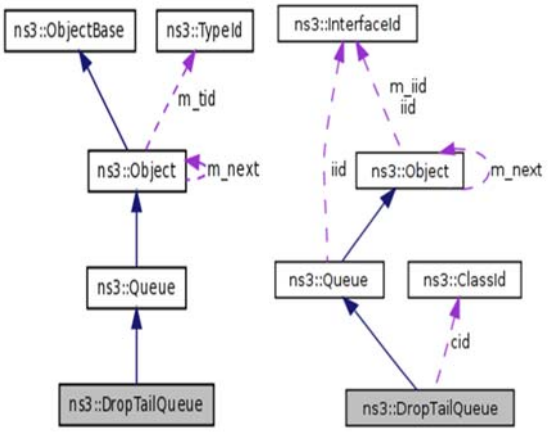
exception **Queue.Empty&Full :** İstisna olarak engelleme olmadığında get(), put() (veya get_nowait ()veya put_nowait ()) boş veya dolu iken boş veya dolu kuyruk nesnesi olarak adlandırılır.

3.2.2. NS-3 Kuyruk Referansları (NS-3 queue reference)

Şekil 8 ve Şekil 9'da NS-3 ağ simülatörünün yapısında kuyruk yönetim algoritmalarının kalıtsal diagram tasarımı ve bu sınıftan türetilen DROP TAIL kuyruk algoritmasının nesne ve bağlantı digramları görülmektedir.



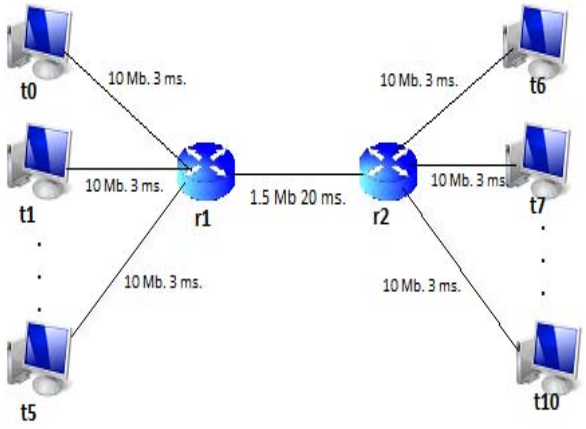
Şekil 8. NS-3:kuyruk kalıtsal diagramı (NS-3 queue inheritance diagram) [24]



Şekil 9. NS-3 DROP TAIL sınıf diagramı (DROP TAIL class diagram) [25]

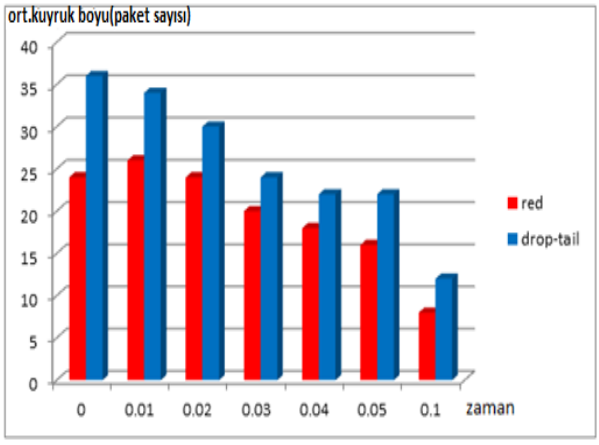
4. NS-3 KUYRUK YÖNETİMİ SİMÜLASYONU (NS-3 QUEUE MANAGEMENT SIMULATION)

Bu çalışmada diğer algoritmalardan performans olarak daha yüksek başarıma sahip olan RED ve DROP TAIL algoritmaları üzerinde testler gerçekleştirilecektir. NS-3'te ön tanımlı olarak DROP TAIL algoritması bulunmaktadır. NS-3 üzerinde bunun haricinde sadece RED algoritması geliştirilmiştir. Bu iki algoritmanın NS-3 ağ simülatöründe karşılaştırılması için gerekli simülasyon hazırlanmış ve testler gerçekleştirilmiştir.

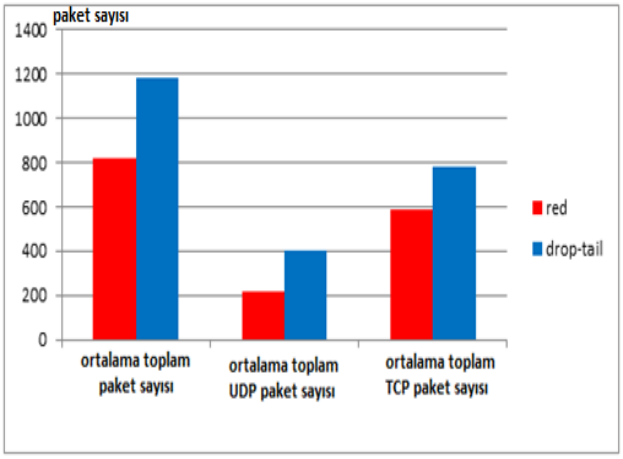


Şekil 10. Deney ağı topolojisi (test network topology)

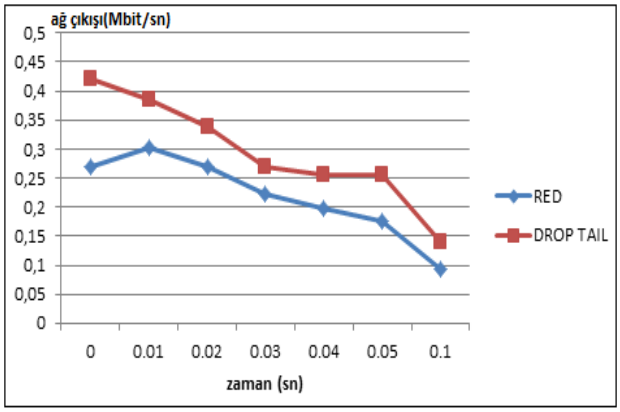
NS-3'te ön tanımlı olarak DROP TAIL algoritması bulunmaktadır. NS-3 üzerinde bunun haricinde sadece RED algoritması geliştirilmiştir. Bu iki algoritmanın NS-3 ağ simülatöründe karşılaştırılması için gerekli simülasyon hazırlanmış ve testler gerçekleştirilmiştir.



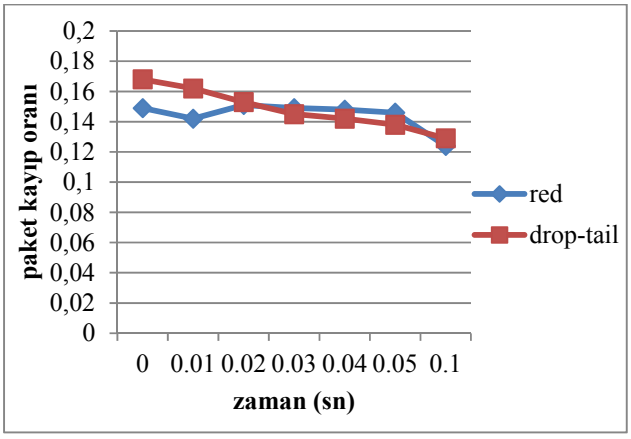
Şekil 11.DROP TAIL ve RED algoritmalarında oluşan kuyruk paket sayıları (the number of packet queuing in DROP TAIL and RED)



Şekil 12. Yönledirici 1 DROP TAIL ve RED paket sayıları (DROP TAIL and RED algorithm number of packet in Router 1)



Şekil 13. Kuyruk ağ çıkış değerleri (the tail of the output values of the network)



Şekil 14. Kuyruk algoritmaları paket kayıp oranları (packet loss ratios of queue algorithms)

Şekil 11, 12, 13 ve 14'teki testler sonucu elde edilen grafikler incelendiğinde, Şekil 11'de işlem süresi boyunca belirli aralıklarda RED ve DROP TAIL algoritmalarının kullanımı sonucu oluşan ortalama kuyruk boyu görülmektedir. Ortalama kuyruk boyu değerlerine bakıldığında her iki algoritma için, zamanın ilerledikçe, kuyruk boyunun azaldığı görülmektedir. Değerlendirme yapılan her zaman aralığı için, DROP TAIL algoritmasının kuyruk boyunun RED algoritmasına göre daha uzun olduğu tespit edilmiştir. Şekil 12'de ise işlem süresi boyunca RED ve DROP TAIL algoritmalarının kullanımı sonucu ortalama toplam paket sayıları, TCP ve UDP protokol bağlantılarındaki ortalama paket sayıları görülmektedir. RED algoritması için yapılan testte, yönlendirici 1 üzerinde toplam 800 paket geçişi gerçekleşmiş ve bu paketlerin 200 adeti UDP, 600 adet TCP paketi olduğu görülmektedir. DROP TAIL algoritması için gerçekleştirilen uygulamada, 1200 paket geçişinin 800 adeti TCP, 400 adeti UDP paketi olarak tespit edilmiştir. Şekil 13'de değerlendirme yapılan zaman aralığında kuyruklar üzerindeki ağ çıkış değerleri görülmektedir. DROP TAIL algoritmasının ağ çıkış değerinin RED algoritmasından daha iyi sonuçlar ürettiği görülmektedir. Şekil 14'te farklı zaman aralıklarında farklı paket kayıp olasılıksal değerleri görülmektedir. Grafiksel değerlere bakıldığında DROP TAIL algoritmasında RED kuyruk yönetim algoritmasından daha fazla paket kaybı olduğu görülmemektedir. RED algoritmasının kuyruk boyu uzunluğu, DROP TAIL algoritmasına göre daha az olmasına rağmen paket kayıp oranları birbirine çok yakındır. Paket kayıp oranları değerlendirmesinde RED kuyruk yönetim algoritmasınındaha iyi bir performans sağladığı söylenebilir.

5. SONUÇ VE DEĞERLENDİRME (CONCLUSION AND EVALUATION)

Literatürde kuyruk yönetim algoritmalarının karşılaştırılması ile ilgili yapılmış olan çalışmalar bulunmaktadır [26] [27]. Bu makalede literatürde tanımlı ve trafik ağ yoğunluğunun kontrol altına alınmasında ve düzenlenmesine büyük öneme sahip olan kuyruk yönetim algoritmaları tanıtılmıştır. Ağ simülatör programları içerisinde yaygın bir kullanım alanına sahip olan NS-3 ağ simülatöründeki kuyruk yönetim yapısı incelenmiştir. NS-3 ağ simülatörü üzerinde hazırlanmış olan ağ topolojisi üzerinde RED ve DROP TAIL kuyruk algoritmaları ile TCP ve UDP bağlantılar üzerinde değişik parametreler kullanılarak testler gerçekleştirilmiştir.

Sistem üzerinde tanımlı kuyruk yapılandırması ne kadar mükemmel olursa olsun, gün geçtikçe artan internet kullanımı ve büyüyen ağ yapıları neticesinde kuyruklarda paketlerin birikimine sebep olmakta ve bunların içerisinden bir kısmının düşürülme zorunluluğunu ortaya çıkarmaktadır. Doğru seçilmiş ve sağlıklı çalışan bir kuyruk yapısı ağ trafiğini rahatlatarak ve tıkanıklıkları engellemede çok büyük bir rol oynayacaktır. RED algoritması klasik yöntemlerin aksine düşürülecek olan paketlerin seçiminde olasılıksal ve istatiksel verileri kullanarak düşürülecek olan paketin seçimine olanak tanımaktadır.

Yapılan testler ve bu konu üzerinde daha önce yapılmış olan çalışmalar birlikte değerlendirildiğinde, RED algoritmasının sağladığı olasılıksal yöntemler ve iki farklı eşik değeri belirlemesinin klasik DROP TAIL yönteminden daha başarılı sonuçlar verdiği tespit edilmiştir.

KAYNAKLAR (REFERENCES)

[1] W. Feng, D. D. Kandlur, D. Saha, and D. G. Shin, "BLUE: A New Class of Active Queue Management Algorithms", Technical Report CSE-TR-387-99, Dept. of EECS, University of Michigan, April 1999.

[2] Blake, S., Black, D.L., Carlson, M.A., Davies, E., Wang Z. and Weiss, W. , " An architecture for differentiated services". 1998.

[3] R. Fengyuan et al., "A Robust Queue Mana gement Algorithm Based on Sliding Mode Variable Structure Control," Proc. IEEE INFOCOM'02,New York, NY, July 2002, vol. 1.

[4] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," IEEE/ACM Trans. Netw., vol. 1, no. 4, pp. 397–413, Aug. 1993.

[5] Santhi V., Natarajan A. M., Active Queue Management Algorithm for TCP Networks Congestion Control, European Journal of Scientific Research, Vol.54 No.2 (2011), pp.245-257.

[6] NS3, http://www.nsnam.org/docs/release/3.14 /tutorial/single html/index.html, 2012.

[7] FIFO, http://opalsoft.net/qos/DS-22.htm , 2012.

[8] Peterson L. L., Davie B. S., Computer Networks: A Systems Approach, Morgan Kaufmann, p. 402–403.

[9] McKenney, P., "Stochastic fairness queuing", internetworking : research and experience, vol.2 pp.113-131, 1991.

[10] SFQ, http://opalsoft.net/qos/DS-25.htm ,2012.

[11] WFQ, http://opalsoft.net/qos/DS-23.htm ,2012.

[12] Braden B., Clark D., Crowcroft J., Davie B., Deering S., Estrin D., Floyd S., Jacobson V., Minshall G., Partridge C., Peterson L., Ramakrishnan K. K., Shenker S., and Wroclawski J. "Recommendations on queue management and congestion avoidance in the internet", Internet Draft, (1998)

[13] Wydrowski B., Zukerman M., "GREEN: An Active Queue Management Algorithm for a Self Managed Internet", Proceedings of ICC 2002, New York, Vol. 4, 2368-2372 , 2002.
[14] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Trans. Net., vol. 1, no. 4, Aug. 1993, pp. 397–413.
[15] Sally Floyd and Van Jacobson ; "Random Early Detection Gateways for Congestion Avoidance" , IEEE/ACM Transactions on Networking , August 1993.
[16] David l.,Ashok Kumar, A Dot to DDoS Attack on Cloud Computing Environment Using Adaptive WRED Congesiton Control Algorithm.
[17] RED, http://opalsoft.net/qos/DS-26.htm ,2012.
[18] RED, http://www.cisco.com/networkers/nw00/ pres /2808_6-28.pdf , 2012.
[19] J.Banks,S.Carson, "Applying the Simulation Process", proceedings of the 1988 winter simulation conference ,1988.
[20] NS-2, http://www.isi.edu/NSnam/NS/ , 2012.
[21] NS-2, http://www.NSnam.org/docs/tutorial/ tutorial.html , 2012.
[22] NS-3, http://www.nsnam.org/overview/what-is-NS-3/ , 2012.
[23] NS-3, http://www2.NSnam.org/ ,2012.
[24] NS-3 Kuyruk yapısı , http://netdb.cis.upenn.edu /rapidnet/doxygen/html/classNS-3_1_1_queue .html , 2012.
[25] DROP TAIL, http://netdb.cis.upenn.edu/rapidnet /doxygen/html/classNS-3_1_1_drop_tail_queue .html ,2012.
[26] M.WANG, "Comparison between drop tail and AQM-red in wireless network", master thesis,2012.
[27] V. Jacobson. Congestion Avoidance and Control. In Proceedings of ACM SIGCOMM, pages 314–329, August 1988.