BİLGİSAYAR AĞLARI ÖDEVİ – 7

(BM402)

Ayben GÜLNAR-191180041

MAYIS 2023

İçindekiler Tablosu

Şekiller Listesi 4
1.GİRİŞ 4
2.UDP Protokolü ve Tıkanıklık 5
2.1 UDP Protokolü nedir? 5
2.2 Network Congestion ve UDP Protokolünde Tıkanıklık 6
2.2.1 UDP Protokolünde Neden Kontrol Yapılmalı 8
3.TRANSPORT LAYER'DA UDP KULLANMAK VE APPLİCATİON LAYER'DA TIKANIKLIK DENETİMİ YAPMAK 9
3.1 UDP Üzerinden Büyük Transferler Gerçekleştiren Uygulamalar İçin Tıkanıklık Kontrolü 9
3.2 Düşük Veri Hacimli Uygulamalar İçin Tıkanıklık Kontrolü 10
3.3 UDP Bidirectional Communications Protokolü Olarak Kullanıldığında Tıkanıklık Kontrolü ... 11
3.4 UDP ve Paket Kaybı 11
3.5 NS2 Kullanarak UDP Protokolünde Tıkanıklık ve Akış Kontrolünün Analizi 12
3.6 Leaky Bucket Algoritması 12
3.7 Token Bucket Algoritması 14
3.8 Random Early Detection RED Yaklaşımı 15
3.9 ECN(Explicit Congestion Notification) Yaklaşımı 16
3.10 Rate-Based Congestion Control (RBCC) Yaklaşımı 16
3.11 Quality of Service – QoS 17
4.SONUÇ 18
5. KAYNAKÇA 19

ŞEKİLLER LİSTESİ

Şekil 2.1.1 Paket Formatı 5
Şekil 2.1.2 Client server app [1] 6
Şekil 2.2.1 Network Congestion 6
Şekil 3.4.1 Paket Kaybı 12
Şekil 3.6.1 Leacky Bucket 13
Şekil 3.8.1 RED Algorithm 15

1.GİRİŞ

UDP, tıkanıklık kontrolü, paket kaybı kontrolü, çoğaltma veya yeniden sıralama kontrolü sağlamaz. Tüm bunlar uygulama seviyesinde uygulanmalıdır ve her protokol uygulamanın özel gereksinimlerine göre farklı şekillerde ele alabilir [1].

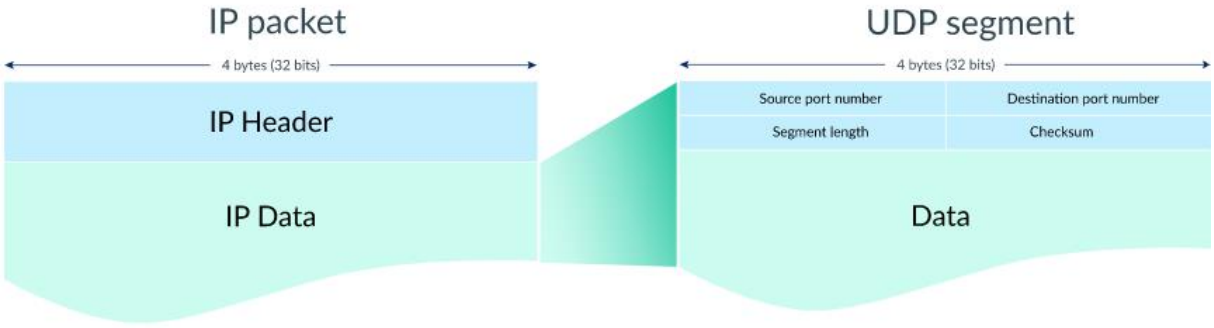
Bu bağlamda, transport layer'da UDP kullanıp, application layer'da tıkanıklık kontrolü yapmak, bu tür uygulamalarda kullanışlı olabilir. Ancak, bu yaklaşım bazı zorluklarla karşılaşabilir. Örneğin, UDP'nin bağlantısız yapısı nedeniyle, ağın durumuna ve yüküne ilişkin bilgilerin doğru bir şekilde tespit edilmesi ve kullanılması zor olabilir. Ayrıca, UDP protokolü, kayıp veya hatalı paketlerin yeniden gönderilmesini otomatik olarak sağlamadığından, tıkanıklık kontrolü mekanizmalarının veri bütünlüğü ve doğruluğu üzerindeki etkisi dikkatle incelenmelidir.

Bu ödevde, transport layer'da UDP kullanıp, application layer'da tıkanıklık kontrolü yapmanın farklı yaklaşımları üzerine detaylı bir analiz yapılacaktır. İlk etapta udp protokolü ve ağ tıkanıkları hakkında detaylı araştırma yapılmıştır. Sonrasında ise çeşiti algoritma ve yaklaşımlar ele alınmıştır. Bu araştırma, gerçek zamanlı uygulamalar gibi özel senaryolarda tıkanıklık kontrolü mekanizmalarının nasıl kullanılabileceği ve bu mekanizmaların performansı üzerindeki etkisi hakkında bir fikir vermesi açısından önemlidir.

2.UDP Protokolü ve Tıkanıklık

2.1 UDP Protokolü nedir?

The User Datagram Protocol (UDP) IP'nin üzerinde veri iletim protokollerinden biridir. UDP, paketlerdeki bozuk verileri tespit etmek için bir mekanizma sağlar, ancak kaybolan veya sırası karışmış paketler gibi diğer sorunları çözmeye çalışmaz. Bu nedenle UDP bazen Güvenilmez Veri Protokolü olarak da bilinir. UDP basit ancak hızlıdır, en azından IP üzerinden çalışan diğer protokollere göre. Hızın doğruluktan daha önemli olduğu zaman hassas uygulamalar (örneğin gerçek zamanlı video akışı) için sıkça kullanılır. Paket formatı, UDP'nin IP üzerinden paket gönderirken, her IP paketinin veri kısmı bir UDP segmenti olarak biçimlendirilir.



Şekil 2.1.1 Paket Formatı

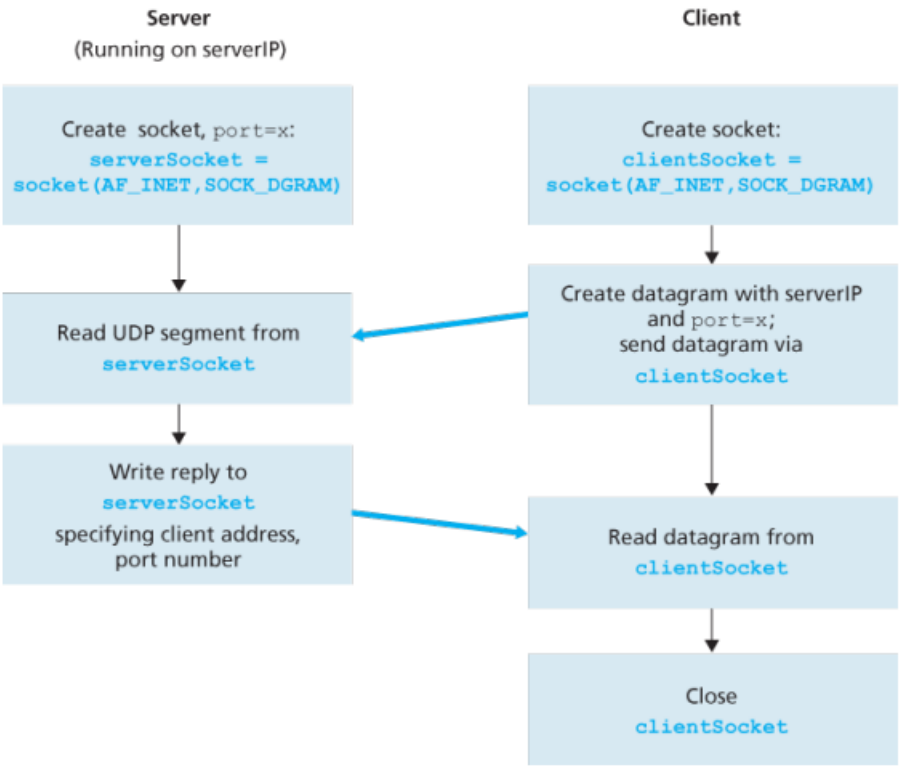
Her UDP segmenti 8 byte başlık ve değişken uzunlukta veri içerir. Port numaraları UDP başlığının ilk dört byte'ı kaynak ve hedef için port numaralarını saklar. Ağ bağlantılı bir cihaz, bir okyanus limanının farklı limanlarına gemiler alması gibi farklı sanal limanlarda mesaj alabilir. Farklı port numaraları, farklı türde ağ trafiğini ayırt etmeye yardımcı olur.

UDP, veri transferi işlemini gerçekleştirmek için IP protokolünü kullanır. Bir veri paketi, bir kaynak IP adresi ve hedef IP adresi ile belirtilen bir hedef porta gönderilir. Her UDP segmentindeki başlık, kaynak ve hedef port numaralarını içerir. Bu port numaraları, hedef cihazın uygun uygulaması tarafından işlenmek üzere belirli bir hizmete yönlendirilmesine yardımcı olur.

UDP, TCP (Aktarım Kontrol Protokolü) gibi diğer taşıma protokollerine kıyasla daha az özellik sunar. Örneğin, UDP paketleri, paket kaybı veya yeniden düzenlenme durumunda yeniden gönderilmez ve ayrıca tıkanıklık kontrolü için özel bir mekanizma sağlamaz. Ancak, UDP hızlı ve basit olması nedeniyle gerçek zamanlı uygulamalarda (örneğin video akışı) sıkça kullanılır.

UDP paketleri, hızlı bir şekilde gönderilir ve alıcıya ulaşır. Bununla birlikte, UDP'nin güvenilirliği daha azdır ve bir ağdaki diğer protokollerin aksine, paketleri sırayla teslim etme garantisi yoktur. Bu nedenle, bazı uygulamalar, UDP'nin sağladığı hız ve basitliğin yanı sıra, güvenilirliği artırmak için ek yöntemler

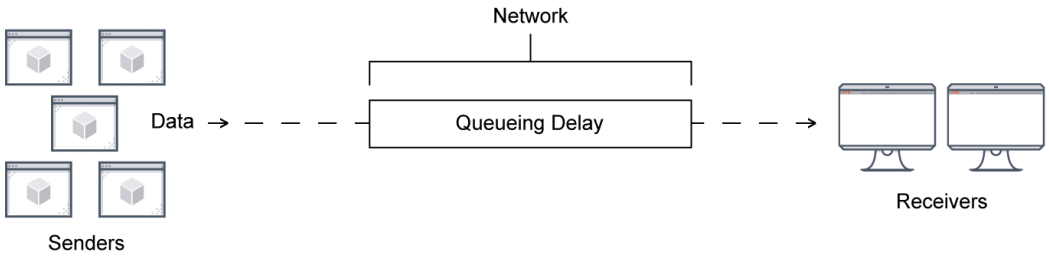
kullanır. Örneğin, bir video akışı uygulaması, paketlerin kaybolması durumunda yeniden gönderme işlemlerini yapabilir [1,2].



Şekil 2.1.2 Client server app [1]

2.2 Network Congestion ve UDP Protokolünde Tıkanıklık

Ağ tıkanıklığı, bir ağın karşılayabileceğinden daha fazla veri paketi trafiğiyle karşılaştığında meydana gelir. Bu veri trafiğinin yığılması, aynı anda çok fazla iletişim ve veri isteği yapıldığında, yeterli ağ genişliği olmayan bir ağ üzerinden gerçekleşir.



Şekil 2.2.1 Network Congestion

Ağ tıkanıklığı genellikle geçici olsa da, performansı etkileyebilen rahatsız edici ağ sorunlarına neden olabilir. Bu sorunlar arasında yüksek düzeyde jitter, paket kaybı, gecikme ve verimlilik azalması yer alır. Tıkanmış bir ağ, ağınızda ve dışında ağ tıkanıklığını önceden tespit edebilen ağ performansı izleme

araçlarına sahip olmanız önemlidir. Ağ tıkanıklığına neden olan en yaygın nedenlerden bazıları aşağıda açıklanmaktadır.

"Broadcast domain" bir ağ yapısına uygulanır. Bu, bir işletme, eğitim kurumu veya VLAN'daki ağ olabilir. "Host" ise yayın alanı içindeki her bir bireysel yönlendirici veya anahtarlayıcıya atıfta bulunur. Yapıda çok fazla ana bilgisayar, aynı anda çok fazla cihazın ağ erişimi istemesi nedeniyle bir aşırı yüke neden olabilir.

Bu kavram, mobil ağlar ve yönlendiriciler için de geçerlidir. Mobil ağlar ve yönlendiriciler yayın alanıdır. Bilgisayarlar, tabletler veya telefonlar ise ana bilgisayarlar olarak kabul edilir. Ağınızı verilerin aktığı bir boru olarak düşünürseniz, bant genişliği iş kritik trafiğin aktığı "borunun boyutu" olarak kabul edilir. Boru, aynı anda tüm trafiği taşımak için yeterince büyük değilse, trafiğin sorunsuz akması sorunlara neden olabilir ve ağ tıkanıklığına yol açabilir.

Örneğin, televizyon yayını saatlerinde ağ bant genişliği kullanımı etkilenebilir. Örneğin, bir video akışı hizmeti %40 interneti tüketiyorsa, ağ tıkanıklığına neden olabilir. Ayrıca, ağınızda aşırı aktif cihazlar çalıştığında da ağ tıkanıklığı oluşabilir. Aynı anda büyük miktarda trafik ağınızdan geçerken, veri paketleri birbirine takılabilir, aynı anda çok fazla araç gibi. "Bant genişliği avcıları" adı verilen, ya çok fazla veri iletişimi yapan ya da çok sık çalışan cihazlar da aşırı aktiviteye neden olur.

Ağ yanlış yapılandırması, ağ tıkanıklığının yaygın nedenlerinden biridir. Bu, örneğin ağ mühendislerinin tekrarlayan ve tek seferlik komut dosyaları aracılığıyla sistemlere hata sokmasıyla olabilir.

Konfigürasyon yönetiminin diğer yönü, düzenli bakım ve test gerektirir. Teknoloji altyapısının tüm bileşenleri gibi, ağ cihazları ve ekipmanları üzerinde bakım yapılmaması ve ağ trafiği izlenmemesi, kaçınılmaz bir çöküşe neden olabilir. Sınır Geçidi Protokolü (BGP), internetin özerk sistemler (AS) arasında yönlendirme bilgileri değiştirmesini sağlayan bir geçit protokolüdür. Bu genellikle en kısa mantıksal yoldan gerçekleşir ve bu yoldan ne kadar çok trafik geçtiği düşünülmeden gerçekleşir. Bu, transit yolların aşırı yükle dolması, daha yavaş hızlar ve ağ tıkanıklığına neden olabilir.

Hasarlı veri paketleri, yeniden gönderilmesi gereken veri paketleri elbette kayıp paketlere neden olur. Bu iki veya daha fazla kez gerçekleşiyorsa, bu ağınızın artı bir yarar elde etmeden ağ tıkanıklığı yaşamasına neden olur.

Veri paketi çarpışmaları, kötü kablo, fazla kullanılan cihazlar veya kötü ekipmanlar nedeniyle oluşabilir ve tüm paketlerin net bir ağ için yeniden iletilmesi için beklemesi gereken ciddi bir durum oluşturabilir. Bu daha büyük bir tıkanıklık ve gecikmeye neden olur ve otoyol çarpışmasıyla aynı durumda olduğu gibi, trafik yönlendirmesi sıklıkla gereklidir [3].

2.2.1 UDP Protokolünde Neden Kontrol Yapılmalı

UDP protokolü, TCP protokolüne kıyasla daha hızlı ve daha az işlemci gücü gerektiren bir iletişim protokolüdür. Ancak, UDP protokolü, TCP protokolünde olduğu gibi tıkanıklık denetimi yapmaz. Bu nedenle, UDP protokolü kullanılarak gerçekleştirilen ağ trafiği, ağdaki tıkanıklık durumunda paket kaybına ve hizmet kalitesinin düşmesine neden olabilir.

UDP üzerinde tıkanıklık denetimi yapmak için farklı yaklaşımlar önerilmiştir. Bu yaklaşımlardan bazıları, paket kaybı oranını ve gecikmeyi ölçerek tıkanıklığı algılamayı ve buna göre önlem almayı önerirken, diğer yaklaşımlar akış kontrolü, kaynak tüketimi ve sıra numaralandırma gibi yöntemleri kullanmaktadır.

Özellikle video ve ses gibi gerçek zamanlı uygulamaların kullanımının artmasıyla birlikte, UDP protokolü üzerinde tıkanıklık denetimi yapılması gerekliliği daha da önem kazanmıştır. Bu uygulamalarda, tıkanıklık denetimi olmadan gerçekleştirilen trafiğin, ses ve görüntü kalitesinin düşmesine neden olabileceği bilinmektedir.

Sonuç olarak, UDP protokolü üzerinde tıkanıklık denetimi yapılması gerekliliği konusu, ağ performansının iyileştirilmesi ve hizmet kalitesinin korunması açısından önemlidir. Bu nedenle, UDP protokolünü kullanan uygulamaların tıkanıklık denetimi yapacak mekanizmaları içermesi gerekmektedir. Bir uygulama veya protokol, tıkanıklık kontrolü yapılmayan bir taşıma protokolü kullanmayı tercih etse bile, gereksinimlerini karşılamak için hedef bir sunucuya UDP datagramlarını gönderme hızını kontrol etmelidir. Önemli olan, bir uygulamanın, nasıl ürettiği trafikten bağımsız olarak, hedefe gönderdiği tüm UDP trafiği üzerinde tıkanıklık kontrolü yapması GEREKTİĞİDİR. Örneğin, çoklu işlem süreçlerini çatallayan veya başka şekillerde birden fazla soket kullanarak UDP datagramları üreten bir uygulama, birikimli trafiğin üzerinde tıkanıklık kontrolü yapmalıdır.

Tıkanıklık kontrolü yapmak için birkaç yaklaşım tartışılmaktadır. Bu bölüm, genel konuları anlatırken özellikle tekil ve çoklu alıcı kullanımı üzerinde durulur. Tartışılan tüm yaklaşımlar, UDP gönderen uygulamalar için uygun değildir. Bölüm 3.1, UDP üzerinden büyük transferler gerçekleştiren uygulamalar için tıkanıklık kontrolü seçeneklerini tartışmaktadır. Bu tür uygulamalar, veri alışverişi sırasında birkaç ardışık tur boyunca yol üzerinde örneklemeler yapan şemalar kullanarak, yükünün mevcut yükünü destekleyebilecek bir gönderim hızı belirlemek için kullanabilirler. Diğer uygulamalar yalnızca birkaç UDP datagramı ile bir hedefe iletişim kurarlar. Bölüm 3.2 bu tür "düşük veri hacimli" uygulamalar için tıkanıklık kontrolü seçeneklerini tartışmaktadır. Genellikle güvenli bir gönderim hızı belirlemek için yol üzerinde iteratif olarak örneklemeye yeterli veri göndermedikleri için, farklı türde tıkanıklık kontrolü mekanizmaları kullanmaları gerekir. Bölüm 3.3, UDP bir tünelleme protokolü olarak kullanıldığında tıkanıklık kontrolü düşüncelerini tartışmaktadır. UDP kullanan bir uygulama veya protokol, tıkanıklık kontrollü bir taşıma protokolü kullanmamayı tercih etse bile, hedef bir sunucuya gönderdiği UDP datagramlarının gönderim hızını kontrol etmelidir. Ayrıca, uygulama bu trafiği nasıl

oluşturursa oluştursun, tüm UDP trafiği üzerinde tıkanıklık kontrolü yapmalıdır. veri transferleri gerçekleştiren uygulamalar için, verileri değiştirdiği sıralarda yolu örnekleyen şemalar kullanabilirler. Diğer uygulamalar ise sadece birkaç UDP datagramını hedefe gönderirler. Bu gibi "düşük veri hacimli" uygulamalar, güvenli bir gönderim hızını belirlemek için yolu iteratif olarak örneklemeye yeterli veri göndermediklerinden, farklı tıkanıklık kontrol mekanizmaları kullanmalıdır [3,4].

3.TRANSPORT LAYER'DA UDP KULLANMAK VE APPLİCATİON LAYER'DA TIKANIKLIK DENETİMİ YAPMAK

Transport Layer'da UDP protokolünün kullanılması, uygulama katmanında tıkanıklık denetimi yapılmasını gerektirebilir. Bu nedenle, farklı tıkanıklık denetimi yaklaşımları kullanılabilir:

Veri Tüketim Kontrolü: Uygulama tarafından gönderilen veri miktarı belli bir oranda sınırlandırılarak ağın etkililiği artırılır. Bu yöntem, kuyruk boyutları, trafik sınıflandırması ve çıktı hızı gibi farklı mekanizmalar kullanarak veri miktarını düzenler.

Akış Yönetimi: Veri gönderen uygulamanın alıcı tarafından işlenemeyecek kadar hızlı göndermesini önleyerek tıkanıklık oluşumunu engelleyen bir mekanizmadır.

Paket Kaybı ve Gecikme Ölçümü: Ağın durumunu ölçmek için paket kaybı oranı ve gecikme süresi gibi ölçümler yapılır. Bu ölçümler sayesinde tıkanıklık algılanır ve önlem alınır.

Sıra Numaralama: Uygulama tarafından gönderilen paketlere sıra numaraları atanır ve alıcı bu numaraları kullanarak paketlerin doğru sırayla işlenmesini sağlar. Bu yöntem, ağ trafiğini düzenleyerek tıkanıklık durumunda ağın etkililiğini artırır.

Trafik Yönlendirme: Tıkanıklık durumunda, trafik yönlendirme mekanizmaları kullanılarak ağ trafiği farklı yollarla yönlendirilir. Bu yöntem, ağın daha dengeli çalışmasını ve tıkanıklık durumunda trafiğin etkin bir şekilde yönetilmesini sağlar.

UDP protokolü üzerinde çalışan uygulamaların performansını artırmak ve hizmet kalitesini sağlamak için tıkanıklık denetimi yaklaşımları kullanılabilir. Ancak, kullanılacak yaklaşımlar uygulamanın ihtiyacına ve kullanım senaryosuna göre belirlenmelidir [4].

3.1 UDP Üzerinden Büyük Transferler Gerçekleştiren Uygulamalar İçin Tıkanıklık Kontrolü

Protokoller ve uygulamalar için etkili tıkanıklık kontrolü uygulamalarının bir parçası olarak iletişim uçları arasındaki gecikmenin anlaşılmasının önemini vurguluyor. Gecikme tahmini, tıkanıklık kontrolü yapılandırması, yeniden gönderme tetikleme ve paket kaybı algılama gibi bir dizi protokol işlevinde kullanılabilir. Uygulamalar, kendi gecikme tahmini mekanizmalarını tasarlamak zorunda kalan durumlar için aşağıdaki yönergeleri sağlar.

Genel öneri, uygulamaların mevcut tıkanıklık kontrol tekniklerinden ve burada belirtilen gecikme tahmincilerinden yararlanması gerektiğidir. Aşağıdaki yönergeler, kendi gecikme tahmini mekanizmalarını tasarlamak zorunda olan uygulamalar için sağlanmaktadır.

Yönergeler, "gidiş-dönüş süresi" değil, "gecikme" terimiyle çerçevelenmiştir, çünkü bazı durumlar sadece ağ tabanlı gecikmeyi (örneğin, TCP-Dostu Hız Kontrolü (TFRC)) karakterize etmeyi gerektirirken, diğer durumlar bir mesajın yeniden gönderilmesi gerektiğinde ne zaman anlayacağımızı içerir. Gelen geri bildirim için uzaktaki uç tarafından gereken zamanı da dahil etmeyi gerektirir.

Uç noktalar arasındaki gecikme genellikle dinamik bir özelliktir. Bu nedenle, tahminler, değişkenliği hesaba katmak için birden fazla son ölçüm örneğinin bir tür ortalamasını temsil etmelidir. Bunu yapmak için üstel Ağırlıklı Hareketli Ortalama (EWMA), TCP'de ve TFRC'de yararlı olmuştur. Bağımsız gecikme tahminleri, bir uç noktanın iletişim kurduğu her hedef için korunmalıdır.

Gecikme örnekleri, belirsiz işlemlerden türetilmemelidir. Örneğin, verileri yeniden gönderen ancak daha sonra hangi kopyanın kabul edildiğini belirleyemeyen bir protokoldeki kanonik örnektir. Bu belirsizlik, gecikmenin doğru hesaplanmasını problemli hale getirir. Karn algoritmasının tartışmasına bakın. Bu gereksinim, gönderenin yanıltıcı ölçümlere güvenmeden gecikmenin sağlam bir tahminini oluşturmasını sağlar. Bir uç noktanın diğer uç noktalarla iletişim kurduğu her hedef için bağımsız gecikme tahminlerinin tutulması önerilir. Gecikme örnekleri, belirsiz işlemlerden türetilmemelidir. Bu, tekrarlanan veri aktarımı protokollerinde ortaya çıkan ve hangi kopyanın kabul edildiğinin belirlenemediği durumlarda geçerlidir. Bu belirsizlik, gecikmenin doğru hesaplanmasını zorlaştırır. Gecikme tahmininin bir zamanlayıcıyı tetiklemesi durumunda, kayıp algılama sağlanacaksa, zamanlayıcının süresi ağdaki tıkanıklık göstergesi olarak yorumlanmalıdır. Bu durumda, gönderim hızı güvenli bir korumalı hızda ayarlanmalıdır.

Bazı uygulamalar, uç noktalar arasındaki gecikme örneklenmeden önce bir başlangıç gecikme tahmini gerektirir. Örneğin, bir yeniden gönderme zamanlayıcısı tetiklenirken, uç noktalar arasındaki gecikme örneklendirilmeden önce gönderilen mesajları korumak için bir başlangıç değeri gereklidir. Bu başlangıç gecikme tahmini, verilen uygulama için mümkün olduğunca koruyucu (büyük) olmalıdır. Örneğin, bir yolun gecikmesi hakkında hiçbir bilgi yoksa, TCP başlangıç Yeniden Gönderme Zaman Aşımı (RTO) değerinin en az 1 saniye olarak ayarlanmasını gerektirir. UDP uygulamaları da benzer şekilde başlangıç gecikme tahmini olarak 1 saniye kullanmalıdır. 1 saniyeden daha kısa değerler problemli olabilir [4,5].

3.2 Düşük Veri Hacimli Uygulamalar İçin Tıkanıklık Kontrolü

TCP ve diğer tıkanıklık kontrol şemalarının kısa UDP datagramları için etkili olmadığını ve bu nedenle bu tür uygulamaların tıkanıklık kontrolünü sağlamak için kendi yöntemlerini kullanmaları gerektiğini açıklar. Bu tür uygulamaların herhangi bir zaman diliminde hedefleriyle yalnızca birkaç UDP datagramı

değiştirdikleri durumda, tıkanıklık kontrol mekanizmalarının uzun süreli iletimler için etkili olduğu için ağda az bir fayda sağlarlar. Bununla birlikte, bu tür uygulamalar hedefleriyle iletişim kurarken ortalama bir RTT başına bir UDP datagramından fazlasını göndermemeleri gerektiğini belirtir. Ancak bazı uygulamalar bir hedef için güvenilir bir RTT tahmini yapamazlar. Bu uygulamalar RTT'yi ölçmek için protokol zamanlayıcılarını kullanamazlar. İki durum belirlenebilir:

İlk durum, istatistiksel olarak doğru bir RTT tahmini yapmak için yeterli sayıda UDP datagramı değişimi yapmayan ancak iletişim güvenilirliğini izleyebilen uygulamaların durumudur (Bölüm 3.3). Bu tür uygulamalar, paketler kaybolduğunda üstel olarak azaltılan bir önceden belirlenmiş iletim aralığı kullanabilirler. TCP, 1 saniyelik bir başlangıç değerini belirtir, bu değer UDP uygulamaları için de önerilir. Bazı düşük veri hacimli uygulamalar, örneğin SIP ve General Internet Signaling Transport (GIST), 500 ms bir aralık kullanırken, daha kısa değerler birçok durumda sorunlu olabilir. Önceki durumdaki gibi, başlangıç zaman aşımı, Bölüm 3.1.1'de belirtildiği gibi maksimum zaman aşımı değildir.

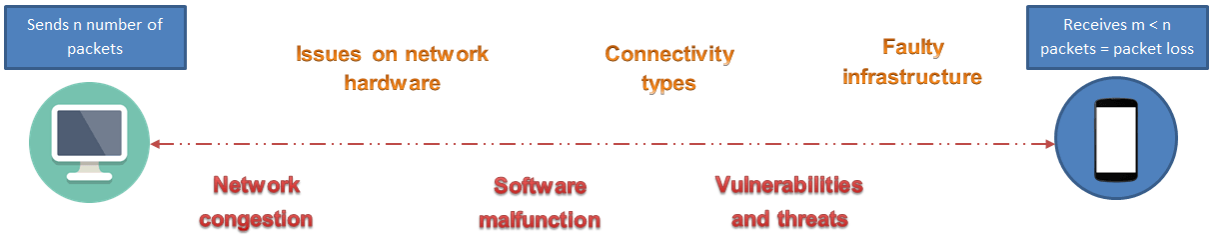
Birçok uygulama, hedef ile geri dönüş trafiği göndermeyen durumlarda bir RTT tahmini yapamaz. Bu gibi durumlarda, bu uygulamalar 3 saniyede bir UDP verisi göndermemeli ve mümkün olduğunda daha az agresif bir oran kullanmalıdır. Daha kısa süreler birçok durumda sorunlu olabilir. Bu durumda gönderim oranı, paket kaybı yani tıkanıklığı tespit edememesi ve uygulamanın yükü azaltmak için üstel geri çekilmeyi gerçekleştirememesi nedeniyle önceki durumlardan daha korumacı olmalıdır [4,5].

3.3 UDP Bidirectional Communications Protokolü Olarak Kullanıldığında Tıkanıklık Kontrolü

İki yönlü iletişim kuran uygulamalar, iletişimin her iki yönü için de tıkanıklık kontrolü kullanmalıdır. Örneğin, istek-yanıt tarzı bir istemci-sunucu uygulaması için istemciler, sunucuya isteklerinin iletimini tıkanıklık kontrolü yapmalıdır ve sunucu da yanıtlarının iletimini tıkanıklık kontrolü yapmalıdır. İleri ve ters yönlü tıkanıklık bağımsızdır ve bir uygulama hem ileri hem de ters yönde tıkanıklığı bağımsız olarak algılamalı ve yanıtların tam round-trip yolu boyunca onaylanmasına dayanarak yeni ve yeniden iletilen istekleri sınırlandırmalıdır [4,5].

3.4 UDP ve Paket Kaybı

Diyelim ki gönderici, bir alıcıya n sayıda paket gönderiyor. Alıcı, m sayıda paketi alıyor. Eğer $m < n$ ise, ağda paket kaybı meydana gelmiştir. UDP protokolü, donanım ve yazılım sistemleri de dahil olmak üzere ağ yığınına tamamen güvenir. UDP'de paket kaybına altı ana sebep vardır: ağ tıkanıklığı, donanım sorunu, yazılım hatası, bağlantı sorunu, zafiyetler ve hatalı altyapı.



Şekil 3.4.1 Paket Kaybı

UDP'de paket kaybının başlıca nedeni ağdaki tıkanıklıktır, çünkü her iletişim ağı bir akış sınırına sahiptir. Örneğin, ağ tıkanıklığı, belirli bir yolda izin verilen araç sayısını aşmanın trafikte yavaşlamaya veya zirve saatlerinde durmaya neden olduğu bir trafik sıkışıklığına benzer.

Ağdaki tıkanıklık, bir paketin hedefe ulaşması için geçen süreyi artırabilir. Bu nedenle, ağın gecikmesini artırabilir. Ayrıca, sunucu paketleri almak için belirli bir süre bekler. Yüksek gecikme nedeniyle, belirtilen sürede paketler gelmezse bağlantı zaman aşımı sorunuyla karşılaşabiliriz.

Ayrıca, ağ trafiği yüksek bağlantı eşiğine ulaştığında, yüksek gecikme nedeniyle UDP paketlerinin iletimi kesintiye uğrar. UDP uygulamalarının atılan paketleri kurtarmak veya yeniden göndermek için yapılandırılmadığı için paket kaybı meydana gelir [6].

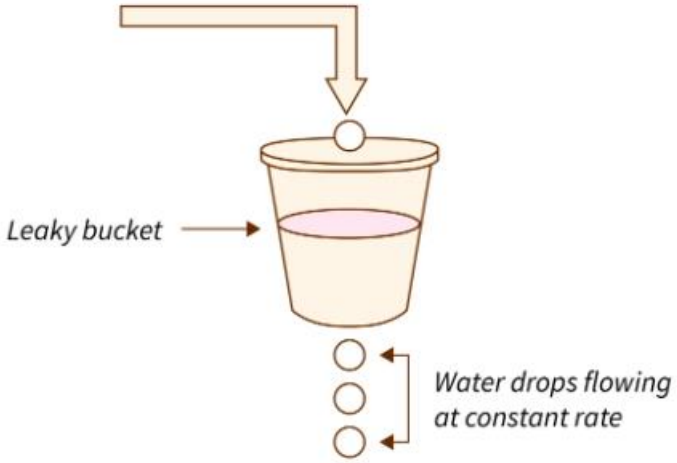
3.5 NS2 Kullanarak UDP Protokolünde Tıkanıklık ve Akış Kontrolünün Analizi

UDP (Kullanıcı Veri Protokolü), veri transferinden önce bir bağlantı kurulmasına gerek olmayan bağlantısız bir taşıma katmanı protokolüdür. UDP, minimum taşıma katmanı işlevselliği ve garanti edilmeyen veri paketi teslimi sağlayarak uygulamalara doğrudan ağ katmanına erişim imkânı sağlar. Çoklama/çoklamadan çıkarma işlevleri ve hafif hata kontrolü sağlar ancak IP paketlerine hiçbir şey eklememektedir. [5] Uygulama geliştiricileri UDP'yi bir taşıma katmanı protokolü olarak kullandığında doğrudan uygulama ve ağ katmanı arasında iletişim kurulur. UDP, veri paketlerini bir uygulama işleminden alır, çoğaltma/çoğaltmadan çıkarma işlemi için kaynak ve hedef bağlantı noktalarını bağlar ve bunlarla birlikte hata kontrolü ve uzunluk bilgi alanlarını ekler ve ardından UDP paketlerini bir ağ katmanı paketine kapsüler ve kapsüllenmiş paketleri alıcı ana bilgisayarda teslim eder. Bir UDP paketi alıcı ana bilgisayara geldiğinde, paket başlığındaki hedef bağlantı noktası alanı tarafından tanımlanan alıcı UDP ajanına daha da iletilir. NS2, UDP protokolünde tıkanıklık denetimi algoritmalarının geliştirilmesi ve test edilmesi için bir simülasyon aracı olarak kullanılabilir [7].

3.6 Leaky Bucket Algoritması

Leaky bucket algoritması, çoklu paketlerin geçici olarak depolandığı ve bu paketlerin gönderimi için gönderen ve ağ arasında belirlenen sabit bir hızla ağa gönderildiği bir tıkanıklık kontrolü yöntemidir. Bu algoritma, veri ağlarında trafik şekillendirme yoluyla tıkanıklık kontrolünü uygulamak için kullanılır. Algoritmayı anlamak için önce bir damlayan kova benzetmesini tartışalım.

Alt kısmında küçük bir delik bulunan bir kovayı hayal edin. Şimdi kovaya rastgele aralıklarla su döküldüğünü düşünün. Her aralıkta, kovaya dökülen su miktarı sabit değildir. Şimdi kovada ne kadar su olduğunun bir önemi yoktur, su delikten sabit bir hızla çıkar. Daha net bir anlayış için aşağıdaki resme bakın.



Şekil 3.6.1 Leacky Bucket

Su damlacıklarının dışarı sızması hızı (sızıntı hızı) içerideki su miktarından bağımsızdır. Eğer kova dolarsa, dökülen su kaybolur. Aynı fikir, veri paketleri için de uygulanabilir. Her bir ağ arabiriminde bir sızıntılı kova olduğunu düşünelim. Şimdi, gönderici paketleri iletmek istediğinde, paketler kovaya atılır. Bu paketler ağ arabirimindeki kovada birikir. Eğer kova doluysa, paketler kova tarafından reddedilir ve kaybedilir. Bu kova sabit bir hızda sızar. Bu, paketlerin sabit bir hızda ağa iletilmesi anlamına gelir. Bu sabit hız Sızıntı Oranı veya Ortalama Hız olarak bilinir. Bu şekilde, patlayıcı trafik sızıntılı kova tarafından düzgün, sabit trafik haline dönüştürülür. Farklı aralıklarla paketleri sıraya alıp serbest bırakmak, ağ tıkanıklığını azaltmaya ve genel performansı artırmaya yardımcı olur. Bu tasarım, özellikle taşıma ve ağ katmanında, ana işletim sistemi içinde simüle edilebilir. Sızıntılı kova algoritması özünde bir tıkanıklık kontrol mekanizmasıdır. Ağa giren trafik şeklini kontrol eder. Her bir ana bilgisayar sabit bir hızda paket gönderdiği için, ağdaki tıkanıklığı azaltmada yardımcı olur. Trafik şekillendirme donanım seviyesinde uygulanır ve gereksinimlere bağlı olarak birçok algoritma kullanılır. Sızıntılı kova algoritması ana bilgisayar işletim sisteminde uygulanır. Veri paketlerini geçici olarak depolayarak ve Sızıntı Oranına bağlı olarak düzenli aralıklarla ağa göndererek dalgalı trafikleri düzgün hale getirir. Leaky Bucket algoritması UDP protokolü için de kullanılabilir. Ancak, bu algoritma genellikle TCP trafiğinde kullanılır. Bunun sebebi, TCP trafiğinin daha yüksek bir güvenilirlik ve doğruluk seviyesine ihtiyaç duymasıdır. UDP protokolü ise daha az güvenilir ve doğruluğu daha düşük bir protokoldür ve paket kaybına daha toleranslıdır. Bu nedenle, Leaky Bucket algoritması genellikle TCP trafiğinde kullanılsa da, UDP trafiğinde de kullanılabilir [8].

3.7 Token Bucket Algoritması

Token Bucket algoritmasına dayalı, IPv4 adreslerine göre belirli jetonları sunan bir UDP sunucusu uygulamasını açıklar; önceki oran sınırlayıcıya göre daha iyi performans sunar. Kovalar bellekte saklanır, böylece mariadb çözümüne göre daha hızlı erişim sağlanır. Web uygulamaları ve betikler, istemci IP adreslerine dayalı talepleri kısıtlamak ve oran sınırlamak için udp sunucusuna jetonlar için sorgu yapabilirler.

Hedefler ve Gerekçe Orijinal php oran sınırlayıcısının iyileştirilebilecek birkaç zayıf yönü vardır. Erişim oranını sınırlandırmak için basit bir sayaç yaklaşımı kullanır ve takip sayıcılarını depolamak için bir mariadb veritabanına güvenir. Bir DBMS'nin (veritabanı yönetim sistemi) kullanımı, performans ve kaynak kullanımı üzerinde bir etkiye sahiptir. Bir oran sınırlayıcısı bir veritabanının tüm özelliklerine ihtiyaç duymaz ve amacımız için kalıcı disk depolamasına ihtiyaç duymaz. Oran sayıcıları doğrudan bellekte saklanabilir.

Basit erişim sınırlandırması sayaç yaklaşımı pürüzsüz ve düzenli bir sınırlandırma sunmaz. Diyelim ki 60 istek / dakika sınırı olan bir sayaç kullanıyoruz, 60 erişim dakikanın ilk saniyesinde gelirse, sayaç ilk saniyede dolacaktır. Kalan 59 saniye boyunca hiçbir erişim izin verilmez. Bu, erişimlerin bir aralığın başlangıcında bir araya toplandığı pürüzlü sınırlamalara neden olabilir. Token Bucket algoritmasının kullanımı daha düzenli bir sınırlama sağlayabilir.

Orijinal php oran sınırlayıcısının geliştirilmesi için yer vardır. Bellek depolama alanı yanı sıra jeton kova algoritmasını kullanan daha hızlı ve verimli bir sistem oluşturulabilir.

Token Bucket algoritması için kısaca bir açıklama yapabiliriz. Belirli bir kapasitesi olan bir kova düşünelim, bu kovada tokenlar saklanır. Tokenlar belirli bir kaynağa erişim sağlamak için kullanılır ve erişim verildiğinde tüketilir. Tokenlar sabit bir oranda yenilenir, kova kapasitesinin üstündeki herhangi fazla token atılır. Başlangıçta, kova dolu olduğunda token tüketimi "patlayıcı" olabilir, başlangıçtaki depodan ve yenilenen tokenlardan hizmet edebilir.

"Patlayıcı" aralık, yenileme hızını aşan erişimlere izin verir ve kova zaman aralığında sınırlı maksimum oranı (Kapasite / Zaman aralığı) + Zaman aralığındaki yenilemeleri sağlayabilir. Burada uygulamamızda, hız sınırlaması başladığında kovayı maksimum token kapasitesinde bırakmamıza izin veriyoruz. Tokenlar kovadan tüketildikçe, sabit bir oranda yeniden doldurulur. Bu nedenle, istekler geldiğinde başlangıçta patlayıcılık olacaktır, ancak daha sonra yenilenme hızına düzleşecektir.

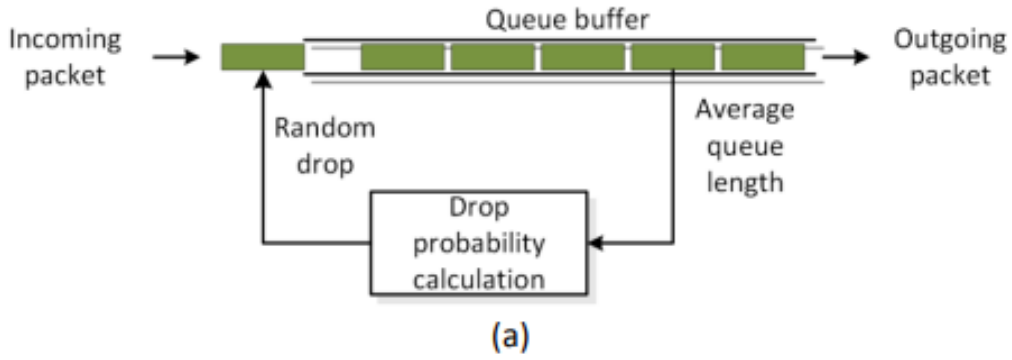
Örneğin, bir IP adresi başına dakikada maksimum 10 çağrıya izin veren bir web tabanlı Json API'yi hız sınırlamak için, her bireysel IP adresi için 10 token kapasiteli bir kova tahsis edebiliriz. Bir IP adresi Json API'yi aradığında, kovasından bir token kullanılır. Tüm tokenlar tüketildiğinde, erişim reddedilir. Her kova içindeki tokenlar, 6 saniyede bir 1 token oranında yenilenebilir. Bu yenileme hızı,

başlangıçtaki 10 token kullanıldıktan sonra, IP adresinin Json API'ye sadece her 6 saniyede bir başarılı bir şekilde erişebileceği anlamına gelir.

Başlangıçta erişim patlayıcıdır, çünkü her IP kovasına başlangıçta 10 token verilir. Bir IP adresi ilk dakikada 20 çağrı yapabilir (20'ye kadar patlayıcı), 10 başlangıç tokeni ve sonraki 1 tokenin (dakikada 10) her 6 saniyede bir yenilenmesi nedeniyle. İkinci dakikadan itibaren, düzenli bir hıza ulaşacaktır, 6 saniyede bir token. Aşağıdaki diyagram, Token Bucket Algoritmasını göstermektedir.

Token Bucket algoritması, UDP protokolü üzerinde de kullanılabilir. UDP protokolü, paket kaybı ve sırasız teslimatlar gibi sorunlara neden olabileceği için, token bucket algoritması UDP trafiğini sınırlamak için önemli bir araçtır [9].

3.8 Random Early Detection RED Yaklaşımı



Şekil 3.8.1 RED Algorithm

Şekil 1(a), bir yönlendiricinin tamponunun Random Early Detection (Rastgele Erken Tespit) tarafından yönetildiği senaryoyu göstermektedir. RED, ortalama kuyruk boyutunu hesaplamak için üstel hareketli bir ortalama ile düşük geçiren bir filtre kullanır. Daha sonra, ortalama kuyruk boyutu iki eşik değeri olan minimum ve maksimum eşik değerleriyle karşılaştırılır. Sonuç olarak, paket düşme olasılığı Şekil 1(b)'de gösterilen fonksiyon tarafından belirlenir. Ortalama kuyruk boyutu minimum eşik değerinden küçük olduğunda hiçbir paket düşürülmez. Ortalama kuyruk boyutu maksimum eşik değerinden büyük olduğunda, her gelen paket işaretlenir ve bu nedenle düşürülür. Ortalama kuyruk boyutu minimum ve maksimum eşik değerleri arasında olduğunda, her gelen paket düşürme olasılığıyla işaretlenir. Böylece, RED'in iki ayrı algoritması vardır. İlk olarak, kuyruk boyutu ortalamasını hesaplayan algoritma, kuyrukta izin verilen dalgalanma derecesini belirler. İkinci olarak, paket işaretlenme olasılığını hesaplayan algoritma, mevcut tıkanıklık seviyesine göre ağ geçidinin hangi sıklıkla paketleri işaretleyeceğini veya düşüreceğini belirler. Ağ geçidinin, paketleri ortalama kuyruk boyutunu kontrol etmek için işaretleyerek küresel senkronizasyonu önlemek için paketleri eşit aralıklarla işaretlemesi amaçlanmaktadır. Random Early Detection (RED) yönetimi, ağ trafiğinin daha iyi bir şekilde yönetilmesine yardımcı olur. Özellikle, User Datagram Protocol (UDP) trafiği için önemlidir çünkü UDP, kayıpsız bir protokol olarak tasarlanmıştır ve ağda aşırı yükleme olduğunda paket kaybı

yaşanabilir. RED, ağda yaşanan aşırı yüklenmeyi algılayarak, paket kaybını azaltmak için ağ trafiğini dengeleyebilir ve paket kaybını azaltarak UDP trafiği için daha iyi bir performans sağlayabilir. Bu nedenle, özellikle ağ trafiği yönetimi için kullanılan router ve gateway cihazlarında, RED gibi akıllı kuyruk yönetimi algoritmaları kullanmak önemlidir [10].

3.9 ECN(Explicit Congestion Notification) Yaklaşımı

ECN, Explicit Congestion Notification (Açık Tıkanıklık Bildirimi) algoritmasının kısaltmasıdır. Bu, yavaş başlatma/yeniden başlatma ve AIMD gibi tıkanıklık kontrolü algoritmaları gibi bir tıkanıklık kontrol algoritması değildir. Sadece tek bir görevi vardır ve yönlendiricilerde oluşan tıkanıklıkları gönderene bildirmektir. Bu nedenle, ECN, tıkanıklık bildirimi veya tıkanıklık sinyal algoritması olarak adlandırılır. Tıkanıklığı önlemek için ilgili önlemlerin alınabilmesi için gönderene tıkanıklık hakkında bilgi verir. ECN, RFC 3168'de tanımlanan bir Tıkanıklık Sinyal Mekanizmasıdır. 1999 yılında ortaya çıktı ve 2001'de nihai hale geldi. TCP başlığındaki iki bit ve IP başlığındaki iki bit kullanır. İşaretlenen paketleri, işaretleme = başlıklarda bir bit çevirme olarak işaretler. ECN (Explicit Congestion Notification) gibi mekanizmalar, ağdaki tıkanıklığı tespit etmek ve önlemek için UDP paketlerini de izleyebilir. Bu nedenle, UDP, hızlı ve gerçek zamanlı veri transferi için önemli olduğu gibi ağ trafiğinin yönetimi için de önemlidir [11].

3.10 Rate-Based Congestion Control (RBCC) Yaklaşımı

Hız tabanlı tıkanıklık kontrol mekanizmaları UDP ile kullanılabilir, ancak TCP ile olduğu kadar etkili olmayabilir.UDP, varsayılan olarak herhangi bir tıkanıklık kontrol mekanizması sağlamayan bağlantısız bir iletişim protokolüdür. Bu, UDP uygulamalarının ağ koşullarını göz önünde bulundurmadan istedikleri kadar veri göndermeleri anlamına gelir. Ağ tıkanırsa, UDP paketleri atılabilir veya geciktirilebilir, bu da kötü performansa ve potansiyel veri kaybına neden olabilir. Hız tabanlı tıkanıklık kontrol mekanizmaları, UDP paketlerinin gönderim hızını sınırlayarak tıkanıklığı hafifletmeye yardımcı olabilir. Örneğin, bir uygulama gönderim hızını önceden belirlenmiş bir değere sınırlandırmak için bir jeton kovası algoritması kullanabilir. Ancak, bu yaklaşımın bazı sınırlamaları vardır. İlk olarak, hız tabanlı tıkanıklık kontrol mekanizmaları yalnızca gönderim hızını azaltabilir, artıramaz. Bu, ağ koşulları iyileşirse uygulamanın ek kapasiteden yararlanamayabileceği anlamına gelir. İkincisi, hız tabanlı tıkanıklık kontrol mekanizmaları göndericiye ağ durumu hakkında herhangi bir geri bildirim sağlamaz. TCP'de, tıkanıklık kontrol algoritmaları atılan paketler ve diğer sinyallerden gelen geri bildirimleri kullanarak gönderim hızını dinamik olarak ayarlayabilir. Bunun aksine, UDP için hız tabanlı tıkanıklık kontrol mekanizmaları genellikle ağ koşulları hakkında statik varsayımlara dayanır.

Akış gönderim hızını kontrol etmek için oran bilgisinin kullanılma fikri yeni değildir. Padhye ve arkadaşları, gözlemlenen kayıp olaylarına dayalı olarak gönderim hızını ayarlayan tükenme kontrolü için denklem tabanlı bir yaklaşım öneren ilk kişilerdi. Ke ve arkadaşları, çoklu paket kayıplarından kaçınmak için paketlerin mevcut hızına göre gönderilmesi gerektiğini önerdiler. Ancak, RTT'nin kesin

tahminlerine ihtiyaç duyarlar ve hücresel veri ağlarında link kalitesinin gerçekten doğru göstergeleri değillerdir. Rate bilgisini kullanarak TCP tıkanıklık kontrolü gerçekleştirmek için başka bir öneri RATCP'dir, ancak bu bizim bağlamımızda pratik bir yaklaşım değildir, çünkü ağın TCP kaynağına mevcut hızı geri bildirmesi gereklidir. TCP hız tabanlı Pacing (TCP-RBP) adlı benzer bir rate teknik, boşta durumdan yavaş başlangıç sonrası cwnd'yi artırmak için kullanılır. Ancak, bant genişliği tahminleri, RTT'yi bir parametre olarak kullanan TCP Vegas'a benzer ve tek yönlü gecikmeler değildir [12].

3.11 Quality of Service – QoS

QoS (Kalite Hizmeti), çalışma alanları veya evlerindeki önemli uygulamaların optimum performansını garanti altına alarak, ağ trafiğini önceliklendirebileceğiniz, jitter'ı kontrol edebileceğiniz ve gecikmeyi azaltabileceğiniz bir sistemdir. Ayrıca, QoS sayesinde belirli uygulamaların ağ performansını optimize edebilir ve ağlarının bit hızı ve paket hızı hakkında görünürlük elde edebilirler. Son olarak, paketlerin iletim gecikmelerinden kaçınmak için ağlarında paketlerin nasıl yönlendirileceğini yapılandırabilirler.

QoS, VoIP, video konferans ve online oyun gibi kritik uygulamaların en iyi performansını garanti altına almak için çeşitli teknikler kullanır. Bu teknikler arasında kaynak rezervasyon protokolü (RSVP), sıralama ve trafik işaretleme bulunur. UDP (User Datagram Protocol) ve QoS (Quality of Service) arasındaki ilişki, QoS'nin ağ trafiğinin önceliklendirilmesi ve yönlendirilmesi için kullanıldığı durumlarda özellikle önemlidir. QoS, ağ trafiğini önceliklendirerek, bant genişliğini belirli uygulamalara tahsis eder ve ağın farklı kısımlarında farklı QoS seviyeleri sağlayarak, öncelikli veri trafiğinin daha az gecikme ile iletilmesini sağlar. Bu sayede gerçek zamanlı uygulamaların gereksinimlerine uygun bir ağ hizmeti sağlanır ve UDP ile transfer edilen verilerin gecikmesi azaltılır [13].

4.SONUÇ

UDP protokolünün tıkanıklık kontrolü yapmadığı ve veri bütünlüğünü garanti etmediği görülmektedir. Ancak tıkanıklık bir sorundur ve çözülmesi gerekmektedir. Uygulama katmanında tıkanıklık kontrolü yapılarak bu sorunlar önlenebilir. Büyük veri transferleri için tıkanıklık kontrolü için farklı algoritmalar kullanılabilir ve düşük veri hacimli uygulamalar için farklı bir yaklaşım gerekebilir. Bidirectional iletişim için de tıkanıklık kontrolü yapılması gerekmektedir. Paket kaybı durumunda ise uygulama katmanı bu kayıpları telafi edebilir. Leaky Bucket, Token Bucket, RED, ECN ve RBCC gibi farklı yaklaşımlar tıkanıklık kontrolü için kullanılabilir. Ayrıca, QoS ile birlikte tıkanıklık kontrolü daha etkili bir şekilde sağlanabilir.

Sonuç olarak, ağ trafiğinin yönetimi, veri transferi hızını, ağ kaynaklarının kullanımını ve performansı etkiler. Tıkanıklık kontrolü, ağda veri transferini yönetirken performansı optimize etmek için kullanılan önemli bir mekanizmadır. UDP, hızlı veri transferi için kullanılan bir protokol olmasına rağmen, tıkanıklık kontrolü sağlamadığı için, uygulama katmanında ek tıkanıklık kontrolü mekanizmalarının kullanımı gereklidir.

5. KAYNAKÇA

1. J.F. Kurose and K.W. Ross, "Computer Networking: A Top-Down Approach," 7th ed. Pearson, 2017.
2. "User Datagram Protocol (UDP)," Khan Academy. [Online]. Available: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/user-datagram-protocol-udp. [Accessed: May 14, 2023].
3. IR. (2022). Network Congestion: Causes, Impacts, and How to Fix It. Erişim tarihi: 14 Mayıs 2023, https://www.ir.com/guides/network-congestion.
4. Maleki, S., & Shahbazian, A. (2017). Network congestion control: Managing internet traffic. In Proceedings of the 3rd International Conference on Frontiers of Educational Technologies (ICFET 2017) (pp. 117-120).
5. RFC 8085: User Datagram Protocol (UDP) Checksum Complement in the Internet Protocol Version 6 (IPv6). (2017). Retrieved from https://www.rfc-editor.org/rfc/rfc8085.html#section-3.1.3
6. "Packet Loss in UDP," Baeldung, 2019. [Online]. Available: https://www.baeldung.com/cs/udp-packet-loss#packet-loss-in-udp. [Accessed: May 14, 2023].
7. J. Muruganandam, S. Lakshmi, and G. Venkatesan, "Analyzing the Congestion and Flow Control in UDP Protocol Using NS2," International Journal of Computer Science and Mobile Computing, vol. 5, no. 5, pp. 311-318, May 2016. DOI: 10.1109/ICACCI.2015.7275667
8. Scaler.com. (n.d.). Leaky Bucket Algorithm - Scaler Academy. [online] Available at: https://www.scaler.com/topics/leaky-bucket-algorithm/ [Accessed 14 May 2023].
9. Wong, W. K., "Token Bucket Rate Limiter," Nighthour.sg, 2017. [Online]. Available: https://www.nighthour.sg/articles/2017/token-bucket-rate-limiter.html. [Accessed May. 14, 2023].
10. "Lab 16: Network Time Protocol (NTP) and Precision Time Protocol (PTP)," University of South Carolina, Department of Computer Science and Engineering, 2013. [Online]. Available: http://ce.sc.edu/cyberinfra/workshops/Material/NTP/Lab%2016.pdf. [Accessed: May. 14, 2023].
11. Mukesh Singh, "What is ECN (Explicit Congestion Notification)?", GeeksforGeeks, Nov. 2019. [Online]. Available: https://www.geeksforgeeks.org/what-is-ecnexplicit-congestion-notification/. [Accessed: May 14, 2023].

12. Leong, W.K. (2010). A Rate-based TCP Congestion Control Framework for Cellular Data Networks. B.Comp. (Hons.) thesis, National University of Singapore.
13. H. Kılıç, "Understanding Quality of Service," Section.io Engineering Education. [Online]. Available: https://www.section.io/engineering-education/understanding-quality-of-service/. [Accessed: May 14, 2023].