Çekişme Temelli Ortam Erişimi Algoritmaları Dilim Atama İhtimalleri Karşılaştırması

Hasan Ferit Enişer[1], İlker Demirkol[2]

[1] Boğaziçi Universitesi, Bilgisayar Mühendisliği Bölümü

[2] Univ. Politecnica de Catalunya, Telematics Engineering Department

hasan.eniser@boun.edu.tr , ilker.demirkol@entel.upc.edu

Özet: Çekişme temelli (Contention-based) ortam erişimi, kablosuz iletişim sistemlerinde yaygın olarak kullanılan bir yaklaşımdır. Bu ortam erişimi şeklinde, düğümler daha önceden atanmış zaman ya da frekans yerine, koordinasyonsuz bir şekilde ortama çıkmaya calışır. Taşıyıcıyı dinleyen çoklu erişim (Carrier Sense Multiple Access (CSMA)) ve çekişme penceresi (çekişme penceresi-CW) yaklaşımları, çekişme temelli bircok protokol tarafından kullanılmaktadır. Bu protokoller arasında IEEE 802.11/WiFi, IEEE 802.15.4 /ZigBee, IEEE 802.16/WiMax gibi standartlar yanında, S-MAC[1], B-MAC gibi görece yakın zamanda önerilmis kablosuz algılayıcı ağ protokolleri de bulunmaktadır. Her protokol farklı bir algoritmayla düğümleri çekişme penceresine atamayı önerir. Biz çalışmamızda bu protokollerin CW adaptasyon ve CW dilim atama kısımlarını, enerji, gecikme, çekişme penceresi büyüklüğü gibi başlıklar altında nasıl sonuçlar verdiğini inceledik. Bunun yanında önerilen metotlar üzerinde hassaslık analizi de yaptık. Amacımız tüm bu çalışmalarla hangi metodun hangi koşullarda kullanılması gerektiğinin saptanması ve belki bir kesin kazananın belirlenmesiydi.

Anahter Kelimeler: Ortam Ulaşım Kontrolü, Ortam Ulaşım Kontrolü Protokolleri, İkili Üstel Geri Çekilme, Birbiçimli Geri Çekilme, Sift

1.Giriş

Kablosuz ağlarda veri paketi yollama kavaramını kısaca özetleyecek olursak; paket yollamak isteyen her bir düğüm cekişme penceresinden bir dilim seçer ve kendi dilimi gelene kadar kanalı dinlemeye başlar. Bir düğüm, kendi dilimi gelene kadar herhangi bir iletim algılamazsa, kendi dilimi zamaninda paketini yollar. Dolayısıyla pencerede dağıtık bir şekilde seçilen dilimlerden ilk seçilen dilimde iletim olur. Ancak, ilk seçilen dilim birden fazla düğüm tarafından seçilmişse, bir çarpışma meydana gelir. Çarpışmadan sonra, bütün düğümler belli bir süre (time-off duration) bekleyip, ardından yeni bir çekişme penceresi başlatır.[2]

Toplam gecikme iki parçaya ayrılabilir:

1) Başarısız (çarpışma ile sonuclanan) cekişme penceresindeki gecikme.

2) Başarılı çekişme penceresindeki ilk seçili dilime kadar olan bekleme gecikmesi.

Çekişme penceresinin boyutunu büyütmek, çarpışma ile sonuçlanma ihtimalini düşürmek için bir seçenektir. Çekişme penceresinin boyutu büyüdüğünde, düğümlerin aynı dilime atanma ihtimali de düşeceğinden çekişme penceresinin başarısız sonuçlanma ihtimali de azalır.

Fakat, çekişme penceresinin boyutunu çok fazla artırmak, ikinci maddede bahsettiğimiz beklemeden doğan gecikmenin artmasına sebep olur. Daha büyük bir çekişme penceresinde, ilk seçili dilime rastlamak için daha fazla beklemek gerekir. Buradan ortada bir ödünleşim/

trade-off olduğunu anlıyoruz. Büyük boyutlu çekişme penceresi bekleme gecikmesini artırırken, küçük boyutlu birçekişme penceresi seçmek dağıtımın başarısızlıkla sonuçlanmasına sebep olabilir. İşte her bir protokol bu ödünleşimi farklı yollarla optimize etmeye çalışır.

İncelediğimiz algoritmalar sırayla; UB-Uniform Backoff/Birbiçimli Geri Çekilme, BEB-Binary Exponential Backoff/İkili Üstel Geri Çekilme, Sift ve yeni bir yaklaşım olan BEB+Sift'tir. Bu bildirinin ikinci kısmında bu algoritmaları detaylı bir biçimde açıklayacağız. Üçüncü kısımda ise bahsettiğimiz algoritmaları hangi başlıklar altında karşılaştırdığımızdan bahsedeceğiz. Bu başlıklar altında nasıl sonuçlar elde ettiğimizi ise bir sonraki bölümde açıklayacağız. Son bölümde ise çalışma hakkında genel bir özet geçeceğiz.

2. İncelenen Algoritmalar

2.a.SİFT

Sift algoritması, 2003 yılında Tay, Balakrishnan ve Jamieson tarafından önerilmiştir.[3, 4] Sift, temelde çekişme penceresi büyüklüğünü sabit tutarak çarpışma ihtimalini en aza indirmeye çalışan bir dilim seçme algoritmasıdır..

Sift algoritması, özellikle çarpışmadan doğan gecikmeyi en aza indirmek için, basitçe her bir düğümün çekişme penceresi üzerine bir dilim seçme ihtimalinin çekişme penceresi diliminin endeksiye birlikte artmasını önerir. Fakat bu artış doğrusal değil üstel bir artıştır. Böylece; çekişme pencerelerinin baştan itibaren dinlendiğini göz önünde bulundurursak düşük endeksli dilimleri seçen daha az düğüm olacağı için çarpışma ihtimali de azalır.

2.B. İkili Üstel Geri Çekilme (Bınary Exponentıal Backoff)

Bu metot da Sift gibi özellikle çarpışmadan doğan gecikme için farklı bir yol önerir. Sift algoritmasında çekişme penceresi sabit tutulurken bu algoritmada çekişme penceresi boyutu her bir çarpışmadan sonra iki katına çıkarılır. Böylece düğümlerin aynı dilimlere yerleşme ihtimali azalır. Örneğin çekişme penceresi büyüklüğü n iken, bir çarpışma saptandığında büyüklük 2*n olur. Böylece, ortamdaki kanala çıkmaya çalışan düğüm sayısı sabit olduğundan çarpışma durumunun ortaya çıkma olasılığı düşer. Fakat çekişme penceresi çok büyüdüğünden bekleme gecikmesi de buna bağlı olarak artar.[5] IEEE 802.11x /WiFi, IEEE 802.15.4 /ZigBee, IEEE 802.16/WiMax gibi çok yaygın olarak kullanılan setlerin de temelinde ikili üstel geri çekilme yatmaktadır. [3] nolu çalışmadan Sift ile 802.11'in bir karşılaştırmasına ulaşılabilir

2.C. Birbiçimli Geri Çekilme (Unıform Backoff)

Birbiçimli geri çekilme algoritmasını diğerleri arasında en basiti olarak adlandırabiliriz. UB, çekişme penceresi boyutu ya da dağılım fonksiyonu üzerinde hiç bir değişiklik önermez. Çekişme penceresi büyüklüğü sabittir ve düğümler tamamen rastgele dağıtılır. Herhangi bir çarpışma durumunda düğümler aynı dağılım fonksiyonuyla(rastgele) tekrar dağıtılır çekişme penceresi büyüklüğü değişmez.Bu yaklaşım, ceşitli kablosuz algılaycı ağ protokolleri (S-MAC, ...) tarafından kullanılmaktadır.[6]

2.D İkili Üstel Geri Çekilme + Sift

Çalışmalaramızda diğer algoritmalardan daha iyi bir sonuç verebilecek bir çözüm önermeyi denedik. Bunun için İkili Üstel Geri Çekilme ve Sift'i birleştirerek yeni bir algoritma ortaya çıkardık. Sift çarpışma ihtimalini düşürmek için yeni bir dağılım fonksiyonu önerirken İkili Üstel Geri Çekilme her çarpışmadan sonra çekişme penceresi büyüklüğünü artırıyordu.

Biz ise dağılım fonksiyonu olarak Sift'in formulünü kullanıp, her çarpışmadan sonra çekişme penceresi büyüklüğünü (İkili Üstel Geri Çekilme'ta olduğu gibi) iki katına çıkarmayı düşündük. Kısaca Sift'in önerdiği algoritmayı sabit büyüklükte bir çekişme penceresinde değil de her çarpışmadan sonra büyüklüğü iki katına çıkan bir çekişme penceresinde uyguladık.

3. Karşılaştırma Parametreleri

3.A. Gecikme

Ortam Ulaşım Kontrolü (MAC) protokollerini karşılaştırmak için kullandığımız en önemli parametrelerden birisi "gecikme"dir. Gecikme sözcüğünden kast edilen anlam düğümlerin çekişme penceresine atamasından sonra, ilk veri paketinin gönderilmesi için gereken süredir.

Yukarıda bahsettiğimiz gibi; iki durumdan kaynaklanan gecikme vardır. Birisi bekleme gecikmesi, diğeri ise çarpışmadan kaynaklanan gecikmedir. Biz bu çalışmada her bir protokol için toplam gecikmeyi hesapladık ve bu değerlerle grafiklerimizi çizdik.

3.B. Enerji

Çalışmalarımızda dikkate aldığımız bir diğer karşılaştırma başlığı ise enerji idi. Çekişme penceresine yerleştirilen düğümler gönderilene kadar belli düzeyde bir enerji harcamaktadırlar. Bu enerji, hem dinleme sırasında, hem yollama esnasında, hem de çarpışma durumunda harcanır.[7] Çarpışma durumunda harcanan enerji diğerlerine nazaran daha fazladır. Biz çalışmamızda farklı protokolleri simule ederek toplamda ortalama ne kadar enerji harcandığını tespit ettik.

3.C. Çekişme Penceresi Büyüklüğü

Çekişme penceresi büyüklüğünü doğru seçmek hem gecikme hem de enerji açısından çok önemlidir. Ortamda bulunan düğüm sayısına göre optimum çekişme penceresi büyüklüğü değişmektedir. Örneğin ortamda 20 düğüm olduğunu varsayalım. Bu durumda Sift protokolü gecikme değerini çekişme penceresi büyüklüğü 30'a ayarlandığında en aza indiriyor olabilir. Dolayısıyla eğer önceliğimiz gecikme ise, ortamda 20 düğüm varken çekişme penceresi büyüklüğünü 30'da tutmamız gerekir.

Benzer bir örneği enerji için de verebiliriz. Biz benzetim yaparak her düğüm sayısı için optimum olan çekişme penceresi büyüklüğünü bulmaya çalıştık. Elbette optimum uzunluk her bir protokol için ve enerji ya da gecikme için değişmektedir.

3.D. Hassaslık Analizi

Teorik çalışmalarda optimum pencere büyüklüğü değerleri kullanılabilirken, pratikte/gerçek hayatta ortamdaki paket yollamak isteyen düğüm sayısını kesin olarak bilemeyebiliriz, en iyi ihtimalle yaklaşık değerler tahmin edebiliriz. Dolayısıyla bir protokolün optimum değerler veren çekişme penceresi büyüklüğünü, tahmin edilen veri paketi sayısına göre ayarlayıp, bu büyüklüğün gerçek veri paketi sayısı karşısında nasıl değerler verdiğini ölçümlemek önemli bir parametredir. Çünkü bu değerler protokollerin gerçekte nasıl davrandığına daha yakındır.

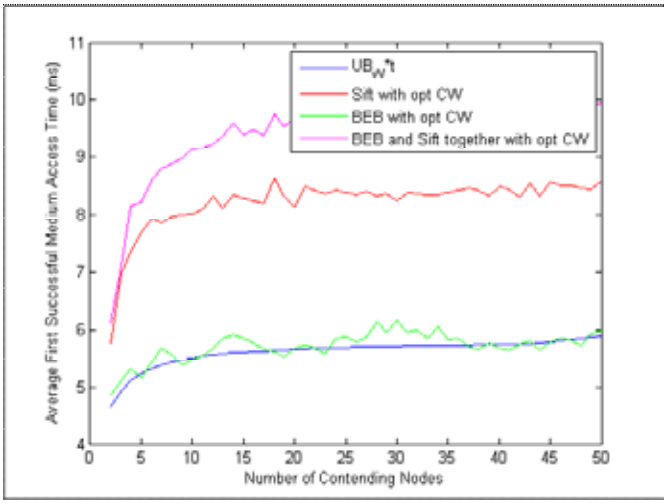
4. Karşılaştırma Sonuçları

Öncelikle benzetimin tertibinden bahsedecek olursak; ortamda en az 2 en çok 50 düğüm olduğunu varsayarak farklı çekişme penceresi boyutlarında her bir algoritmanın nasıl sonuçlar verdiğini inceledik. Örneğin Sift ve UB algoritmalarında ortamdaki (2 ile 50 arasında) her düğüm sayısı için çekişme penceresi boyutunu 3 ile 127 arasında değiştirerek en az hangi boyutta gecikme olduğunu/enerji harcandığını bulduk. Bunu da, daha kesin sonuçlar elde etmek için, aynı işlemi 10.000 kez koşup hepsinin ortalamasını alarak yaptık. BEB ve BEB+Sift benzetimlerinde ise farklı olarak çekişme penceresi boyutunu 23 ile 28 arasında değiştirdik.

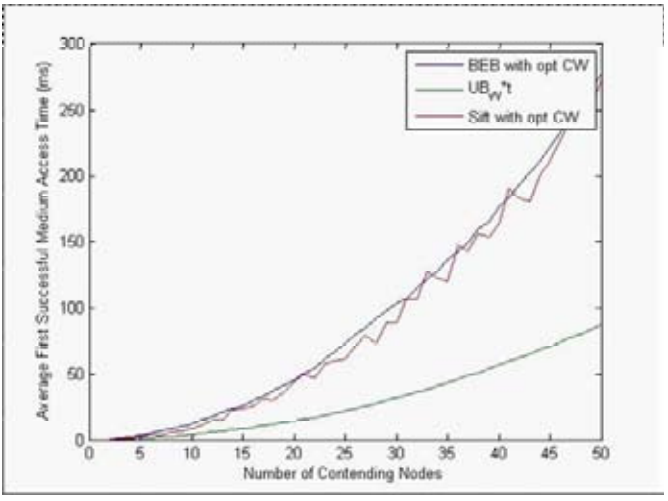
4.A. Gecikme

Aşağıdaki grafikte görüldüğü gibi, BEB ve UB arasında fazla fark yokken, Sift algoritması paket gönderimini biraz daha geç tamamlamaktadır. Bizim yeni bir yaklaşım olarak denediğimiz BEB+Sift algoritmasının ise daha büyük bir gecikme değerine sahip olduğu görülüyor. Bunun sebebi de hem BEB'in hem de Sift'in sadece çarpışmadan doğan gecikmeye karşı önlem almaları olabilir.

Gecikme



Enerji



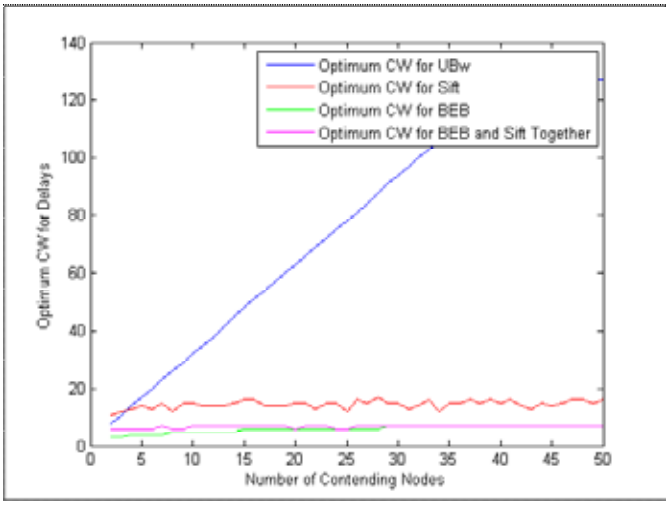
4.B. Enerji

İncelediğimiz algoritmaların enerji konusundaki davranışlarını yukarıdaki grafikten okuyabiliyoruz. Düğüm sayısı küçük rakamlardayken harcanan enerji değerleri yakın olsa da daha büyük sayılar için fark açılıyor. Buna göre, UB algoritması en az enerji harcayan algoritma. BEB ve Sift ise çok yakın miktarda enerji harcıyorlar.

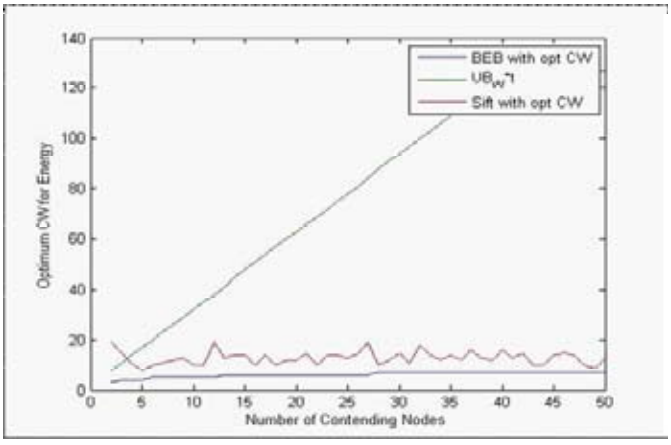
4.C. Çekişme Penceresi Büyüklüğü

Aşağıdaki ilk grafik, algoritmaların ortamdaki farklı düğüm sayıları için hangi çekişme penceresi büyüklüğünde en iyi-en küçük gecikme değerini verdiğini gösteriyor. Bir sonraki grafikten de enerji açısından optimum çekişme penceresi büyüklüğünü okuyabiliriz. Çekişme penceresinin büyüklüğü doğrudan hangi algoritmayı kullanacağımızı bize söylemez, bunun yerine belli bir düğüm sayısı için, hangi algoritmanın hangi büyüklükte daha iyi davrandığını anlatır.

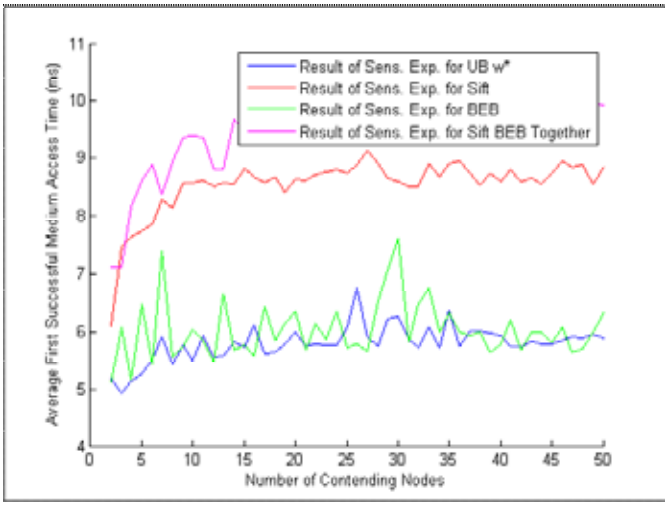
1. Grafik



2. Grafik



Hassaslık Analizi



4.D. Hassaslık Analizi

Daha önce de belirttiğimiz gibi, hassaslık analizi bu algoritmaların gerçek hayattaki kullanımı açısından çok önemli bir parametredir. Biz yukaridaki grafikte algoritmaların gecikme açısından ortamdaki düğüm sayısı-çekişme penceresi arasındaki ilişkide ne kadar hassas olduklarını gösterdik. Görüldüğü gibi bütün algoritmalar aşağı yukarı benzer hassaslıkta çalışmaktadır. Bu da gerçek hayatta hangi algoritmayla çalıştığımızın gecikme açısından bir önemi olmadığını gösteriyor.

5. Sonuç

Görüldüğü gibi çalışmalarımızı 4 parametre üzerinde yürüttük. Var olan üç çekişme penceresi yönetimi algoritmaları (UB, BEB, Sift) ve kendi yaklaşımımızı bu dört başlık altında simule ettik ve karşılaştırdık.

Amacımız özellikle enerji ya da gecikme konusunda diğerlerinden daha üstün görünen ve ortamdaki düğüm sayısı konusunda çok hassas olmayan bir algoritma belirlemekti. Fakat grafiklerde görüldüğü gibi benzetimler bize genel bir kazananın olmadığını gösterdi.

Bizim bulgularımız, karşılaştırdığımız başlıklar altında protokollerin benzer sonuçlar verdiğini gösteriyor. Dolayısıyla, bazı grafiklerde, bazı metotlar az bir farkla daha iyi görünse de, bu bir algoritmayı diğerine tercih etmek için yeterli bir sebep olmayabilir.

Fakat bu bulgular bütün metotların birbirinden farksız olduğunu söylemek için de yeterli olmayabilir. Bu çalışmanın devamı olarak, düğüm başına harcanan enerji, gecikme değerlenin hangi oranlarda çarpışmadan, hangi oranda beklemeden kaynaklandığı .. vs gibi konular da incelenirse bazı algoritmaların daha iyi sonuçlar verecebileceği görülecektir, ya da bu sonuçlara göre daha mantıklı protokoller önerilebilir.

6. Kaynaklar

[1] Ye, W., J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks", IEEE/ACM Trans. Netw., Vol. 12, No. 3, pp. 493–506, (2004)

[2] Demirkol, I. and C. Ersoy, "Energy and Delay Optimized Contention for Wireless Sensor Networks", Elsevier Computer Networks, 53, 2106–2119 (2009)

[3] Kyle Jamieson, Hari Balakrishnan, and Y.C. Tay "Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks" Third European Workshop on Wireless Sensor Networks (EWSN), Zurich, Switzerland, (February 2006)

[4] Y. C. Tay, Kyle Jamieson, and Hari Balakrishnan, "Collision-Minimizing CSMA and Its Applications to Wireless Sensor Networks" IEEE Journal On Selected Areas In Communations, Vol. 22, No. 6, 1048-1057 (August 2004)

[5] Woo, A. and D. E. Culler, "A transmission control scheme for media access in sensor networks", MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking, pp. 221–235, (2001)

[6] Demirkol, İ., "Medium Access Control Layer Performance Issues In Wireless Sensor Networks" Phd Thesis, Bogazici University, (2008)

[7] Halkes, G. P., T. van Dam, and K. G. Langendoen, "Comparing energy-saving MAC protocols for wireless sensor networks", Mob. Netw. Appl., Vol. 10, No. 5, pp. 783–791, (2005)