İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

AI	Artificial Intelligence
API	Application Programming Interface
CL	Children List
CPA	Continuous Probabilistic Algorithm
CRC	Cyclic Redundancy Check
DPA	Dicrete Probabilistic Algorithm
eCos	Embedded Configurable Operating System
FLD	Flooding
GUI	Graphical User Interface
ISM	Industrial, Scientific and Medical
MANTIS	MultimodAI NeTworks of In-situ Sensors
MEMS	Micro-Electro-Mechanical Systems
MNP	Multihop Network Programming
MOAP	Multihop Over-the-Air Programming
PA	Parent Advertisement
PDF	Probability Distribution Function
POSIX	Portable Operating System Interface
PR	Parent Request
R. D.	Rasgele Değişken
R. V.	Random Variable
τ-CA	τ-Counter Algorithm
TAP	Tree-BAsed Code Propagation inWireless Sensor Networks
TinyOS	Tiny Microthreaded Operating System
TDA	Telsiz Duyarga Ağları
WSN	Wireless Sensor Networks

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Tez çalışması boyunca bana her türlü desteği sunan, çalışmanın dolambaçlı ve bazen çıkmaz yollarından çıkmam için, yılmayan bir sabırla beni yönlendiren, danışman hocam Prof. Dr. Oya KALIPSIZ'a teşekkür ederim.

Bu tez çalışması sırasında, tecrübe ve değerli görüşleri ile beni yönlendiren, laboratuvar imkânlarından faydalanmamı sağlayan, değerli hocam Prof. Dr. Şebnem BAYDERE'ye teşekkür ederim.

Sıkıntılı anlarımda her zaman yanımda olan ve beni sabırla destekleyen aileme ayrıca teşekkür ederim.

ÖZET

Teknolojideki hızlı gelişmeler sayesinde, boyutları çok küçük, algılama, işleme ve haberleşme yeteneklerine sahip aygıtların, bir arada işbirliği içinde çalışarak, fiziksel dünyayı gözleme imkânı sunacak şekilde bir ağ oluşturmaları mümkün olmuştur. "Duyarga ağları" diye isimlendirilen bu ağlar, günümüzde bilgisayar dünyasının en hızlı gelişen ve en çok araştırma yapılan alanlarından biri haline gelmiştir. Bu ağların gelecekte, bugünkü kişisel bilgisayarlardan daha çok hayatımızın içine gireceği tahmin edilmektedir.

Duyarga ağları, hedef ortama yerleştirildikten sonra, aygıtlara fiziksel olarak erişim pek çok durumda imkânsızdır. Bu yüzden, ağ üzerinde çalışan uygulamanın değiştirilmesi ihtiyacı doğarsa, bu işlem uzaktan yapılmalıdır. Duyarga ağlarının uzaktan programlanması anlamına gelen "kod yayma" konusunda çok az çalışma yapılmıştır ve bu alan gelişmeye çok açıktır.

Bu tezde, duyarga ağları için etkin bir kod yayımı algoritması geliştirilmiştir. Duyarga ağlarının programlanması için bugüne kadar geliştirilmiş olan bütün algoritmalar, sürekli çalışan mekanizmalara sahiptir. Yeni bir program yüklenmemiş kararlı durumda bulunan bir ağda bile, düğümler belirli aralıklarla çevrelerine, üzerlerindeki programın versiyonu hakkında mesajlar yaymaktadır. Bu, duyarga ağlarında sınırlı ve değerli olan enerjinin ve bant genişliğinin boşuna harcanması anlamına gelmektedir.

Süreklilik sorununun çözümü için, bütün düğümlerin güncellenmesinden sonra, kod yayımını durduracak bir mekanizma geliştirilmiştir. Algoritmanın temeli, kod yayımından önce, programlanacak düğümler arasında, sezgisel bir ağaç topolojisi oluşturmaya dayanmaktadır. Yavru düğümlerinin hepsinin güncellendiğini gören düğümler, kod yayma işlemine son vermektedir. Böylece ağacın en altındaki düğümler de güncellenince, kod yayımı tamamen durmaktadır.

Geliştirilen kod yayma algoritmasının en önemli özelliklerinden biri, topoloji-duyarlı bir algoritma olmasıdır. Verimi artırmak için, önce en uygun ağ yapısı belirlenmekte ve kod yayma işlemi bu yapı üzerinde çalışmaktadır. Bu yaklaşımın verimi artırdığı ve ağa gönderilebilecek program sayısını iyileştirdiği, matematiksel olarak ispatlanmıştır. En uygun ağ yapısının belirlenmesinde gerekli olan parametreler, bu matematiksel analizden elde edilmiştir.

Ağaç altyapısının, ağ kopmalarına karşı dayanıklı olmak üzere tasarlanmış, yayın tabanlı ve sürekli olan algoritmalara göre, sistemin toplam performansını geliştirdiği gösterilmiştir. Değişik algoritmalarla karşılaştırmalı sonuçlar alınabilmesi için, simülasyonlar yapılmıştır. Bu simülasyon sonuçları, yaklaşımın, var olan sürekli algoritmalara göre, enerji ve güncelleme maliyeti açısından daha iyi bir performans yakaladığını göstermiştir; aynı zamanda, başarı oranları ve ağ kopmaları dayanıklılığı açısından da karşılaştırılabilir sonuçlar elde edilmiştir.

Anahtar Kelimeler: Duyarga ağları, uzaktan programlama, kod yayımı, ağaç benzeri topolojiler.

ABSTRACT

EFFICIENT CODE DISTRIBUTION ON WIRELESS SENSOR NETWORKS

Recent advances in the technology have enabled networks consisting of small devices capable of sensing, processing and communicating; working in parallel and cooperatively to observe the physical world. These networks, named as “sensor networks”, have brought out one of the most challenging field of todays computer science world. It is envisioned that, in the future, wireless sensor networks will be an integral part of our lives, more so than the present-day personal computers.

Physically reaching the sensor devices after deployment is mostly inefficient. Hence, if it is needed to update the application running on the network, this must be done remotely. Today, there are not many algorithms designed for “code propagation”, which stands for the remote programming of sensor networks, and this field is open to research.

In this thesis, an efficient code propagation algorithm for sensor networks is developed. All of the algorithms that have been developed so far for the programing of sensor networks have mechanisms that run continuously. The nodes on the network send messages periodically, about the version of the program they have, even if the network is in the stable state where there is no new program loaded to the network. This means waste of the resources like energy and bandwith, which are already scarce and valuable on sensor networks.

In order to solve the contunity problem, a mechanism which stops the propagation of the code totally, after all the nodes are updated is developed. The approach is based on the construction of a heuristically optimized tree among the nodes to be programmed prior to code distribution. Nodes detecting that all their children are updated, stop sending messages concerning code propagation. Hence, when all the leaf nodes are updated, the code propagation process is stopped totally.

One of the main features of the algorithm developed is that it is a topology-sensitive algorithm. To increase the efficiency, first an optimum network structure is determined and the code propagation runs on this optimum structure. It is proved mathematically that this approach increases the efficiency and optimizes the number of programs that can be deployed on the network. The necessary parameters needed for the optimum network structure were obtained from this mathematical analysis.

It has been shown that underlying tree infrastructure improves overall performance of the system in comparison to continuous and broadcast algorithms designed to be resistant to network losses. Simulations have been made to obtain comparable results with different algorithms. These simulation results revealed that this approach achieves better performance than currently available continuous algorithms in terms of energy and cost update time while retaining comparable success rates and network loss tolerance.

Keywords: Sensor networks, remote programming, code propagation, tree-type topologies.

1. GİRİŞ

Mikro elektro-mekanik sistem (MEMS), telsiz iletişimi, dijital sistemler ve duyargalar alanındaki hızlı gelişmeler sonucunda ortaya çıkan **duyarga ağları**, araştırmacıların önünde yepyeni bir alanın açılmasını sağlamıştır. Algılama, veri işleme, iletişim ve güç ünitelerinden meydana gelen, madeni bir para boyutlarındaki çok sayıda aygıttan oluşan bu ağların çok değişik uygulama alanlarının olması beklenmektedir. Hatta bu ağların, gelecekte, günümüzdeki kişisel bilgisayarlardan daha yaygın bir şekilde günlük hayatımızın ayrılmaz bir parçası haline geleceği tahmin edilmektedir (Akyıldız, 2002).

Duyarga aygıtları, üzerlerindeki değişik duyargalar yardımıyla ortamdan istenilen bilgiyi algılayabilmekte, işlemcileri sayesinde gerekirse bu bilgiyi işleyebilmekte ve alıcı/verici birimi sayesinde de gerekli bilgiyi diğer duyarga aygıtlarına gönderebilmektedir. Bütün bu işlemleri yapmak için gerekli enerjiyi ise, üzerlerindeki, çok kısıtlı enerji seviyelerine sahip, pillerden sağlamaktadırlar.

Duyarga ağları, habitat gözlemleme, orman yangınlarının tespiti, savaşta düşman takibi gibi pek çok değişik konularda uygulama alanı bulmuştur. Duyarga aygıtları, üzerlerindeki duyargalarla algıladıkları bilgilerin, doğrudan veya diğer aygıtlardan gelen bilgilerle sentezlenmesi suretiyle ve etraflarındaki diğer aygıtlar aracılığıyla merkeze gönderilmesini sağlarlar. Bu süreçte, enerjinin büyük bir kısmı haberleşme sırasında harcanır. Bu yüzden duyarga ağının ömrünün uzun olabilmesi için, haberleşme işlemleri olabildiğince azaltılmalıdır. Zira, ağlar ortama yerleştirildikten sonra, tekrar erişilip bataryalarının doldurulması çoğunlukla mümkün değildir.

Duyarga ağları genelde uzun süreler için kurulur ve bu süre içinde kullanıcıların ihtiyaçlarında değişiklikler olabilir. Böyle bir durumda, ağın "*uzaktan programlama"* yoluyla programlanması gerekir. Fiziksel olarak aygıtlara ulaşıp programlamak verimsiz, hatta bazı durumlarda imkânsızdır. Bu yüzden, yeni programın, ağdaki bütün aygıtlara uzaktan gönderilmesini ve yüklenmesini sağlayacak bir mekanizmaya ihtiyaç vardır. Programın tamamı, ağdaki bütün aygıtlara gönderilmeli ve bu işlem, mümkün olduğunca az enerji harcanarak yapılmalıdır. Bu işlem için harcanan enerji, uygulama için harcanan enerjiye oranla çok düşük olmalıdır. Aksi halde, asıl amaçtan uzaklaşılmış ve kullanıcının ihtiyaçları yeterince karşılanamamış olur.

Bu konuda şimdiye kadar yapılmış olan çalışmalarda, kod yayımı için önerilen yöntemler "sürekli" (continuous) yöntemlerdir. Duyargalar sürekli olarak, belli aralıklarla (sabit veya

değişken) etraflarına, üzerlerinde yüklü olan programla ilgili versiyon bilgisini göndermekte, bu sayede güncellenmemiş aygıtlar tespit edilmeye çalışılmaktadır. Ağda "bölünme" (network partitioning) meydana geldiğinde, ağdan kopan aygıtlar tekrar ağa dahil olunca, üzerlerindeki programın güncellenmesi gerekir. Sürekli çalışma yönteminin, bu tarz sorunlara da çözüm getirdiği düşünülmektedir.

Bu tez çalışmasında, kod yayma mekanizmasının, yukarıda belirtilen sorunlarla ilgilenmesinin gerekli olmadığı, ağ üzerinde çalışan uygulamaların problemi fark etmesinin mümkün olduğu gösterilmiştir. Kod yayma işlemi belirli bir süre aldığı için, güncelleme sırasında ağda aynı anda iki (veya daha çok) programın çalışması durumu ortaya çıkabilecektir. Böyle bir durumda, eski programın yüklü olduğu duyargaların göndereceği duyarga bilgileri geçersiz olacaktır. Bunun anlaşılabilmesi için ise, gönderilen duyarga bilgileriyle beraber program versiyon bilgisi de gönderiliyor olmalıdır. Bu mekanizmanın sağlanması durumunda, güncel olmayan duyargaların tespit edilmesi için, ayrıca bir işleme gerek kalmayacaktır. Ağ bölünmesi durumunda ise, kopan duyargalar ağa dahil olduğu zaman, kopuk oldukları esnada ağdaki program değişmiş ise, ağa gönderecekleri ilk pakette durum tespit edilecektir. Eski programa ait verileri alan duyargalar, güncellenmemiş duyargaların yeni programı almalarını sağlayacaklardır. Bu nedenle, kod yayma mekanizmasının sürekli olması gerekli değildir.

Önerilen kod yayma mekanizması, sezgisel olarak (heuristically) iyileştirilmiş bir ağaç yapısına dayanmakta ve iki evreden oluşmaktadır. Kod yayma işlemi başlamadan önce düğümler arasında bir ağaç yapısı oluşturulmaktadır. Hop (atlama) sayısı, batarya doluluk düzeyi, ağ yoğunluğu gibi performans kriterleri dikkate alınarak, programlanacak duyargalar arasında meydana gelecek bu ağaç yapısının uyarlanabilir olması sağlanmaktadır. En uygun ağ yapısının belirlenip kod yayma işleminin bu yapı üzerinden yapılmasının verimi artırdığı ve ağa gönderilebilecek program sayısını iyileştirdiği matematiksel olarak gösterilmiştir. Ağ yapısının en uygun hale getirilmesi için gerekli parametreler, bu matematiksel analizden elde edilmiştir.

Uygun ağ yapısının belirlenmesi ve ağaç yapısının oluşturulmasından sonraki kod yayma aşamasında, ağdaki her duyarga sadece kendi çocuklarının programlanmasında sorumlu olmakta ve onlar güncellendikten sonra versiyon bilgileri gönderme işini durdurmaktadır. Sürekli olmayan bu yöntemin, sürekli yöntemlere göre, özellikle enerji tasarrufu konusunda önemli avantajlar sağladığı, analizler ve testler yardımıyla gösterilmiştir.

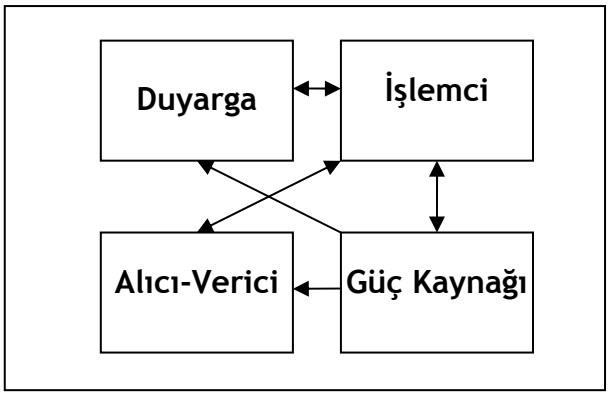
Bu tezin ikinci bölümünde, telsiz duyarga ağları ve kullanım alanları anlatılmaktadır. Üçüncü

ve dördüncü bölümlerde ise, sırasıyla telsiz duyarga ağları için geliştirilmiş olan işletim sistemleri ve uzaktan programlama mekanizmaları incelenmiştir. Beşinci bölüm, bu tez çalışmasında geliştirilmiş olan uzaktan programlama algoritmasını ve bu algoritmanın diğer algoritmalarla simülasyon ortamında karşılaştırılmasından elde edilen sonuçları içermektedir. Altıncı bölümde ise, geliştirilen algoritmanın matematiksel analizi verilmiştir.

2. DUYARGA AĞLARI VE KULLANIM ALANLARI

Telsiz iletişim ve dijital elektronikte, son yıllarda meydana gelen ilerlemeler, oldukça küçük boyutlu ve kısa mesafelerde bağımsız olarak haberleşebilen, düşük maliyetli ve daha düşük enerjili telsiz duyarga aygıtlarının geliştirilmesine imkân vermiştir. Bu küçük aygıtlar, üzerlerine yerleştirilebilecek değişik duyargalar sayesinde sıcaklık, nem, ışık, vs gibi ortam bilgilerini algılayabilmekte; üzerlerinde bulunan işlemci sayesinde bu veriler üzerinde uygulamaya uygun işlemleri yapabilmekte ve yine sahip oldukları telsiz haberleşme bileşeni sayesinde, diğer düğümlerle veri alışverişinde bulunabilmektedir.

Bütün bu elverişli özellikler, bu aygıtların pek çoğunu bir arada kullanarak, bir ağ oluşturma fikrine ilham vermiştir. Böylece, çok değişik ortamlarda kullanım alanı bulacağına inanılan Telsiz Duyarga Ağlar (TDA) ortaya çıkmıştır (Wireless Sensor Networks – WSN).



Şekil 2-1 Bir duyarga aygıtının yapıtaşları

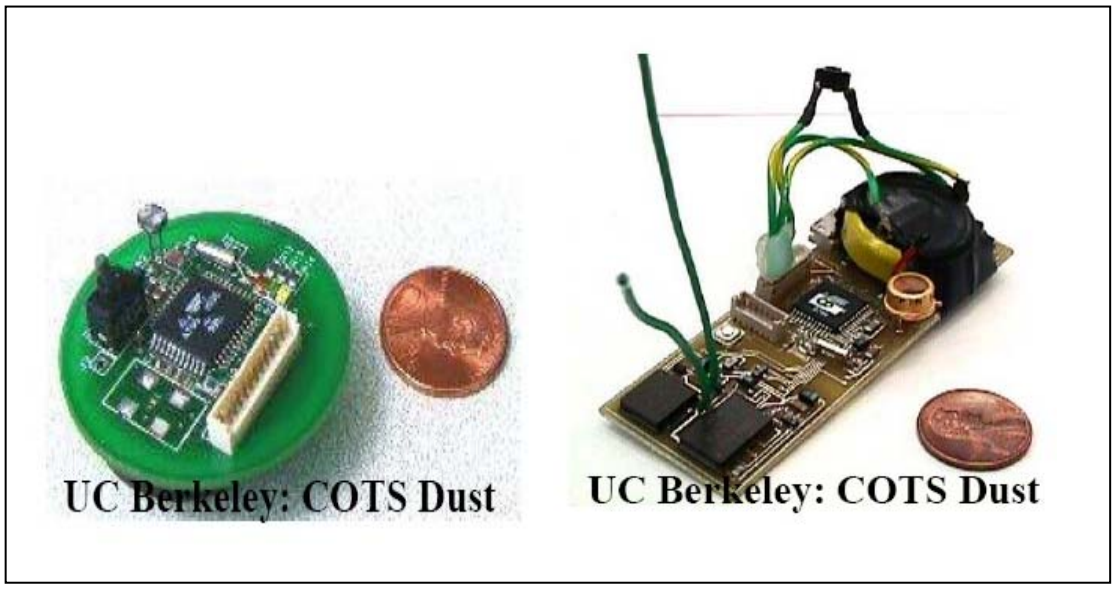
2.1 Duyarga Aygıtlarının Yapısı

Bir duyarga düğümü temel olarak dört birimden oluşmaktadır (Şekil 2-1): İşlemci, duyarga, alıcı-verici devresi ve güç kaynağı. Şekil 2-2'de de bazı duyarga düğümlerinin resmi verilmiştir (Crossbow, web sitesi).

Duyarga aygıtlarının ana birimlerinden olan algılayıcılar, yapılmak istenen işe göre çok çeşitlidir. Bunlardan bazılarını şöyle sıralamak mümkündür:

- Sıcaklık algılayıcıları
- Basınç algılayıcıları
- Nem algılayıcıları
- Sismik algılayıcılar
- Görüntü algılayıcıları

- Hareket algılayıcıları
- Mekanik gerginlik algılayıcıları
- Hız algılayıcıları
- Yön algılayıcıları
- Miktar algılayıcıları
- Aydınlık algılayıcıları
- Canlı-cansız varlık tespiti algılayıcıları
- Gürültü algılayıcıları…



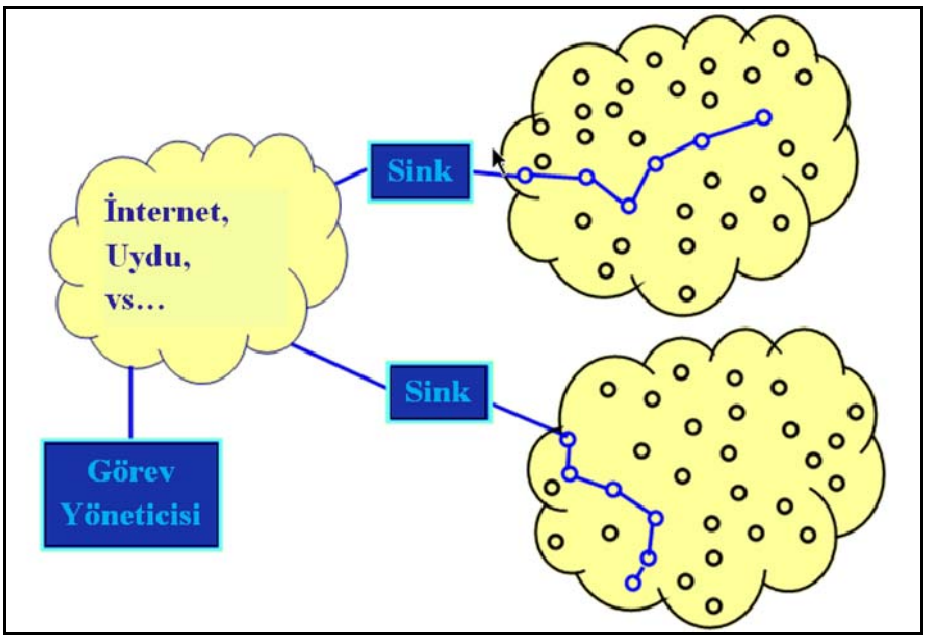
Şekil 2-2 Örnek duyarga aygıtları

Başlangıçta, kibrit kutusu büyüklüğünde imal edilen duyarga aygıtları, tekniğin gelişimi ile giderek küçülmüş ve boyutları daha sonra bozuk para, şimdilerde ise, neredeyse bir fasulye hatta buğday tanesi büyüklüğüne düşmüştür.

Günümüz piyasasında kullanılan duyarga düğümlerinin boyut ve kapasiteleri çok farklıdır. Diğer bir ifadeyle, tek tip algılayıcı düğümden bahsetmek söz konusu değildir. Ancak işlevsel olarak; boyut ve kapasitesi ne olursa olsun, veri algılama, işleme ve kulaktan kulağa iletişim yöntemiyle, algıladığı bilgileri, komşu düğümlerle paylaşma, algılayıcı düğümlerin ortak özelliğidir. Algılayıcıların bu özelliğinden yararlanılarak kurulan atlamalı (multi hop) ağlar, bir taraftan kapsama alanını genişletirken, diğer taraftan da, düğümlerde güç kaybını en aza indirmektedir.

Telsiz duyarga ağları, bu küçük aygıtların onlarca, yüzlerce, hatta binlercesinin bir araya gelerek oluşturduğu, bulunulan ortamla ilgili istenilen bilgileri elbirliği ile elde edip kullanıcıların ulaşabilecekleri bir aygıta ileten ağlardır (Şekil 2-3). Duyarga aygıtlarından

daha gelişmiş, daha fazla enerjiye sahip (tercihen şebekeye bağlı) ve duyarga ağından elde edilen bilgileri merkeze gönderen aygıtlara “sink” adı verilir.



Şekil 2-3 Duyarga ağı genel görünümü (Akyıldız, 2002)

Zaman içinde maliyetlerinin düşmesi, boyutlarının her gün biraz daha küçülmesi, bu ağların kullanım alanlarını da sürekli genişletmiştir. Özellikle dağıtık (distributed) algılama tekniklerinin sağladığı yeni imkânlarla, üretim, güvenlik, savunma ve trafik gibi değişik alanlarda kullanımları yaygınlaşmıştır.

Duyarga ağları, duyargaların belirlenen ortama, (elle koyma, elle serpme, araç üzerinde giderken serpme, uçaktan atarak serpme vs…gibi) en uygun yöntemle yerleştirilmelerinden sonra, aralarında, bilgi alış verişine koyulmaları ile oluşmaya başlarlar. Önceden belirlenen bilgi biriktirici (sink) etrafında, programlanmış "protokol" uyarınca, organize olmaya başlayan düğümler, algıladıkları verileri, birbirleri üzerinden "sink"e ulaştırırlar. Sink de aldığı verileri, (uydu, sabit yahut hareketli aktarıcı üzerinden veya doğrudan) kullanıcıya aktarır. Böylece ulaşılması amaçlanan bilgilere ulaşılmış olur. Şartların uygunluğuna göre, veri iletiminin "internet" veya "intranet" üzerinden sağlanması da mümkündür (Şekil 2-3).

2.2 Duyarga Ağlarının Ayırt Edici Özellikleri

Duyarga ağlarını, geleneksel ağlardan ayıran özelliklerin bir kısmını şöyle sıralamak mümkündür (Çimen, web sitesi):

- Duyarga ağlarında haberleşme elemanı olarak kullanılan duyarga sayısı, geleneksel telsiz ağlarda kullanılan bilgisayar sayısından çok fazladır.
- Duyargaların gerek çok küçültülmüş yapıları, gerekse atıldıkları ortamların müsait olmayan şartları dolayısı ile bazılarının çalışmaması söz konusudur.
- Duyarga ağlarında, güç kaynağı (batarya), bellek, işlemci gibi donanım kapasiteleri sınırlıdır.
- Duyarga ağlarının, adrese dayalı, statik bir topolojileri yoktur.
- Duyargaların her birinin başında, kullanıcıları yoktur. Alana bırakıldıktan sonra, kendi kendilerine organize olurlar.
- Telsiz duyarga ağlarında, düğümler, yerleştiriliş biçimlerine bağlı olarak, alanda veya alanın bazı bölgelerinde çok yoğun olarak bulunabilirler.

2.3 Duyarga Ağlarında Tasarımı Etkileyen Faktörler

Telsiz duyarga ağlarının yukarıda listenen özellikleri dolayısıyla, üzerlerinde çalışacak uygulamalar tasarlanırken dikkate alınacak kriterler, geleneksel ağlara göre çok farklıdır. Bu tasarım sırasında dikkat edilmesi gereken faktörler aşağıdaki gibidir (Çimen, web sitesi):

Ağın Hataya Karşı Dayanıklılığı

Ağın içindeki bazı düğümler, enerjinin bitmesi veya yetersizliği, hasara uğrama, çevreden kaynaklanan olumsuzluklar gibi nedenlerle işlevlerini yitirip devreden çıkabilirler. Bu gibi durumlarda bile, ağın işlevini sürdürebilmesi onun hatalara karşı dayanıklılık kabiliyetidir. Bu kabiliyet, uygulamanın türüne ve uygulama alanına göre belirlenmesi gereken bir ölçüttür.

Düğüm Sayısı ve Yoğunluğu

Gözlemlenecek alana dağıtılacak düğümler, binlerle, yüz binlerle ve hatta milyonlarla ifade edilecek derecede kalabalık, yoğunlukları da çok yüksek olabilir. Tasarımcıların, protokol ve algoritma geliştirirken, bu yüksek rakamları göz önünde bulundurması gerekir.

Maliyet

Her türlü üretimde olduğu gibi, ağ tasarımcıları da, maliyeti dikkate almak zorundadır. Ancak buradaki asıl zorluk bir taraftan işlevsel kalite ve dayanıklılığı yükseltmeye çalışırken diğer yandan maliyeti düşürmeye çalışmaktadır.

Donanım Kısıtları

Bir duyarga aygıtı, algılayıcı, işlemci, alıcı/verici ve güç birimleri gibi dört ana biriminin yanında, gerekli durumlarda, yer bulma, güç üretimi, konum değiştirici birimlerinden oluşabilmektedir. Tüm bu birimlerin, yine, gerektiği hallerde, bir kibrit kutusuna sığacak kadar veya küçük bir bozuk para boyutunda ve hatta bir buğday tanesi büyüklüğünde olması

gerekebilecektir.

Tasarım ve üretimde tüm bunların yanında, düşük enerji sarfiyatı, yüksek yoğunlukta çalışabilme, düşük maliyet, kendi kendine çalışma ve çevreye uyum sağlama gibi zorlayıcı koşulları, diğer bir deyimle "kısıt"ları da göz önüne almak zorunludur.

Topoloji

Bir duyarga ağının yerleştirilme biçimi ve oluşturulan fonksiyon şemasına, o ağın topolojisi denir. Ancak, arıza, enerji tükenmesi vs gibi durumlar sonucu bazı düğümlerin ağdan ayrılması, hareketli düğümlerin yer değiştirmesi gibi nedenlerle, ağın topolojisi sürekli değişir. Duyarga ağlarında topoloji değişiminin üç aşaması aşağıdaki gibidir:

Konuşlandırma öncesi ve konuşlandırma evresi: Şartların gerektirdiği duruma göre, düğümler alana; insan eli veya robotlar mahareti ile, tek tek yerleştirilebileceği gibi, toplu halde, uçaktan, top mermisi veya bir mancınıkla da atılabilirler. Elde olmayan nedenlerle, duyargaların istenildiği gibi yerleştirilememesine rağmen, yerleştirme yapılırken, düşük maliyet, yeniden düzenlemeye yol açmama ve hataya karşı dayanıklılık gibi kurallar göz önüne alınmalıdır.

Konuşlandırma sonrası evresi: Duyarga aygıtları, daha önce belirtildiği gibi, yerleştirildikten sonra, enerjilerinin tükenmesi, tahribata uğramaları, akıntı, rüzgâr gibi etkilerle sürüklenmeleri gibi olumsuzluklara maruz kalma riskini taşımaktadırlar. Dolayısı ile tasarımcıların bu ve benzeri riskleri göz önünde bulundurmaları zorunludur.

Yeniden veya ek düğüm yerleştirme evresi: İzleme boyunca, yukarıda belirtilen sebeplerle (enerji tükenmesi, bozulma, sürüklenip kaybolma vs…) düğümlerin yerine yenilerinin konması da mümkündür. Ancak bunun için özel "protokol ve algoritma"ların gerekliliğine dikkat edilmelidir.

Çevre şartları

Doğal olarak, gözlemlenecek alan veya olayın durumuna göre, algılayıcıların konuşlandırılacağı çevre ve çevre şartları çok değişik olabilir. Örneğin: Algılayıcılar; geniş ve açık bir alanda, ormanlık bir arazide, uzun otlarla kaplı çayırlık bir alanda veya buğdaylarla kaplı bir ovada, çok çamurlu veya kirli bir alanda, bina-ev-depo-gibi kapalı alanlarda, hareket halindeki araçlar veya canlı hayvanlar üzerinde, denizlerin dibinde veya akarsuların içinde, bir savaş esnasında cephe gerisinde vs. çalışıyor olabilirler. Dolayısı ile konuşlandırıldıkları alana

bağlı olarak da faaliyet gösterilecek ortam, çok yüksek veya alçak basınç altında, hızlı bir hava akımının etkisinde, akan bir akarsu veya dalgalı bir denizle boğuşulan bir durumda, çok gürültülü veya sakin bir ortam olabilir.

İletişim ortamı

"Telsiz duyargalarla" kurulan bir ağda, doğal olarak iletişim de telsiz olacaktır. Klasik algılama ağlarındaki kablolu iletişimin yerini, telsiz duyarga ağlarında; **radyo frekansları**, **optik** veya **kızılötesi** ile iletişim almaktadır.

> **Radyo frekansları ile iletişim:** Bu yöntemde, ücretsiz, belli bir standarda bağlı olmaksızın dünya çapında kullanılabilen, geniş spektrumu sayesinde, enerji tasarruflu tasarımları destekleyen Industrial, Scientific, Medical (ISM) bantları kullanılır. Ancak yaygın bir şekilde kullanılmakta olan bu yöntemin, enerji ve dış etkenlerden çok fazla etkilenmesi gibi bazı dezavantajları dikkate alınmak zorundadır.
>
> **Kızılötesi ile iletişim:** Üretimi kolay, lisansa ihtiyaç gerektirmeyen, elektrikli aygıtlardan etkilenmeyen bu yöntem çok avantajlı olmasına rağmen, sağlıklı işlevi için, algılayıcıların birbirini görme zorunluluğu dolayısı ile duyarga ağlarında nadiren tercih edilmektedir.
>
> **Optik aygıtların kullanımı yolu ile iletişim:** Ayna kombinasyonlarının kullanılmasıyla uygulanan pasif mod veya lazer diyot kullanarak uygulanan aktif mod şeklinde gerçekleştirilebilen bu iletişim yönteminin de her ortamda kullanılması zor ve bazen imkânsızdır.

Enerji sarfiyatı

Kolay ve risksiz kullanımın sağlanması için aşırı derecede küçültülen duyargaların haliyle enerji kaynakları da çok küçüktür. Uygulamaların genelinde de enerji kaynağının yenilenmesi mümkün değildir. Veri alma, yönlendirme, gönderme gibi işlevlerini icra ederken enerji harcayan düğümlerin ömrü doğal olarak bu enerjinin dayanma süresi kadardır.

Bir ağın içinde bir çok düğümün enerji bitimi sebebiyle devreden çıkması ağın topolojisini de değiştirir. Bu da paketlerin yeniden yönlendirilmesi ihtiyacını doğurur ki bu durumda, enerji tüketimi daha büyük bir önem arz eder.

Diğer mobil ve tasarsız (ad-hoc) ağlarda da, tasarımda önem arz eden enerji tüketimi, enerji

kaynağı kullanıcı tarafından değiştirilebildiği için duyarga ağlarındaki kadar önemli bir tasarım faktörü değildir. Duyarga ağlarında böyle bir imkânın bulunmaması, araştırmacıları, daha düşük enerji sarfiyatlı yöntemleri geliştirmeye yönlendirmiştir. Günümüzde bu alanda çalışanların mesailerinin büyük bir kısmını bu konuya ayırdıkları söylenebilir.

2.4 Duyarga Ağlarının Kullanım Alanları

Başlangıçta bahsedilen, algılayıcı türlerine paralel olarak, duyarga ağları çok değişik alanlarda kullanılmaktadır. Kullanım alanları, günden güne çeşitlenip çoğalmakta olan duyarga ağlarının, günümüzde kullanıldığı en önemli alanları şöyle sıralamak mümkündür (Akyildiz, 2002):

- Askeri uygulamalar
- Sağlıkla ilgili uygulamalar
- Çevre ile ilgili uygulamalar
- Ticaret ile ilgili uygulamalar
- Ev ve ev eşyaları ile ilgili uygulamalar
- Doğal afetler ve sonrası ile ilgili uygulamalar

2.4.1 Askeri Uygulamalar

Bilinen yaygın kanıya uyarak, duyargalar, askeri alanda hissedilen ihtiyaçlar dolayısı ile yapılan araştırmalar neticesinde ortaya çıkmıştır denilebilir. Dolayısı ile telsiz duyarga ağlarının en geniş ve yaygın bir şekilde kullanıldığı alan askeri alandır. Telsiz duyarga ağları, keşif ve gözetleme, komuta ve kontrol, haberleşme ve istihbarat, hedefleri tespit gibi sistemlere kolayca eklenebilirler. Diğer taraftan, hataya karşı mukavemetleri, konuşlanma kolaylıkları, ucuz maliyetleri, bir kısmı imha edilse bile görevin engellenememesi ve kendi kendine çalışabilme kabiliyetleri, bu ağların askeri amaçla kullanımını yaygınlaştıran başlıca etmenlerdir.

Duyarga ağlarının kullanıldığı askeri alanlara daha yakından bakılacak olursa, bu alanlar ve uygulama detayları şöyle sıralanabilir:

> **Arazi, alan ve düşman kuvvetlerinin keşifleri:** Yollar, geçitler, mola ve konuşlanma alanları, mevziler, düşman kuvvetlerinin durumu, çok pratik bir şekilde yerleştirilecek algılayıcılarla, tatmin edici şekilde gözlenebilir ve takip edilebilir.
>
> **Sahip olunan cephane ve donanım ile dost kuvvetlerin gözlenmesi ve takibi:** Komuta kademesi, birliklerine yerleştirecekleri duyarga ağları ile sahip bulunduğu cephane ve teçhizat durumunu sürekli izleyebilir, gerektiğinde bu bilgileri derli toplu

bir şekilde üst kademelerine rapor edebilir.

Hedef saptama: Olayın derinlemesine detayları, alan uzmanlığını gerektirir. Ancak, duyarga ağlarının, güdüm sistemleri ile birlikte kullanılması suretiyle, hedeflerin, başarılı bir şekilde saptanıp vurulabildiği bilinmektedir.

Hasar tespiti: Yapılacak bir saldırıdan önce ve sonra, saldırı alanına yerleştirilecek duyarga ağları sayesinde, hedefe verilen hasar tespit edilebilir.

Nükleer, biyolojik ve kimyasal (NBC) saldırıların erken tespiti ve hasar keşfi: Dost kuvvetlerin mevzilerine yerleştirilecek duyarga ağları mahareti ile, bu saldırılar önceden tespit edilerek, kayıplar en aza indirilebilir. Saldırı sonrası da, zehirlenme ve radyoaktif etkilerden sakınmak için, ortama duyarga ağları konuşlandırılarak, hasar tespiti yapılabilir.

2.4.2 Sağlıkla İlgili Uygulamalar

Duyarga ağları, sağlık sektöründe hastaların, gerek kendilerinin ve gerekse organlarının gözlenip izlenmesi, hasta davranışlarının belirlenmesi, personelin ve hastaların yer tespiti gibi değişik konularda kullanılmaktadır. Yöntem olarak, küçük boyutta ve hafif algılayıcılar, hasta veya personelin üzerine yerleştirilerek ve rahatsızlık yaratmadan kullanılabilmektedir. Bu yolla örneğin, psikolojik hastaların anlık davranışları, normal hastaların ve personelin nerede olduğu, kolaylıkla tespit edilebilmektedir. Böylece psikolojik hasta ve yaşlıların kayma, düşme gibi risklere karşı korunması, daha serbest ve özgür ortamlarda yaşamalarının temini, hasta ve personele kısa zamanda ulaşma konularında kolaylıklar sağlanmaktadır.

2.4.3 Çevreyle İlgili Uygulamalar

Duyarga ağları, çevre ile ilgili olarak, orman yangınları, hayvanların izlenmesi, çeşitli doğa olaylarının izlenmesi ve önlemlerin alınması, hava, su ve toprak incelemeleri, çevre şartlarının tarım ürünlerinin yetişmesi ve verimliliklerine etkileri gibi konularda kullanılabilirler.

Geniş ormanlara serpilecek, binlerce hatta milyonlarca düğümle oluşturulacak ağlar sayesinde, çıkabilecek orman yangınları kısa sürede tespit edilip söndürülebilecek; kuş, böcek, sürüngen gibi hayvanlar gözlemlenebilecek; akarsu, baraj ve göletlerdeki su seviye ve hareketleri gözlenerek, taşmalarını önlemeye yönelik tedbirler alınabilecek; sulama, ilâçlama ve gübreleme faaliyetleri kontrollü bir şekilde yapılarak, verim artırımı sağlanabilecek; erozyon ve hava kirliliği gibi olumsuz durumlar önceden gözlenip engellenebilecektir.

Sistemin güneş pilleri mahareti ile beslenmesi sağlanabilirse, tüm bu faaliyetlerin, uzun süreli olması mümkün hale gelecektir.

2.4.4 Ev Uygulamaları

Evlerin içine konuşlandırılacak ağlarla, hırsızlık, su, gaz ve elektrik tesisatlarından kaza ile çıkabilecek tahribat başlangıçlarından, anında haber alınabilir. Buzdolabı, çamaşır veya bulaşık makinesi, elektrik süpürgesi vs. gibi ev aletlerine monte edilecek duyarga ağlarıyla da, yerinden veya uzaktan, kontrol ve kumandaları sağlanabilir.

2.4.5 Doğal Afetler ve Sonrası İle İlgili Uygulamalar

Doğal afetler sonrası, özellikle kurtarma çalışmalarında zaman unsuru çok önemlidir. Duyarga ağları sayesinde insanlar için yer tespiti süratle gerçekleştirilerek can kayıpları mümkün olan en aza indirilebilir. Yine bu ağlar yardımı ile hasar tespitini de sağlıklı bir şekilde yapmak mümkündür.

2.4.6 Ticari ve Diğer Uygulamalar

Bir ticarethanede, depodaki malların her birine entegre edilecek bir düğümle kurulacak ağ sayesinde, ileride yeri ve sayısı unutulabilecek malın, yerinin neresi olduğu ve elde kaç tanesinin bulunduğu kolaylıkla anlaşılabilir.

Arabalara ve çevrelerine uygulanacak duyarga ağları sayesinde çalınmaları çok rahatlıkla önlenebilir. Kurulacak duyarga ağları ile, binaların düzenli ısıtılmaları ve sağlıklı bir şekilde havalandırılmaları sağlanabilir.

2.5 Telsiz Duyarga Ağlarının Üstünlükleri

Telsiz duyarga ağlarının, geleneksel ağlara göre belli başlı üstünlüklerini şöyle sıralayabiliriz (Tekin, 2006):

Hareket Serbestîsi: Telsiz duyarga ağlarında, telsiz haberleşen duyarga aygıtlarının yer değiştirmeleri, işlevlerini yerine getirmelerini engellememektedir. Bu durum, önemli bir avantajdır.

Düşük Maliyet: Telsiz iletişim ve sayısal elektronikteki hızlı gelişmeler sayesinde, duyarga ağlarının maliyeti günden güne düşmektedir. Çalışmalar ilerleyip kullanımları yaygınlaştıkça, maliyetlerin daha da düşmesi beklenmektedir.

Taşıma Kolaylığı: Kablo ve bağlantılarının olmaması, bir yerden başka bir yere taşınmalarını kolaylaştırmıştır. Oysa kablolu ağlarda bu çok daha zordur.

Kullanım Kolaylığı: Telsiz duyarga ağlarının, yerleştirilmelerinden sonra, duyargaların kendi aralarında organize bir şekilde faaliyet göstermeleri, kullanım bakımından büyük bir kolaylıktır.

Ölçeklenebilme Kabiliyeti: Herhangi bir duyarga ağına istenildiği zaman yeni düğümlerin ilave edilebilmesi veya komple yeni bir ağın eklenebilmesi de önemli bir avantaj teşkil etmektedir.

Yeniden Kullanılabilme Özelliği: Temel işlevi, algılama ve aktarma olan düğümler, tekrar toplanabilecek bir ortama yerleştirilmiş iseler, tahrip olmadıkları müddetçe başka uygulamalar için de kullanılabilirler. Bu da özellikle maliyetler açısından önemli bir avantajdır.

2.6 Telsiz Duyarga Ağlarının Zayıf Yönleri

Telsiz duyarga ağlarının zayıf yönleri ise aşağıdaki gibidir (Tekin, 2006):

Yüksek Hata İhtimali: Telsiz duyarga ağlarında, iletişim hataları ve elverişsiz çevre şartları, haberleşmeyi olumsuz etkilemekte ve hata paylarını yükseltmektedir.

Kaynakların Kısıtlı Olması: Duyarga aygıtları, enerji miktarı, işlemci gücü ve hafıza büyüklüğü bakımından çok kısıtlı kaynaklara sahiptir. Bu durum, üzerlerinde çalışabilecek uygulamalar açısından, önemli sınırlamalar anlamına gelmektedir.

İzleme ve Yönetim Zorlukları: Telsiz duyarga ağlarının uzaktan kontrol ve denetimi önemli ölçüde kaynak israfına yol açmaktadır. Hem enerji hem de bant genişliği açısından ağa ciddi bir yük anlamına gelen uzaktan yönetim, uygulamaların verimini düşürebilmektedir.

Servis Kalitesi: Telsiz duyarga ağlarında, topolojinin değişken durumu ve hata paylarının yüksekliği, serviste belli bir kaliteyi tutturmayı engellemektedir.

3. DUYARGA AĞLARI İŞLETİM SİSTEMLERİ

Bir işletim sisteminin temel işlevi, sistemdeki kaynakların yönetimini sağlamak ve kullanıcıların bu kaynaklardan kolay ve elverişli bir şekilde yararlanmasını sağlamaktır (Tanenbaum, 1997). Geleneksel işletim sistemleri, bu ihtiyaca cevap vermek için, çok işlemli (multi-process) tasarlanmış ve eş zamanlı çalışan pek çok işlemin haberleşmesi için gerekli mekanizmalarla donatılmıştır.

Gömülü (embedded) sistemlerdeki işletim sistemlerinde, yukarıda bahsedilen ihtiyaçların geçerli olup olmadığı konusunda değişik fikirler öne sürülmekle birlikte, gömülü sistemin veya uygulamanın türüne göre ihtiyaçlar değişiklik gösterebilmektedir. Daha önce de belirtildiği gibi, gömülü sistemlerde, geleneksel sistemlere göre çok daha fazla kaynak sorunu vardır. İşlemci, bellek, enerji, bant genişliği gibi kaynaklar genellikle kıttır. Bu nedenle, geleneksel işletim sistemlerini bu sistemlere doğrudan taşımaya çalışmak elverişli sonuçlar vermeyecek, kaynak israfına neden olacaktır. Bazı durumlarda kaynak kısıtlılıkları nedeniyle taşımak bile mümkün olmayacaktır.

Telsiz duyarga aygıtlarında da kaynakların çok sınırlı olduğu bilinen bir gerçektir. Dolayısıyla, geliştirilecek işletim sistemleri, bu ağların ihtiyaçlarına yanıt verir nitelikte olmalıdır. Örneğin, enerji çok kısıtlı olduğu için, işletim sisteminin enerji yönetimi imkânı sunuyor olması kullanıcıların beklentileri arasındadır. Bu sistemlere işletim sistemi yerine çalışma ortamı da (execution environment) denilebilir, çünkü işletim sistemlerinin sunduğu olanakların pek çoğunu sunmamaktadırlar.

Gömülü sistemlerde çalışacak işletim sistemlerinden beklenecek en önemli özelliklerden biri de, gerçek-zamanlı işletim sistemlerinin sağlamak zorunda olduğu bir özellik olan, zaman şartını sağlıyor olmalarıdır (Lorin, 1981). Üretilen sonucun doğruluğunun yanında, istenilen zaman aralığında üretilmesi de önemlidir. Pek çok işlemin çalıştığı, bir işleme sistemin ne zaman hizmet vereceğinin (işlemciyi ne zaman tahsis edeceğinin) belli olmadığı geleneksel işletim sistemleri bu açıdan da duyarga ağları için uygun olamamaktadır.

3.1 Bir Duyarga Ağları İşletim Sisteminin Sağlaması Gereken Özellikler

Bir duyarga aygıtında, belli bir anda, örneğin, aygıtta bulunan bir duyargadan ortam bilgisi alınırken, üzerindeki alıcı-verici devresine bir mesaj gelebilir. İşlemci bunu dikkate almaz ve duyarga aygıtı ile ilgilenmeye devam ederse, o sırada gelen mesajın kaybolma riski doğacaktır. Yani, başlamış bir işlemin, işlem bitene kadar işlemciyi işgal etmesi, yeni bir

işlemin ancak bundan sonra çalışmaya başlayabilmesi şeklindeki seri işlem mantığı, bu aygıtlar için yeterli olmayacaktır. Aynı anda birden çok işi yapabilme yeteneğine ihtiyaç duyulacağı açıktır.

Geleneksel işletim sistemleri, böyle bir ihtiyacı çok işlemlilik özelliği sunarak karşılamaktadır. İlk bakışta, çok işlemlilik özelliğinin bu sorunu duyarga ağları için de çözebileceği düşünülse de, bunun ciddi performans düşüklüğüne neden olacağı, kaynak kısıtlılığı dikkate alınırsa, rahatlıkla fark edilebilecektir (Li, 2001). İşlemler arasında geçiş yaparken yapılması gereken, eski işlemin yığın bilgisinin saklanması ve yeni işlemin yığın bilgisinin yüklenmesi gibi bağlam değişimi (context switching) operasyonları büyük bir yük getirecektir.

Yukarıda anlatılan sorunlara çözüm olarak, olay-tabanlı (event-based) programlama sistemleri geliştirilmiştir. Bu sistemlerde, bir olay meydana geldiğinde, hızlı bir şekilde bu olayın meydana geldiğini belirten bilginin saklanması ve meydana gelebilecek yeni olaylara karşı sistemin hazır hale gelmesi sağlanmıştır. İki olay arasında geçen sürede ise, daha önce oluşmuş olayların gerektirdiği işler yapılmakta, eğer yapılacak iş yok ise, bir sonraki olaya kadar en az enerji harcanacak durumda bekleme konumuna geçilmektedir.

Geleneksel ağ protokolleri, karmaşıklığı yönetebilmek, modülerlik ile yeniden kullanıma imkân vermek ve büyük protokol yığınını (protocol stack) yönetilebilir kılmak için, katmanlı mimari yöntemini benimsemiştir. Duyarga ağ uygulamaları, katmanlar arası bilgi alışverişine ihtiyaç duyabileceği için, katmanlı ağ protokolü bu ağların ihtiyaçlarına cevap verememektedir (Köpke, 2004). Ağ katmanların sadece komşu katmanlar ile değil, bütün katmanlar ile bağlantılı olduğu bileşenler mimarisi, duyarga ağları için daha uygundur.

Duyarga ağları işletim sistemlerinin sağlaması gereken bir başka özellik ise, enerji yönetimidir. Aygıtların sahip olduğu enerji çok kısıtlı olduğu için, işletim sistemi hem enerjinin elverişli kullanılmasını sağlamalı, hem de uygulamalara, bileşenlerin kapatılması, uyku durumuna geçirilmesi gibi enerji yönetimi için gerekli servisleri sunmalıdır.

3.2 TinyOS

TinyOS, Kaliforniya (Berkeley) Üniversitesinde, Hill vd. (2000) tarafından, duyarga ağları için nesC (Gay, 2003) dilinde geliştirilmiş olan olay-tabanlı (event-based) bir işletim sistemidir. Sistem, duyarga aygıtlarının işlemci ve hafıza boyutu bakımından çok sınırlı kaynaklara sahip olmaları dikkate alınarak tasarlanmıştır. Duyarga ağları uygulamaları, çok

sayıda eş zamanlı işlem gerektiren (high-level concurrency) uygulamalardır. Bu ihtiyacı, geleneksel işletim sistemlerindeki gibi çok izlekli (multithreaded) yöntemle karşılamak, kaynakların kısıtlılığı nedeni ile elverişli değildir. Çok izlekli yöntemde, yığında (stack) her izlek için bir yer ayrılması ve her izlek değişiminde "bağlam değiştirme" (context switch) işleminin yapılması gerekir.

Bu işlemler duyarga ağlarına aşırı bir yük gelmesi anlamına geleceği için, yüksek eş zamanlılık ihtiyacını karşılayacak başka bir yöntem benimsenmiştir. Bu yöntemin adı, olay-tabanlı programlamadır. Mesaj alımı, zamanlayıcı tetiklemesi veya duyarga bilgisi gelmesi gibi bir olay meydana geldiğinde, o olay ile ilgili işlemler hızla ve kesilmeden yerine getirilmekte ve bir sonraki olaya kadar uyuma durumuna geçilmektedir.

3.2.1 TinyOS Yapısı

TinyOS işletim sistemi, bir basit zaman planlayıcı (scheduler) ve bir "bileşenler grafiği"nden (graph of components) meydana gelmektedir. Modülerlik ve olay-tabanlı programlama bu bileşenler kavramı ile sağlanmaktadır. Örnek bir TinyOS bileşeninin yapısı Şekil 3-1'de verilmiştir. Bir bileşen, birbiri ile bağlantılı dört parçadan oluşmaktadır:

- Durum bilgisini içeren "çerçeve" (frame),
- Normal "görevler" (task) için program kodu,
- Komut işleyicileri (command handlers) ve
- Olay işleyicileri (event handlers).

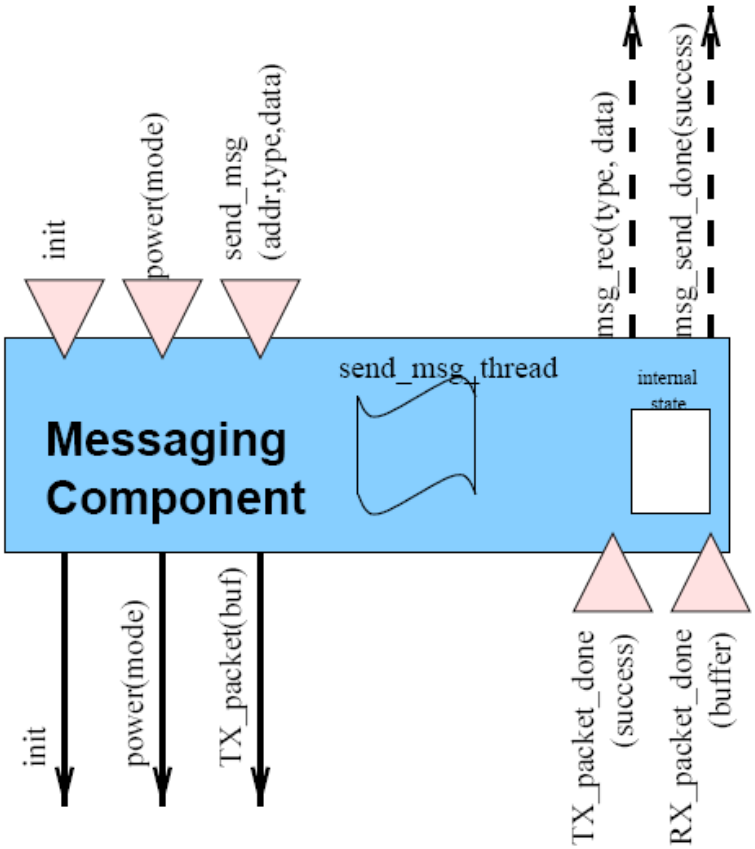
Şekil 3-1'deki kutucuk, bileşenin çerçevesini, aşağı yönlü üçgenler komut kümesini, yukarı yönlü üçgenler işleyicileri, aşağı yönlü oklar kullandığı komutları ve yukarı yönlü kesik çizgili oklar bileşenin yarattığı olayları temsil etmektedir.

Komutlar, görevler ve işleyiciler çerçeve bağlamında çalışır ve yaptıkları işler çerçevenin durumunu etkiler, değiştirir. Ayrıca, her bileşenin bir arayüzü vardır ve bileşenler bu arayüzlerle birbirlerine bağlanır. Arayüzler, bileşenlerin kullandıkları komutlar (command) ve yarattıkları olaylardan meydana gelir.

Çerçevelerin boyutları sabit uzunluktadır ve hafızada yerleri statik olarak ayrılmıştır (static allocation). Bir bileşenin bellek ihtiyacı, derleme sırasında bellidir. Bu sayede, dinamik ayırmanın getireceği ek yük bertaraf edilmiştir. Bunun bir faydası da, bellek bölgelerinin derleme sırasında statik olarak adreslenmesi ve bu sayede, işaretçilere olan ihtiyacın ortadan kaldırılmasıyla programın çalışma hızının artırılmasıdır.

Bileşenler arasında hem komut hem de olay alışverişi olabilmektedir. Bileşenler hiyerarşik bir

biçimde, donanıma daha yakın, düşük seviyeli bileşenlerden, asıl uygulamayı meydana getiren yüksek seviyeli bileşenlere doğru organize edilmiştir (Karl, 2006). Komutlar, yüksek seviyeli bileşenlerden düşük seviyeli bileşenlere gönderilirken olaylar düşük seviyelerden yukarı seviyelere doğru çıkarlar.



Şekil 3-1 Örnek bir TinyOS bileşeninin yapısı (Hill, 2000)

Asıl işi görevler (task) yapar. Görevler, birbirlerine göre atomik, yani bölünemezdirler. Hiçbir görev başka bir görev tarafından bitmeden kesilemez, yarıda bırakılamaz. Bu açıdan bakınca TinyOS iş kesmesiz (non-preemptive) görünse de, görevler, olaylar tarafından kesilebilirler (preempted). Görevler, düşük seviyeli komutları çağırabilir, yüksek seviyeli olaylar yaratabilir ve aynı bileşen içindeki başka görevleri zamanlayabilirler (schedule). Görevlerin kesilemez olması, kısıtlı hafıza açısından bir avantaj sağlamaktadır. Sadece çalışmakta olan göreve ayrılmış bir tek yığın (stack) yeterli olmaktadır.

Zaman planlayıcı (scheduler) basit bir FIFO planlayıcıdan ibarettir. Eğer görev kuyruğu boş ise, işlemci uyuma durumuna sokulabilmektedir, çünkü yeni bir görev artık sadece bir olay tarafından yaratılabilir. Yeni bir olay meydana gelene kadar uyumakta hiçbir sakınca yoktur.

3.3 Contiki

Contiki, (Dunkels, 2004), duyarga ağları dahil olmak üzere, sınırlı hafızalı, mikrodenetleyici tabanlı gömülü sistemler için, İsveç Bilgisayar Mühendisliği Enstitüsü'nde (Swedish Institute of Computer Science) geliştirilmiş olan bir başka hafif (lightweight) işletim sistemidir. İşletim sistemi çekirdeği (kernel) TinyOS'ta olduğu gibi olay-tabanlıdır ama, buna ek olarak Contiki, iş kesmeli çok izleklilik (preemptive multihreading) de sağlamaktadır. Ayrıca gömülü bir TCP/IP yığınını da desteklemektedir.

Tipik bir Contiki yapılanması, 40 kilobaytlık ROM hafızası ve 2 kilobaytlık RAM hafızasından ibarettir. Olay-tabanlı çekirdeğin (kernel) üzerine uygulamalar, gerçek zamanda, dinamik olarak yüklenip kaldırılabilmektedir. Ayrıca, olaylar arası mesaj iletişimi aracılığı ile, izlekler (thread) arası haberleşme sağlanmaktadır.

Contiki'nin sahip olduğu özellikler aşağıdaki gibi özetlenebilir:

- Çok görevli (multitasking) çekirdek
- İş kesmeli çok izleklilik (preemptive multithreading)
- Dinamik program ve servis yükleme özelliği
- TCP/IP yığını
- Grafiksel kullanıcı arayüzü (GUI)

3.4 SOS

Kaliforniya (Berkeley) Üniversitesinde, telsiz duyarga ağları için geliştirilmiş olan bir başka işletim sistemi, SOS'dir (Han, 2005). TinyOS yüklü sistemlerde, aynı anda birden çok uygulama çalıştırılamamakta ve uygulamaları artımlı (incremental) olarak güncelleme yapılamamaktadır. SOS, bu eksiklikleri gidermek amacıyla tasarlanmıştır.

SOS, dinamik hafıza yönetimi, haberleşme ve bileşen (module) yükleme işlevine sahip ortak bir çekirdek (kernel) ve dinamik olarak yüklenip kaldırılabilen bileşenlerden meydana gelmektedir. Bileşenler işlem (process) değildir ve hafıza korumalı tasarlanmamıştır. İşbirliği içinde zamanlanabilen (scheduled) bileşenler, sistemi çok az keserek (interrupt), dinamik olarak yüklenip kaldırılabilirler.

SOS'nin özelliklerini şöyle sıralayabiliriz:

- Dinamik bağlı (linked) bileşenler
- Öncelikli zaman planlayıcı (priority scheduling)
- Basit dinamik hafıza
- Bileşenler arası haberleşme (mesajlaşma, fonksiyon çağrımı, kernel çağrıları)
- Duyarga API (duyarga sürücüleri ve bileşenler arası etkileşimin yönetimi için)

- Taşınabilirlik (portability)

3.5 Diğerleri

Duyarga ağlarında kullanılabilecek diğer işletim sistemlerinden biri, eCos'tur (eCos). Gömülü sistemler için geliştirilmiş, açık kaynak kodlu bir işletim sistemi olan eCos, çok esnek ve yüksek yapılandırılabilir özelliklere sahiptir. Bu işletim sistemi, çok özel uygulamalar için özelleştirilebilmektedir (customise). Duyarga ağları da bu uygulamalardan biridir.

MANTIS (Abrach, 2003), gömülü sistemlerde çalışmak üzere POSIX izlek API'sinin (POSIX threads) bir alt kümesini sunan bir işletim sistemidir. Geleneksel çok izlekli (multithreading) yapıya sahiptir. Mantis, sistemin tamamının veya bir parçasının yeniden programlanabilmesine imkân vermektedir.

Sensorware (Boulis, 2003) ve Agilla (Fok, 2004), hareketli ajan (mobile agent) soyutlamalarının duyarga ağlarında mümkün kılınabilmesi amacıyla tasarlanmış diğer sistemlerdir.

4. DUYARGA AĞLARININ PROGRAMLANMASI

Duyarga ağları, sınırlı kaynaklara sahip çok sayıda, küçük düğümlerden meydana gelir ve yerleştirildikten sonra aylar hatta yılları bulan süreler boyunca çalışmaları gerekir. İhtiyaçlar ve ağın üzerinde bulunduğu ortam değiştikçe, ağın işleyişini değiştirmek için ağa yeni bir programın yüklenmesi gerekebilmektedir. Bu ihtiyacın giderilebilmesi için, ağ üzerinde kod yayabilecek bir mekanizmaya ihtiyaç vardır.

Kod yayma mekanizmasının sağlaması gereken özellikler aşağıdaki gibidir:

- **Uzaktan olmalıdır**: Ağ elemanları yerleştirildikten sonra, bunları geri toplamak çoğunlukla imkânsızdır. Bu yüzden, yeni programın fiziksel erişime gerek duyulmadan, uzaktan yüklenebilmesi gerekmektedir.
- **Güvenilir (reliable) olmalıdır**: Bütün düğümler, yeni programı hatasız olarak alabilmelidir.
- **Düşük bant genişliği**: Bant genişliği duyarga ağlarında değerli bir kaynak olduğu için, programlama işlemi mümkün olan en az sayıda iletim ile sağlanmalıdır.
- **Hızlı olmalıdır**: Yeni program, mümkün olan en kısa sürede bütün düğümlere yayılmalıdır.
- **Düşük enerji**: Düğümlerin enerji kaynakları (piller) çok sınırlı olduğu için, kod yayma mekanizması mümkün olduğunca az enerji harcamalıdır.
- **Düşük hafıza**: Düğümler üzerindeki hafıza alanı çok sınırlıdır. Bu nedenle, her düğüm üzerinde kalıcı olacağı düşünülen kod yayma servisinin kapsayacağı hafıza alanının boyutu da mümkün olduğunca düşük olmalıdır.

Kod yayma işlemindeki sınırlamaları ise aşağıdaki gibi sıralayabiliriz:

- **Hafıza**: Düğümler üzerinde çok sınırlı bir hafıza bulunmaktadır. Kod yayma programı ile uygulama programı bu sınırlı hafızaya sığmak durumundadır,
- **Enerji**: Düğümler enerjilerini üzerlerindeki sınırlı enerjiye sahip pillerden sağlamaktadır. Kod yayımının neden olduğu, çok sayıda mesaj iletimi, en çok pil harcayan işlemdir.
- **Paket boyutu**: Haberleşmede kullanılan paketler çok küçüktür. Bu yüzden gönderilen program "parça"lara (segment) ayrılarak gönderilmek zorundadır.

Duyarga ağlarında kod yayımı konusundaki ilk çalışmaları Berkeley Üniversitesi'nde TinyOS (Hill, 2000) işletim sistemini geliştiren ekip yapmıştır.

4.1 Maté

Bu ekip, kod yayma konusunda ilk olarak "Maté" (Levis, 2002) ismindeki mekanizmayı geliştirmiştir. Maté, duyarga ağları için TinyOS üzerinde çalışacak bir sanal makine olarak tasarlanmıştır. Duyargalar üzerinde çalışan bu "bytecode yorumlayıcı" sayesinde, duyarga ağları uygulamalarının çok büyük bir kısmı, az sayıda yüksek seviyeli komutlardan oluşturulabilmektedir. Çok karmaşık programlar bile, 100 byte uzunluğunu geçmemektedir.

Program kodu, 1 byte'lık 24 komuttan oluşan kapsüllere bölünmekte ve büyük programlar çok sayıda kapsülden meydana gelebilmektedir. Her kapsül, bir TinyOS paketiyle gönderilebilecek boyuttadır. Kapsüllerde bytecode'ların yanında kimlik ve versiyon numaraları da bulunmaktadır.

Duyargalar, kendilerinde yüklü olandan daha güncel bir kapsül duyarsa bunu yüklemekte ve daha sonra bunu etraflarına yaymaktadır. Duyulan eski kapsüller ihmal edilmektedir. Böylece yeni kod, bir virüs gibi ağa yayılmaktadır. Yapılması gereken tek şey, yeni kodun bir duyarga üzerine yüklenip çalıştırılmasıdır.

Çalışmayı yapanların gerçek duyarga ağları üzerinde yaptıkları deneyler sonucunda, yukarıda anlatılan kod yayma yönteminin elverişli olmadığı ve ağı kısa sürede aşırı doyurduğu (saturate) ortaya çıkmıştır. Bunu çözmek için aynı ekip "Trickle" isimli bir algoritma geliştirmiş ve yayımladıkları çalışmada, bu algoritmanın üç değişik türünü önermiştir.

4.2 Trickle

Bu algoritmalar, Maté sanal makinesi üzerinde çalışan dağıtılmış algoritmalardır. Duyargalar, kapsüllerin tamamını göndermek yerine, sahip oldukları kapsüllerin versiyon numaralarını içeren "versiyon vektörleri"ni göndermektedir. Bir vektörü alan düğüm, bunu kendi vektörüyle karşılaştırıp, yayıp yaymayacağına karar vermektedir. Her algoritma için, bütün duyargalar üzerinde aynı frekansta çalışan bir zamanlayıcı bulunduğu kabul edilmiştir.

4.2.1 Discrete Probabilistic Algorithm (DPA):

Her duyarga üzerinde, ilk değerleri sıfır olan p ve c isimli sayıcılar ve τ değeri bulunmaktadır. Duyargalar, kendi vektörleriyle aynı olan bir vektörü alınca p'yi bir artırmaktadır. Her zaman aralığının sonunda c bir artırılmaktadır ve $c = \tau$ olduğu zaman, her duyarga

$$P(V) = 1 / (p+1) \tag{4.1}$$

olasılığıyla kendi vektörünü gönderip p ve c'yi sıfırlamaktadır.

4.2.2 Continuous Probabilistic Algorithm (CPA):

Bu algoritmada, DPA'dakinden farklı olarak c sayıcısı yoktur. Her zaman aralığı sonunda duyargalar

$$P(V) = 1 / [\ \tau * (p+1)\ ] \tag{4.2}$$

olasılığıyla kendi vektörünü gönderip p'yi (τ-1) / τ ile çarpmaktadır.

4.2.3 T-Counter Algorithm (τ-CA):

Bu algoritmanın DPA'dan farkı, her duyarga üzerinde [0,τ] aralığında rasgele bir t değeri bulundurmasıdır. c = t olduğu zaman, duyargalar

$$p < k \text{ ise } Pr(V) = 1 \quad (4.3)$$

olasılığıyla kendi vektörlerini göndermektedir (k, sabit bir değerdir). Ayrıca c = τ olduğu zaman [0,τ] aralığında yeni bir t değeri belirlenmekte ve p ve c sıfırlanmaktadır.

4.3 Deluge

Deluge (Hui 2004), büyük programları da yayabilen bir kod yayma mekanizmasıdır. Program Şekil 4-1'deki gibi sabit uzunluklu sayfalara bölünmekte ve sayfa sayfa gönderilmektedir. Bu bölme işleminin sağladığı üç yarar vardır: (i) alıcı düğümün uzun süre bir durumda kalmasını engellemek, (ii) eski programın sadece bazı sayfalarını değiştirerek yeni programa geçebilmek, (iii) programın tamamını almamış olsa bile, düğümlere ellerindeki sayfaları gönderme imkanı vermek (spatial multiplexing).

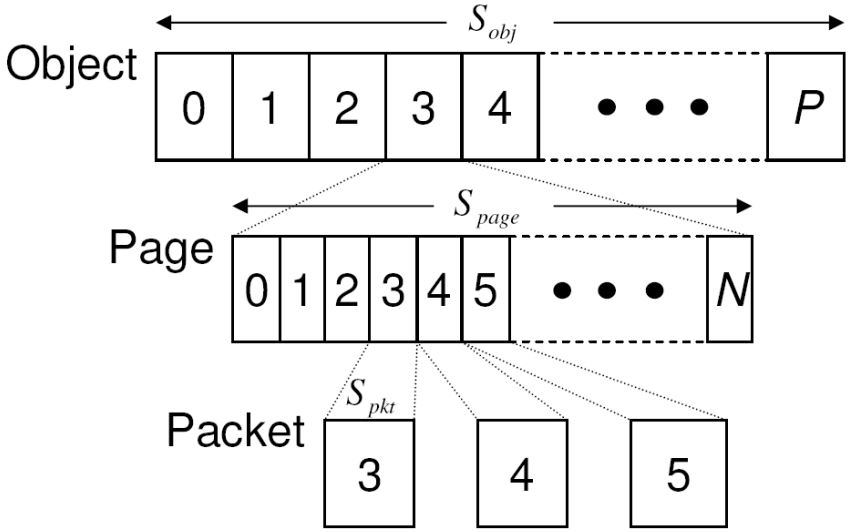
Deluge'te, hem paket hem de sayfa düzeyinde 16-bit CRC (Cyclic Redundancy Check) kontrol bilgisi kullanılmaktadır. Bu sayede, alınan sayfanın kesin doğru ve eksiksiz olup olmadığı anlaşılabilmekte ve yanlış bir sayfayı alan bir düğümün, bunu bütün ağa yayması önlenmektedir.

Bazı durumlarda, yeni program eski programdan sadece birkaç sayfa düzeyinde farklı olabilmektedir. Böyle durumlarda, bütün programı baştan yaymak yerine sadece gerekli sayfaların yayılması yeterli olacaktır. Bu amaçla, her program güncellemesinde düzenli olarak artan bir versiyon numarası kullanılmaktadır.

Hangi sayfaların güncellenmiş olduğunun anlaşılması için Deluge'te aşağıdaki gibi bir yaş vektörü kullanılmaktadır:

$$a = < a_0, a_1, a_2, ...a_{p-1}, > \quad (4.4)$$

Bu vektöre göre, v versiyonundaki i sayfası en son "v-a_i" versiyonunda değişmiştir.



Şekil 4-1 Deluge segment yapısı (Hui, 2004)

Deluge'teki her düğüm, belli bir anda üç durumdan birinde bulunmaktadır: sürdürme (maintain), alma (rx) ve gönderme (tx). Sürdürme durumundaki bir düğüm, etrafındaki düğümlerin son versiyonu bilmelerinden ve bu versiyon için gerekli sayfaları almış olmalarından sorumludur. Bir düğümünün i numaralı sayfayı gönderebilmesi için 1'den i-1'e kadar olan bütün sayfaları almış olması gerekmektedir. Bu durumda bulunan düğümler, belli zaman aralıkları ile program özetlerini yayımlarlar. Gereksiz göndermeleri engellemek için Trickle'daki [2] "bir şeyi k kere duyduysan konuşma" şeklinde tanımlanabilecek olan "nazik dedikodu" (polite gossip) prensibinden yararlanılmıştır.

4.4 MNP (Multihop Network Programming)

MNP, duyarga ağları için geliştirilmiş diğer bir ağ programlama servisidir. Mesaj çakışmalarını (collision) ve gizli terminal problemini (hidden terminal) çözmek için, MNP bir gönderici-seçici algoritması kullanmaktadır. Bu algoritmadaki amaç, bir hücre içinde aynı anda sadece bir düğümün program yaymasını sağlamaktır. Ayrıca, bu seçilen göndericinin, en çok faydayı sağlayacak düğüm olmasına da çalışılmaktadır. Enerjinin az harcanması için ise, gönderilen programa ilgi duymayan düğümlerin belli bir süre uyutulması yolu seçilmiştir.

Programın tamamını alan her düğüm, bir kaynak düğümü haline gelir ve bunu etrafına bildirir. Daha sonra, çevresindeki düğümlerden gelecek istek mesajlarını beklemeye başlar. Bir sayıcı, gelen toplam istekleri tutar. Yeni koda ihtiyaç duyan düğümler, kaynaklardan gelen her mesaja bir istek mesajıyla cevap verir. Belli bir süre sonra her kaynak, kendilerine gelen

toplam istek mesajlarını etraflarına bildirir. Kendisinden daha çok istek almış bir kaynağın varlığını fark eden düğümler, gönderici olmaktan vazgeçip uyumaya geçer. Böylece en çok istek alan düğüm, bulunduğu çevredeki gönderici olur. Uyanan kaynaklar, sayıcılarını sıfırlayıp tekrar etraflarına kaynak olduklarını bildirir. Program gönderme işini bitiren kaynak ise belli bir süre uyumaya geçerek, diğer düğümlere de gönderici olma imkânı verir. Bu işlem, program gönderme yükünün düğümlere dengeli bir şekilde dağıtılmasını sağlar (load balancing).

Alma tarafında ise, program boyutu çok büyük olduğu için, program parçalara bölünmektedir. Her parçaya bir numara verilir ve bu numara hem "advertise" hem de "download" mesajlarına eklenir. Bir düğüm, beklediği parçadan daha büyük numaralı bir parçayı alırsa, atlanmış olan parçalar için istekte bulunur.

4.5 MOAP (Multihop Over-the-Air Programming)

MOAP özellikle "Mica2" duyarga aygıtları için geliştirilmiş bir kod yayma mekanizmasıdır. Tek hop'lu kod yayma mekanizmasının rekürsif olarak çok hoplu bir ağa genişletilmesi prensibine dayanmaktadır. Flooding ile karşılaştırıldığında, gönderilen mesaj sayısında %60-90 arasında bir iyileştirme (azalma) sağlanmıştır. MOAP'ta program gönderiminde "windowed transmission" yöntemi kullanılmaktadır.

Her bölgede, düğümlerden oluşan bir küme, kaynak olarak, diğerleri de alıcı olarak seçilmektedir. Yeni programı alan her alıcı, önceki kaynakların erişiminde olmayan çevresi için kaynak durumuna gelir ve kendi programını etrafına bildirir. İlgi duyan düğümler program için bu kaynaktan istekte bulunur. Eğer hiçbir istek gelmezse, kaynak düğüm, etrafta başka kaynaklar olduğuna hükmeder ve susma durumuna geçer. Böylece kaynaklar ile alıcılar arasında hep bir hop uzaklık bulunur ve hata oranı en aza indirilmiş olur. Bu mekanizmaya "Ripple" denmektedir.

4.6 Diğer Çalışmalar

Jeong (2004), Rsync protokolüne benzer bir protokol yardımıyla, iki programı blok bazında karşılaştırarak parçalı (partial) güncelleme yöntemi kullanmıştır. Bu yöntemde, program yapısının önceden bilinmesi gerekir. Reijers (2003) ise, kodlar arasındaki farkı, ağda yayılmış bulunan betiklerle (script) küçültmeye çalışmıştır.

Impala, (Liu 2003) uygulama modülaritesi, adapte edilebilirlik ve onarılabilirlik sağlamaya

çalışan, orta-seviyeli bir mimariden oluşmaktadır. Olasılıksal yayın protokolü üzerine kurulu bir ağ programlama aracılığıyla yeni kodu ağa yaymaya çalışır. Impala ayrıca verimli enerji kullanımı ve güvenilirlik sağlamak için, bir uzaktan uygulama adaptasyon arayüzü sunmaktadır.

Dutta (2006) ve arkadaşları tarafından, 2006 yılında, Deluge programlama sistemi için bir güvenlik mekanizması geliştirilmiştir. Geliştirilen sistemin, duyarga ağlarının sınırlı kaynakları ile uyumlu olduğu öne sürülmektedir. Sistem, program kodunun parçalarından oluşan her mesajın, kendisinden sonra gelecek mesaj için bir "hash kodu"na sahip olması temeline dayanmaktadır.

Deng (2006) ve arkadaşlarının geliştirdikleri güvenlik yönteminde ise hem "hash kodları" hem de bir "hash ağacı" beraber kullanılmıştır. Mesajların sırayla gelmemesi durumunda bu hash ağacından yararlanılmaktadır.

Deluge üzerine güvenlik çalışması yapan bir başka grup da, Kuzey Carolina Üniversitesi'nden An (2008) ve arkadaşlarıdır.

Güvenli programlama konusunda bir başka çalışma da 2005 yılında Lanigan (2005) ve arkadaşları tarafından yapılmıştır. Kötü niyetli kişiler tarafından ağın programlanmasını engellemek amacıyla, kaynağa duyarlı programlama yöntemi önerilmiştir.

5. KOD YAYMA MEKANİZMASININ TASARIMI

Kod yayma mekanizmalarının en önemli dezavantajlarından biri "sürekli" (continuous) olmalarıdır. Sürekli olmalarından kastedilen, yeni programın gönderilmesi işlemi bittikten sonra da, daha yeni bir program gelmese bile etkinliklerini devam ettirmeleridir. Düğümler periyodik olarak ağda kendilerininkinden daha güncel bir programın bulunup bulunmadığını sorgulamakta ve bu işlem sırasında da mesaj alışverişi olduğu için enerji harcanmaktadır.

Bu sorunu aşmak için, bu çalışmada iki aşamalı bir kod yayma mekanizması sunulmuştur. Birinci aşamada, ağdaki düğümler, aralarında bir "sorumluluk ağacı" meydana getirmektedir. Ağaçtaki her düğüm, kendi yavrularının en güncel programa sahip olduklarını sağlama sorumluluğuna sahiptir. Böylece, güncel olmayan bir düğüm var ise, bunu o düğümün ata düğümü fark etmekte ve yeni programı almasını sağlamaktadır. Ayrıca, ağ uzaktan programlanabildiği için, belli bir anda ağın bir kısmında eski, bir kısmında yeni program bulunması durumları ortaya çıkabilecektir. Böyle durumlarda, uygulama verilerinin karışmaması için, düğümlerin gönderdiği verilere, program versiyon numarası eklenmesi gerekecektir. Bu numara sayesinde, güncel programa sahip olmayan bir düğüm, bunun farkına varabilecektir.

Düğümler arasında oluşturulacak sorumluluk ağacının, kod yayma mekanizmasının performansını ve ağın ömrünü iyileştirecek şekilde oluşturulması gerekir. Örneğin, bir hoptaki bütün düğümler, ata düğüm olarak tek bir düğüme bağlanırsa, bu ata düğümün enerjisi kısa zamanda tükenecek ve ağın ömrü kısalmış olacaktır. Sorumluluk ağacındaki sorumlulukların, düğümlerin enerjileri de dikkate alınarak, dengeli bir şekilde dağıtılması gerekir.

Ağaç oluşturma işlemi tamamen dağıtık (distributed), dolayısıyla ölçeklenebilirdir. Ata düğümünü belirlemiş her düğüm, bir alt seviye için potansiyel bir ata durumuna gelir ve bu durumunu belirten bir mesaj yayımlar. Ata düğümünü belirlemeye çalışan düğümler, belirli bir süre potansiyel düğümleri bekledikten sonra, enerji seviyesi sırasına göre, bunlardan birini ata düğümü olarak seçer.

Yeni program bu yapı üzerinde gönderilmekte ve bütün düğümler güncellendikten sonra, yeni bir program gelene kadar, kod yayma işlemi durmaktadır. Önerilen yöntemin daha önceki çalışmalardan en önemli farkı, sürekli olmayan bu mekanizmadır.

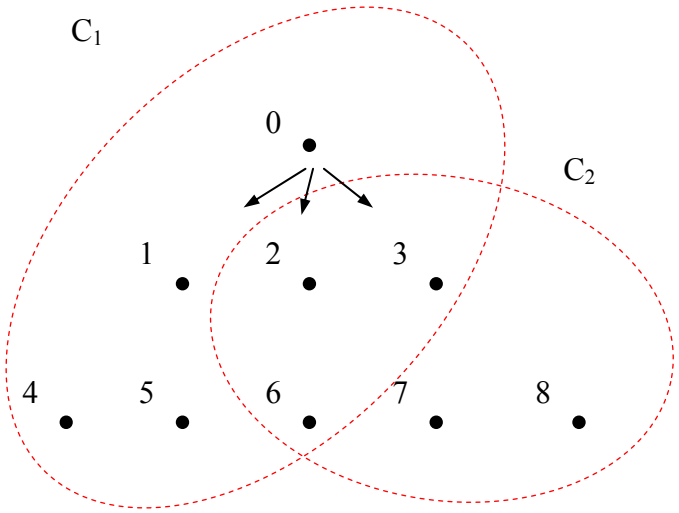
Ağa yeni bir düğüm eklendiği zaman, yerel olarak bir ağaç oluşturma işlemini tetiklemekte ve böylece yeni düğümün ağa katılması sağlanmaktadır. Bu düğümün ata düğümü, yeni programın ağa yeni katılan bu düğüme gönderilmesinden sorumlu hale gelir.

5.1 Ağaç Oluşturma Evresi

Ağaç oluşturma aşaması Şekil 5-1'de verilen örnek bir topoloji üzerinden açıklanacaktır. Ağda iki hücre (C_1, C_2) içinde sekiz düğüm bulunmaktadır.

Varsayımlar:

- Kesik çizgilerden oluşan bölgeler, ağdaki hücreleri göstermektedir. Bir hücre içindeki bütün düğümler birbirlerini duyabilmekte, ancak hücre dışındaki düğümleri duyamamaktadır.
- *i* numaralı düğümün enersi seviyesi E_i'dir.
- $E_8 < E_7 < E_6 < \ldots E_1 < E_0$
- Bir düğümün sahip olabileceği alt düğümlerin sayısının sınırlaması olan "*n*" burada açıklamanın basitleştirilmesi amacıyla, "3" olarak kabul edilmiştir.



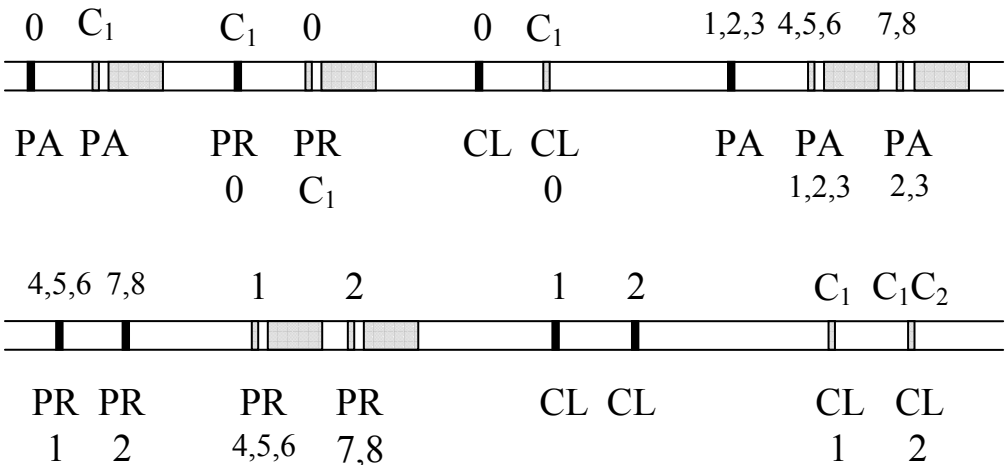
Şekil 5-1 Ağaç oluşmadan önceki topoloji

Ağaç oluşturma işlemi, sink düğümü (düğüm 0) tarafından başlatılmaktadır. Bu düğüm, bir "ata ilân mesajı" ("parent advirtesement" - PA) yayarak işlemi başlatır. Daha sonra meydana gelen adımlar aşağıda maddeler halinde verilmiştir. Aynı adımlar, Şekil 5-2'de de gösterilmiştir.

- C_1 hücresindeki düğümler düğüm 0'ın PA mesajını aldıktan sonra bir zamanlayıcı başlatıp, bu zaman bitene kadar diğer PA mesajları için beklerler. İlk başta, sadece sink düğümü PA mesajı göndereceği için, başka bir PA mesajı alınmayacaktır.

- Daha sonra bu düğümler düğüm 0'dan ataları olması için istekte bulunur. Düğüm 0'ın istek mesajlarını önce 1, 2 ve 3 numaralı düğümlerden aldığı varsayılırsa, bu üçünü alt düğümleri olarak kaydeder ve gelecek başka istekleri reddeder (en çok 3 alt düğüm

olabileceği varsayılmıştı).



Koyu şeritler gönderilen mesajları, gri şeritler alınan mesajları ve büyük olan gri dikdörtgenler ise bekleme (delay) sürelerini göstermektedir. Kısaltmalar: PA (Ata ilân mesajı), PR (Ata istek mesajı) ve CL (Yavru listesi mesajı).

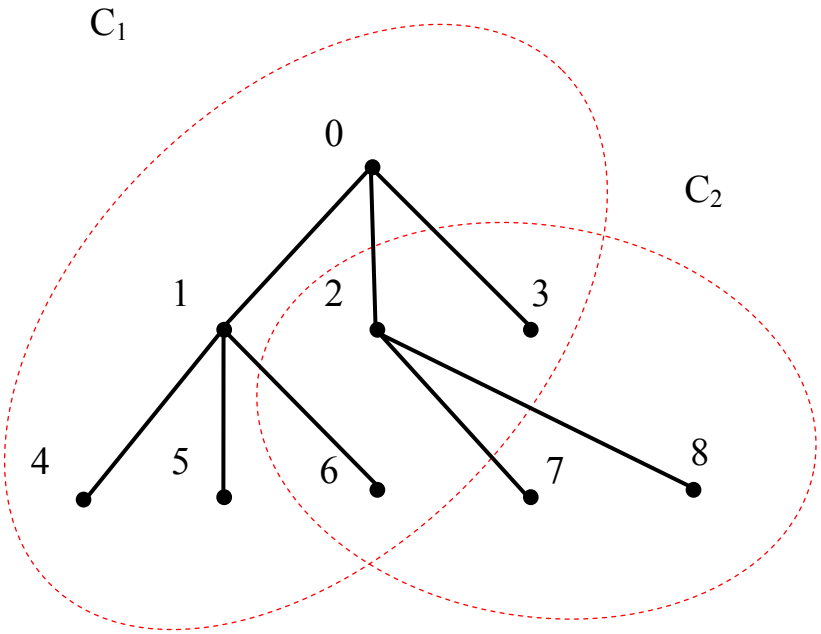
Hattın üzerindeki sayılar, mesajları gönderen veya alan düğümleri ya da hücreleri göstermektedir. Hattın altındaki mesaj türlerinin altındaki sayılar ise, mesajın kaynağını ya da hedefini belirtmektedir.

Şekil 5-2 Ağaç oluşturma mesajları

- Düğüm 0, daha sonra CL'sini (yavru listesi) yayımlar. Bunu duyan 1, 2 ve 3 numaralı düğümler atalarını 0 olarak kaydeder. Aynı mesaj 4, 5 ve 6 numaralı düğümler için de isteklerine ret cevabı işlevi görür. Yani, düğüm 0'ın, yavru olarak kabul etmediği düğümlere ayrı olarak bir "ret" mesajı göndermesine gerek yoktur. Burada telsiz ağların bir özelliğinden yararlanılmış olunmaktadır. Gönderilmiş bir yavru listesi, aynı hücrede bulunan bütün düğümler tarafından duyulacağı için, bu mesaj bazıları için isteklerinin kabulü, bazıları için de isteklerinin reddi anlamına gelmektedir. Telsiz ortamın bu özelliğinden (kapalı-zımnî -implicit- mesaj) algoritmanın değişik aşamalarında faydalanılmaktadır.

- Daha sonra 1, 2 ve 3 numaralı düğümler PA mesajları gönderir. C_1 hücresindeki düğümler bütün PA'ları duyarken 7 ve 8, 1 ile aynı hücrede olmadıkları için, sadece 2 ve 3'ün gönderdiği PA'ları duyar. Düğümler bir PA mesajı aldıkları zaman, bir zamanlayıcı kurarak, belli bir süre beklemede kalır ve diğer düğümlerden gelebilecek PA mesajlarını beklerler. Burada amaç, düğümlerin etraflarındaki en yüksek enerji seviyesine sahip düğümlere başvurmalarını sağlamaktır. Ata ilân mesajlarının yanında, düğümler, enerji seviyelerini de gönderir. Düğümler, PA mesajı gönderen düğümleri,

enerji seviyelerine göre sıralayıp, bir listeye kaydederek, ata istek mesajlarını bu sıraya göre gönderirler. Önce, sıradaki en yüksek enerjili düğümden istekte bulunur; ret cevabı gelirse, bir sonraki en yüksek enerjili düğüme başvururlar.



Şekil 5-3 Oluşmuş olan sorumluluk ağacı

- 4, 5 ve 6 numaralı düğümlerin üçü de "ata istek" ("parent request" – PR) mesajlarını düğüm 1'e gönderir, çünkü enerjisi en çok olan düğüm bu düğümdür. 7 ve 8 ise PR mesajlarını düğüm 2'ye gönderir. 1 numaralı düğüm, sınır 3 olduğu için, gelen 3 isteğin hepsini kabul eder. Bu düğümleri yavru listesine kaydederek yavru listesini yayımlar. Aynı şekilde, düğüm 2 de, gelen iki isteği kabul ederek yavru listesini yayımlar. Böylece, alttaki düğümler ata düğümlerini belirlemiş ve kaydetmiş olur. Sonuçta meydana gelen ağaç yapısı Şekil 5-3'te gösterilmiştir.

Ağaç oluşturma algoritması da, Şekil 5-4'te ve 5-5'te verilmiştir.

```
For all s
   while ( !receive (set-up) )
      s.state = idle
   end while
   s.state = active
   parentPower = 0
   while ( !timeout( parent selection ) )
      switch ( receive )
         case receive( parentAdv, id, power )
            addToList(id, power)
                     end switch
   end while
   while ( parent == NO_PARENT )
      m = findNodeWithMaxPower()
      send( parentRequest, m )
      switch ( receive )
         case receive( accept, id, power )
            s.parent = id
            parentPower = power
         case receive( deny, id )
            removeFromList( id )
            m = findNodeWithMaxPower()
            send( parentRequest, m )
      end switch
   end while
```

Şekil 5-4 Ağaç oluşturma algoritması (I. Evre)

```
   while ( !timeout( children selection ) )
      Broadcast ( parentAdv, s, myPower )
      switch ( receive )
         case receive( parentRequest, id, s )
            if (childNumber < optimumChidren( ) )
               Broadcast ( accept, s, id )
               childNumber++
            else
               Broadcast ( deny, s, id )
         case receive( parentAdv, id, power )
            execute new parent algorithm
      end switch
   end while
```

Şekil 5-5 Ağaç oluşturma algoritması (II. Evre)

5.2 Ağaç Bakımı

Ağaç oluşturma aşaması sırasında ortaya çıkabilecek özel durumlardan biri, "çoklu ata" durumu olarak isimlendirilebilir. Bu durum, düğümlerden birinin, kendisinin de içinde yer aldığı bir yavru listesi mesajını alamadığı zaman ortaya çıkar. Bu düğüm, bir düğüme atası olması için başvurmuş ve bu isteği kabul edilmiştir. Bu işlem sonucunda, ata düğümün yavru listesine bu düğüm de eklenmiş ancak, yavru düğüm bunu öğrenememiştir. Bu yüzden,

isteğinin reddedildiğini sanarak, başka bir düğümden ata düğümü olmasını talep eder.

Bir süre sonra, bu düğümden de isteğine kabul cevabı alacak; böylece, iki ayrı düğüm, istekte bulunan düğümü yavru listesine eklemiş olacaktır. Bir düğümün iki ayrı atası olması şeklindeki bu durum, matematiksel olarak aşağıdaki gibi ifade edilebilir:

$$i \in CL_j \wedge i \in CL_k \tag{5.1}$$

Yani, i düğümü hem j düğümünün hem de k düğümünün yavru listesinde bulunmaktadır. Ama düğümün kendisi bunlardan yalnızca birini bilmektedir (*k* düğümünü atası olarak kaydetmiştir).

Bu problemi çözmek için izlenen yol şöyledir: Bir düğüm, kendi yavruları arasında bulunan bir düğümün, enerji seviyesi kendisininkinden yüksek başka bir düğümün yavru listesinde de bulunduğunu duyarsa, bu yavruyu kendi yavru listesinden çıkarmaktadır.

$$E_j < E_k \Rightarrow CL_j = CL_j - \{i\} \tag{5.2}$$

Bir başka çözüm yolu olarak, "ikincil ata" mekanizması getirilebilir. Düğümün asıl atasının yanında ikincil bir atası da olabilir. Asıl atanın ölmesi durumunda, hemen devreye girecek olan ikincil ata, ağacın kolaylıkla toparlanmasını (recovery) sağlayabilir.

5.2.1 Ağa Yeni Düğümlerin Eklenmesi

Kullanıcının ortama yeni düğümler yerleştirmesi veya daha önce çeşitli sebeplerle ağdan kopmuş olan bir düğümün yeniden ağa bağlanması gibi çeşitli sebeplerle, var olan ağa yeni düğümler eklenebilir. Böyle durumlarda, yeni eklenen düğümlerin de oluşturulmuş olan sorumluluk ağacına dahil edilmesi ve böylece yeni programı almalarının garanti altına alınması gerekir.

Yeni gelen düğüm, etrafına, kendisinin varlığını bildiren PR mesajını gönderir. Bu düğümü yavru olarak kabul etme potansiyeli olan düğümler, bu mesaja yanıt verir. Yeni düğüm, gelen bu mesajlardan en yüksek enerji seviyesine sahip olan düğümü tespit edip bu düğüme başvurur. Böylece, o düğümün yavrusu olarak ağaca dahil olmuş olur.

Kod yayma mekanizmasının hataya karşı dayanıklı (fault-tolerant) olması için, ölen bir ata düğümün fark edilmesi gerekir. Bunun için ilk akla gelen çözüm, belli aralıklarla veya program güncellenmesi sırasında, düğümlerin atalarına "yaşıyor musun" şeklinde bir mesaj göndermeleridir. Bir başka yöntem de, düğümlerin ağaç yapısındaki seviyelerini ve gelecek

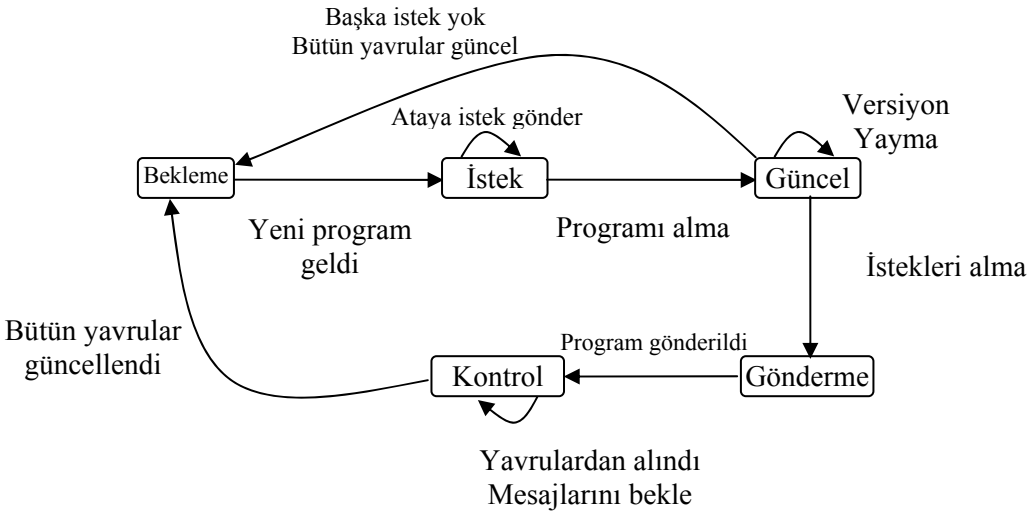
yeni programın boyutunu bilmelerini sağlamak olabilir. Böylece düğümlerin, yeni programın bulundukları seviyeye gelmesi için geçecek zamanı tahmin ederek, zamanlayıcılarını buna göre ayarlamaları sağlanabilir. Bu zaman sonunda, hâlâ program gelmezse, ata düğümlerini sorgulamaları sağlanabilir.

Hareketli düğümlerin bulunduğu ortamlarda ise, yeni gelen bir programın gönderilmesinden hemen önce, yeni bir ağaç oluşturma işlemi yerine, var olan ağacın kontrol edilmesi işlemi başlatılabilir. Bu işlem, yeni bir ağaç oluşturma işlemine göre çok daha hızlı olacak ve daha az kaynak sarf edecektir.

5.3 Kod Yayımı Evresi

Ağaç oluşturulduktan sonra, ağın güncellenmesi ihtiyacı doğduğu zaman, başlangıç düğümü yeni gelen kodun versiyon numarasını yayımlamaya başlar. Bu mesajı duyan düğümler, ağa yeni bir programın yüklenmekte olduğunu anlamış olur.

Düğümlerin kod yayımı sırasında bulundukları durumlar Şekil 5-6'da gösterilmektedir. Bir düğüm kendi programından daha yeni bir programa ait bir versiyon numarası duyduğunda, atasına yeni program için bir istek mesajı gönderir. İstek durumu için geçerli algoritma Şekil 5-7'de gösterilmiştir.



Şekil 5-6 Durum diyagramı

Yavru düğümlerinin herhangi birinden program isteği duyan bir düğüm, belli bir süre bekler. Bu süre sonunda, çevresinde program gönderen başka düğüm yok ise, programı etrafına yaymaya başlar. Bunu duyan eski program yüklü düğümler, ata düğümlerinden gönderilip

gönderilmediğine bakmaksızın, yeni programı almaya başlarlar. Burada, bir güvenlik mekanizması olmadığı varsayılmaktadır.

Yeni programı alan her düğüm, potansiyel bir gönderici durumuna gelir ve program versiyonunu etrafına yayar. Bu mesaj, aynı zamanda bu düğümün, ata düğümü için bir "ACK" mesajı işlevini görür: Ata düğüm, bu mesajla, yavru düğümün yeni programı almış olduğunu anlar ve bu düğümü "güncellendi" şeklinde işaretler.

```
For all s
   switch ( receive )
      case receive( newVersion, parent )
         if ( version > myVersion )
            sendRequest( parent )
         else sendACK( parent )
      case receive( newProgram )
         receiveProgram()
         sendNewVersion( version, s )
      end case
   end switch
```

Şekil 5-7 "İstek" durumu algoritması

Versiyon numarası gönderme işlemi, bütün yavru düğümler güncellenene kadar devam eder. Sonra gönderme işi durdurulur. Bu algoritma Şekil 5-8'de gösterilmiştir. Ayrıca, ata düğümünden kendi programının versiyonu ile aynı versiyon numarasını duyan bir düğüm, ata düğümüne "bende güncel program zaten var" anlamına gelecek bir "ACK" mesajı gönderir. Bu sayede, ata düğümün kendisini güncellenmiş olarak işaretlemesini sağlamış olur.

Düğümlere, ata düğümü olduğuna bakmaksızın, yeni programı istediği düğümden alma izni verilmiştir. Bu sayede önemli bir enerji tasarrufu sağlanmış olmaktadır. Aksi halde, bir düğüm duyabildiği halde yeni programı başka bir düğümden alamayacak, ata düğümünün programı yeniden göndermesi gerekecektir.

```
For all s
     sendNewVersion( version, s )
     notUpdated = childNumber
     while ( notUpdated > 0 )
        sendNewVersion( version, s )
        switch ( receive )
           case receive( request, child )
              if ( noOneSending )
                   sendProgram()
           end case
           case   receive( newVersion, child) OR
                  receive( ACK, child )
                notUpdated --
                markAsUpdated( child )
           end case
           end switch
        end while
```

Şekil 5-8 "Güncel" durumu algoritması

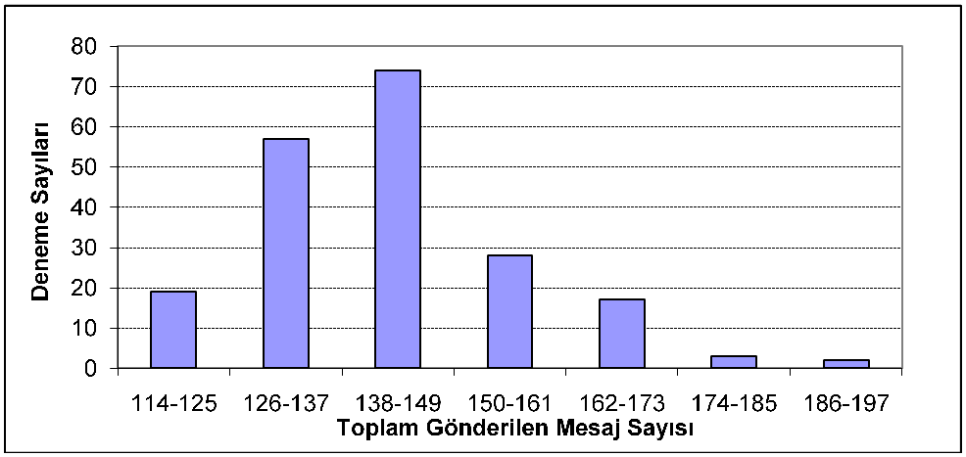
5.4 Elde Edilen Sonuçlar

Sonuçlar, algoritmanın TOSSIM (Levis, 2003) altında çalıştırılmasıyla elde edilmiştir. TOSSIM, üzerinde doğrudan TinyOS kodu çalıştırabilen ve programları bit seviyesinde simüle edebilen bir ayrık olay simülatörüdür. Simülasyonlar sırasında yapılan kabuller aşağıdaki gibidir:

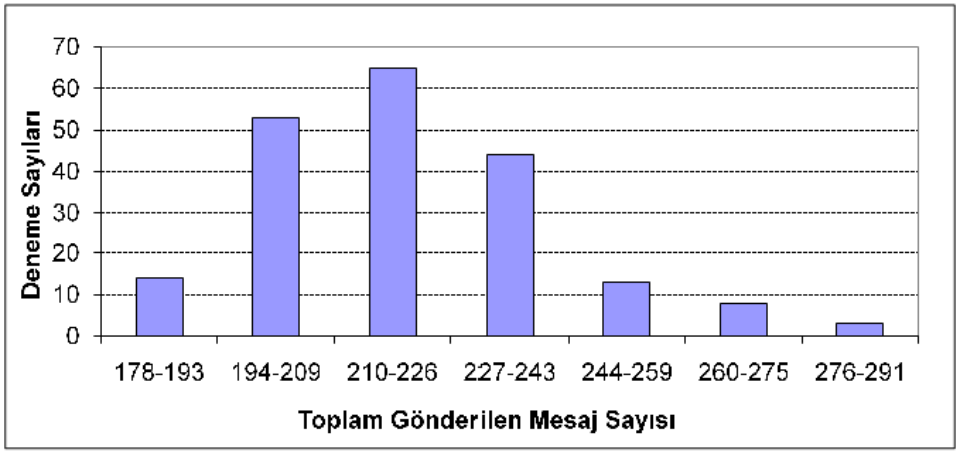
- Bütün simülasyonlarda, karşılaştırılabilir ve tekrar üretilebilir sonuçlar elde edebilmek amacıyla, statik bir grid topoloji kullanılmıştır.
- Kanal hatalarının MAC protokolü tarafından ele alındığı ve "gizli terminal problemi" (hidden terminal problem) olmadığı sürece, gönderilen bir mesajın bulunduğu hücredeki bütün düğümler tarafından başarıyla alındığı varsayıldı. Gizli terminal probleminin aşılması için de, mesaj gönderimlerinin önüne rasgele zamanlı gecikmeler konulmuştur.
- Ağa yayılacak kodun bir pakete sığacak kadar küçük olduğu varsayılmıştır.
- Düğümler, ata düğümlerini seçerken, sadece enerji seviyelerini dikkate almaktadır.
- Bir düğümün sahip olabileceği alt düğümlerin sınırı olan n, 5 olarak kabul edilmiştir.

5.4.1 Kararlılık

Algoritmanın, 4x10, 6x10 ve 10x10'luk grid topolojili ve aynı düğüm yoğunluğuna sahip ağlarda, 200'er kez çalıştırılmasıyla, düğümlerin tamamı tarafından toplam gönderilen mesaj sayıları açısından elde edilen sonuçlar Şekil 5-9, Şekil 5-10 ve Şekil 5-11'de gösterilmiştir.

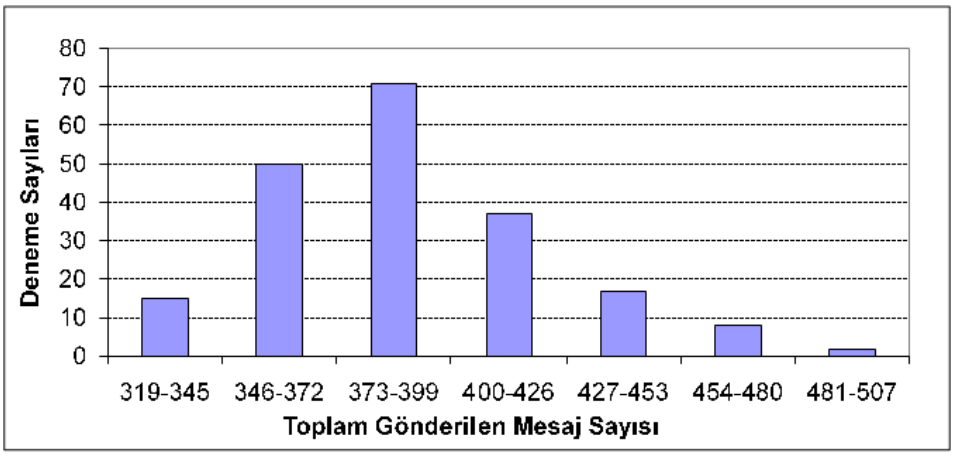


Şekil 5-9 "40 düğümlü ağ"da, toplam mesaj sayısı



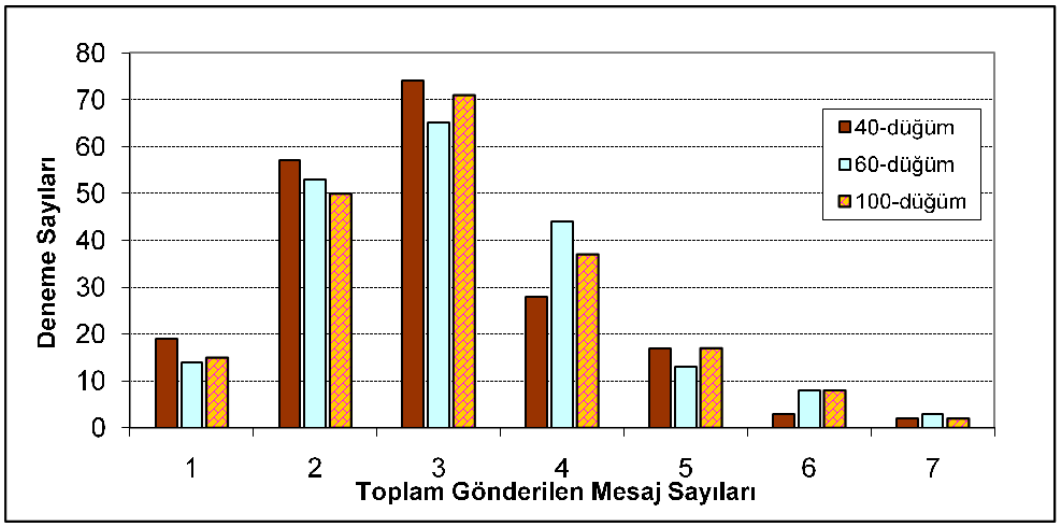
Şekil 5-10 "60 düğümlü ağ"da, toplam mesaj sayısı

Bu şekillerde, x-eksenlerindeki aralıklar, programın 200 kez çalıştırılması sonucunda elde edilen mesaj sayılarının, yedi eşit aralığa bölünmesi ile elde edilmiştir. Y-eksenlerinde ise, bu aralıklara karşılık gelen deneme sayıları görülmektedir. Örnek olarak Şekil 5-11 incelenirse, program 100 düğümlü ağda 200 kez çalıştırılınca, ortaya çıkan mesaj sayılarının; 15 denemede 319-345 arasında, 50 denemede 346-372 arasında, 71 denemede 373-399 arasında, 38 denemede 400-426 arasında, 17 denemede 427-453 arasında, 9 denemede 454-480 arasında ve 2 denemede 481-507 arasında olduğu görülür.



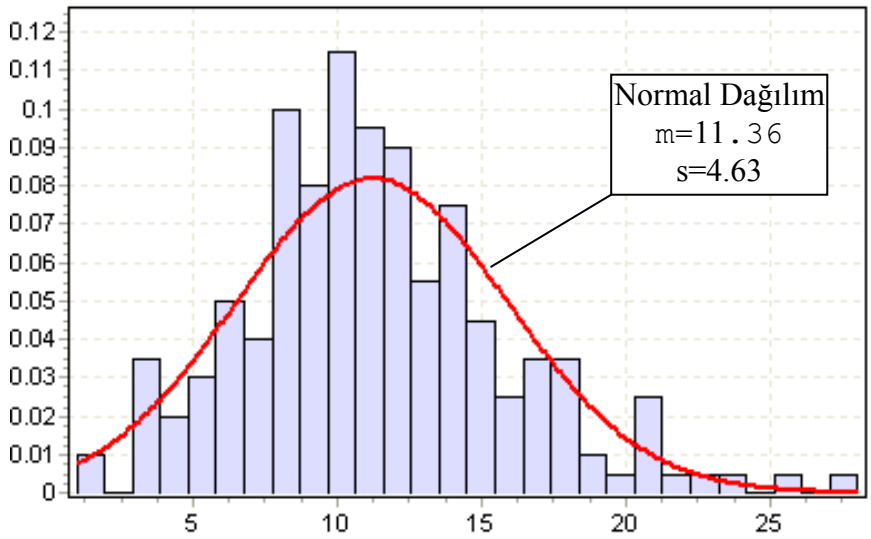
Şekil 5-11 "100 düğümlü ağ"da, toplam mesaj sayısı

Bu üç grafikte, x-ekseninde bulunan aralıklar 1'den 7'ye numaralandırılır ve grafikler bu şekilde birleştirilirse Şekil 5-12 elde edilir.



Şekil 5-12 "40, 60 ve 100 düğümlü ağlar"da toplam mesaj sayıları

100 düğümlü ağ için toplam gönderilen mesaj sayısı Şekil 5-13'te tekrar verilmiştir. Ancak, aralıklar bu sefer 28 eşit parçaya bölünmüş durumdadır. Şekilde de görülebileceği gibi, bu sonuçlar, ortalaması 11,36'ya, varyansı 4,63'e eşit olan, normal dağılıma uymaktadır.



Şekil 5-13 "100 düğümlü ağ"da toplam mesaj sayısı

Normal dağılımın olasılık dağılım fonksiyonu,

$$f(x) = \frac{1}{s\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} \tag{5.3}$$

şeklinde verilmektedir. Ortalama değer (mean) olan 11.36 değeri, 387-393 aralığına denk gelmektedir. Bu durum, Şekil 5-11 ile de uyumludur. 3. aralıkta, yani 373-399 aralığında, toplam gönderilen mesaj sayısı en yüksek seviyesine ulaşmaktadır.

Sonuçların normal dağılıma uyduğu değişik yollarla doğrulanabilir. Örneğin, toplam gönderilen mesaj sayısının 363-397 aralığında (8 ve 12. eşit aralıklar arasında) olması olasılığı, olasılık dağılım fonksiyonu kullanılarak aşağıdaki gibi bulunabilir:

$$P(8-12) = \int_{8}^{12} \frac{1}{s\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} = 0.41 \tag{5.4}$$

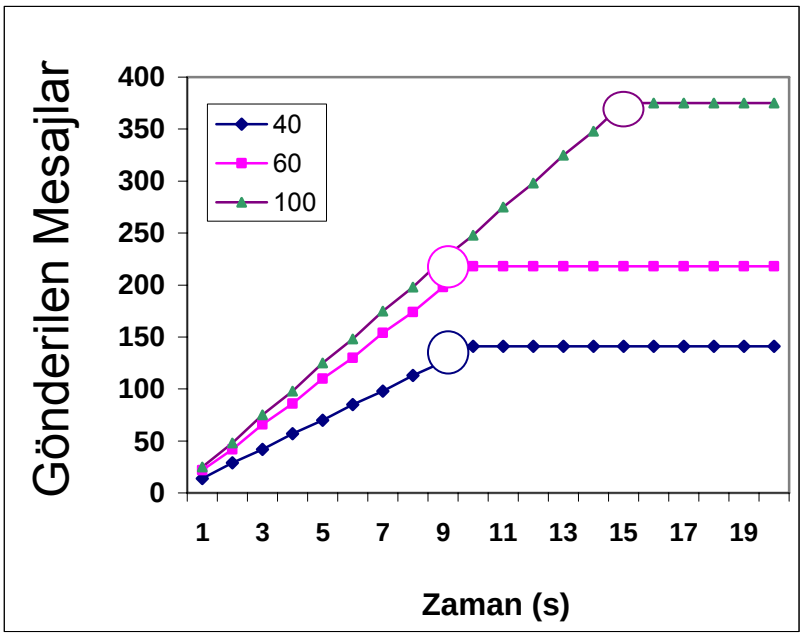
Bu sonuca göre, 200 denemenin yaklaşık 82 tanesinde, toplam gönderilen mesaj sayısı 363-397 arasında olmalıdır. Bu hesap, alınan sonuçlarla uyumlu bir durumdur; 200 denemenin 94 tanesinde sonuçlar bu aralıkta olmuştur.

5.4.2 Karşılaştırmalı Sonuçlar

Toplam gönderilen mesaj sayısı ve ağ güncellenme zamanı açısından, üç ayrı topoloji için

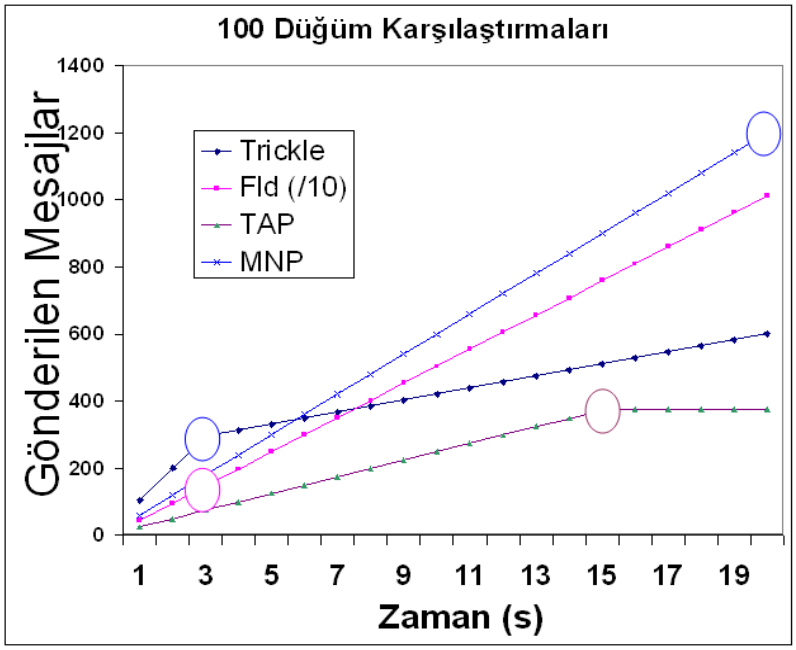
elde edilmiş olan sonuçların grafikleri Şekil 5-14'te gösterilmektedir. Grafiklerde de görülebileceği gibi, bir süre sonra toplam gönderilen mesaj sayısı sabitlenmektedir (grafikteki yuvarlaklar bu noktaları göstermektedir). Ağdaki bütün düğümler yeni programı aldıktan sonra, mesaj gönderimi durmaktadır. Bu durum, sürekli olan diğer algoritmalar için geçerli değildir.

Önerilen TAP isimli algoritma (Mutluoğlu, 2007); Trickle (Levis, 2004), Flooding Yöntemi (Fld) ve MNP (Kulkarni, 2005) algoritmaları ile karşılaştırılmıştır. Bütün algoritmaların 100 düğümlü aynı topolojiye sahip ağlarda çalıştırılmasıyla elde edilen sonuçlar Şekil 5-15'te verilmiştir (Şekilde Fld yöntemi için y-eksenindeki değerlerin 10 ile; MNP için de x-eksenindeki değerlerin 30 ile çarpılması gerekmektedir. Bir grafikte bütün algoritmaların sonuçlarının anlaşılır bir şekilde görülebilmesi için, eksenlerde böyle bir ayarlama yapılmıştır).



Şekil 5-14 "40, 60 ve 100 düğümlü ağlar"da mesaj sayısı grafikleri

Trickle ve Fld algoritmaları, ağı, TAP algoritmasına göre 4-5 kat daha hızlı güncelleyebilmektedir; ama, daha fazla mesaj gönderdikleri için, harcadıkları enerjileri daha çoktur. Çünkü, ağdaki güncelleme bittikten sonra da mesaj gönderimi devam etmektedir.



Şekil 5-15 "100 düğümlü ağ"da karşılaştırmalı sonuçlar

Grafikte de rahatlıkla görülebileceği gibi, en az mesaj gönderimi, dolayısıyla en az enerji sarfiyatı TAP algoritmasında olmaktadır; zira, mesaj gönderimi bütün ağ güncellendikten sonra tamamen durmaktadır.

En çok enerji, Fld yönteminde harcanmaktadır. Ağı en yavaş güncelleyen algoritma ise MNP algoritmasıdır. MNP, ağı 600 saniyede güncellerken ağın TAP algoritmasıyla güncellenmesi 15 saniye sürmektedir.

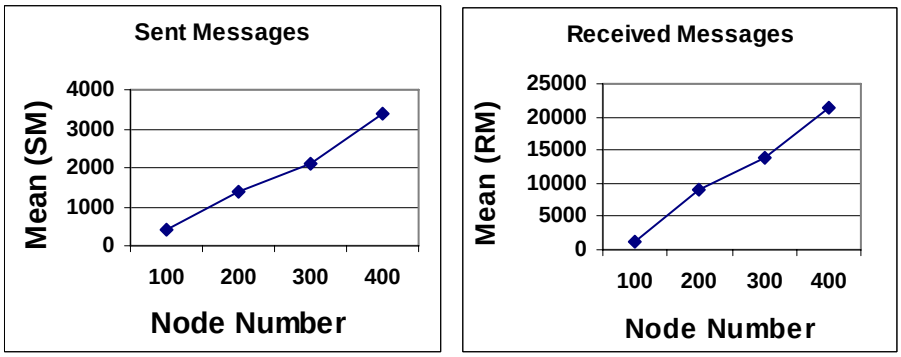
Çizelge 5-1'de, algoritmaların karşılaştırmalı sonuçları verilmiştir. Bir pakete sığan bir programı, en kısa sürede (3 saniye) Fld ve Trickle gönderebilmektedir. MNP, en güncel algoritmalardan biridir ama, ilân mekanizmasının getirdiği yük son derece yüksek olduğu için, bu algoritmada programın ağın tamamına yayılması çok yavaş olmaktadır (706 saniye). Çizelgede verilen toplam mesaj sayıları, hem program paketlerini hem de "ek yük" (overhead) olarak adlandırılan, ilân mesajlarını içermektedir. TAP haricindeki algoritmalar, sürekli oldukları için, getirdikleri ek yük, zaman geçtikçe sürekli artmaktadır.

Çizelge 5-1 Karşılaştırmalı sonuçlar

	Güncelleme Süresi (s)	Toplam Mesajlar	Program Paketleri	Ek Yük
Flooding	3	1452	242	1210
Trickle	3	285	112	173
MNP	706	1693	150	1543
TAP	15	380	35	345

5.4.3 Ölçeklenebilirlik (Scalability)

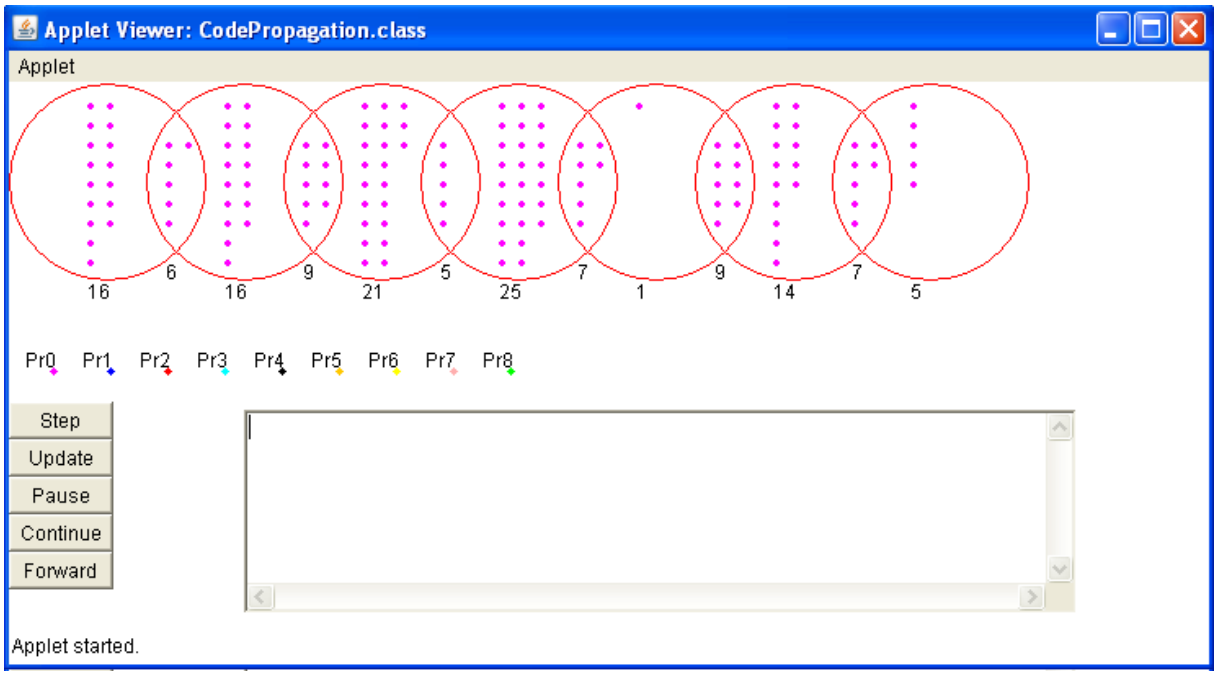
TAP algoritması, 200, 300 ve 400 düğümlü daha büyük ağlarda da çalıştırılıp sonuçlar alındı. Toplam gönderilen ve toplam alınan mesaj sayılarının ortalama değerleri açısından elde edilen sonuçlar Şekil 5-16'da verilmiştir. Telsiz ağlarda, gönderilen bir mesajın, aynı hücredeki mesaj alma devresi açık bütün düğümler tarafından alınacağı göz önünde tutulursa, alınan mesaj sayısının gönderilen mesaj sayısının yaklaşık yedi katına eşit olmasının nedeni anlaşılabilir. Grafiklerde, ölçeklenebilirlik açısından, önerilen algoritmada bir sorun olmadığı rahatlıkla görülebilmektedir. Düğüm sayısı arttıkça, toplam mesaj sayısı da nerdeyse doğrusal olarak artmaktadır.



Şekil 5-16 Büyük ağlardaki sonuçlar

5.5 Trickle Algoritmaları İçin İyileştirme Çalışmaları

Bölüm 4.2'de anlatılmış olan Trickle algoritmalarının (DPA, CPA ve τ-CA) iyileştirme çalışmaları için, öncelikle bu algoritmaların ve değişikliklerin sonuçlarının görülebileceği bir simülatör, java dilinde geliştirilmiştir. Simülatörün örnek bir ekran görüntüsü Şekil 5-17'de görülmektedir.



Şekil 5-17 Örnek simülatör ekranı

Bu simülatör, şekildekine benzer topolojiler için, duyarga ağlarında kod yayma denemeleri yapılmasına imkan vermiştir. Şekildeki daireler, ağdaki hücreleri; noktalar ise duyarga düğümlerini temsil etmektedir. Bir hücre içindeki bütün düğümler birbirlerini, hücrelerin kesişim bölgelerinde bulunan düğümler ise her iki hücredeki düğümleri duyabilmektedir.

5.5.1 Algoritmaların Orijinal Haliyle Elde Edilen Sonuçlar

Önce, üç algoritma, 400 düğümlü bir ağda, τ=10 değeri için çalıştırılmış ve Çizelge 5-2'deki sonuçlar elde edilmiştir. Çizelgedeki son sütunda, ağa bir paketlik yeni bir program yüklendiğinde, bu programın bütün ağa yayılma süreleri verilmiştir.

Çizelge 5-2 Karşılaştırma tablosu

Algoritma	Toplam Gönderilen Paket Sayısı	Güncellenme Zamanı (Simülasyon Saniyesi)
DPA	5700	120
CPA	6600	130
τ-CA	6400	100

5.5.2 Algoritmalarda Denenmiş Olan Değişiklikler

Algoritmaların nasıl geliştirilebileceğiyle ilgili aşağıdaki değişiklikler denenmiştir:

1) Algoritmalardaki p değerinin birer birer değil, geometrik olarak artırılması, (**exp**)

2) P değerinin düğüm yoğunluğu ile orantılı olarak artırılması, (**d**)
3) Bütün düğümlerin değil, sadece hücrelerin kesişim alanlarında bulunan düğümlerin kod yayması, (**intr**)
4) 2 ve 3'teki değişikliklerin aynı anda yapılması. (**intr-d**)

Bu değişiklikler, simülatörde $\tau = 5$ değeri için ve 300 zaman aralığında denenmiş ve Çizelge 5-3'teki sonuçlar elde edilmiştir.

Çizelge 5-3 İyileştirme sonuçları

	Orijinal alg.		**exp**		**d**		**intr**		**intr-d**	
	TG	**GZ**	**TG**	**GZ**	**TG**	**GZ**	**TG**	**GZ**	**TG**	**GZ**
DPA	2641	121	3623	232	1885	109	2321	85	1643	91
CPA	2417	113	2038	110	2030	93	2112	81	1847	98
τ-CA	2433	107	3592	268	2512	138	2143	89	2017	83

(**TG**: Toplam Gönderilen Paket Sayısı **GZ**: Güncelleme Zamanı (s.))

Bu çizelgede verilen değerlerin birincisi, toplam gönderilen vektör sayısını; ikincisi de güncelleme zamanını göstermektedir. Çizelgenin incelenmesinden de anlaşılacağı üzere, p değerinin geometrik olarak artırılması, sadece CPA algoritmasında; diğer iki değişiklik (d, intr) ise, bütün algoritmalarda iyileşme sağlamıştır. Bu son iki değişikliğin aynı anda uygulanması ise, bütün algoritmalarda en iyi sonuçları vermiştir.

Yukarıdaki simülasyonlarda, duyargaların, hücrelerin kesişim alanları içinde olup olmadıklarını bildikleri varsayılmıştır. Oysa duyarga ağlarında bu bilgi mevcut değildir. Duyargaların bu bilgiyi elde edebilmesi için Şekil 5-18'deki algoritma geliştirilmiş ve değişik topolojilerde çalıştığı test edilmiştir. Bu algoritma çalıştıktan sonra, ağdaki her düğüm, hop sayısını ve hücrelerin kesişimi içerisinde olup olmadığını belirlemiş olmaktadır.

```
hop count = -1, başlangıç düğümü için 0
intr = false
başlangıç düğümü hop count'unu gönderir
bir hop count alan her düğüm
     hop count -1 ise
          hop count'unu alınan hop count + 1 yapar
          n zaman aralığı bekler
          hop count'unu gönderir
alınan hop count kendi hop count'undan büyük ise
     intr = true
```

Şekil 5-18 Kesişim belirleme algoritması

Bu iyileştirme çalışmalarından çıkan en önemli sonuç, kod yayımı sırasında, program paketlerini yayacak düğümlerin seçiminde, ağdaki hücrelerin kesişim kümelerinde bulunan düğümlerin öncelikli olmasının, performansı artırdığı gerçeğidir. Bu düğümler, daha çok düğümü duyabildikleri için, programı daha hızlı ve verimli yayabilmektedir.

6. ANALİZ ÇALIŞMALARI

Bu bölümde; kod yayma açısından, bir ağın gönderebileceği toplam program sayısının belirlenmesi; düğüm sayısı verilmiş bir ağda, en uygun ağ altyapısının tespiti ve kod yayma algoritmasının bu altyapıya uygun olarak şekillendirilmesi konuları incelenecektir.

Analiz boyunca kullanılacak simgeler aşağıdaki gibidir:

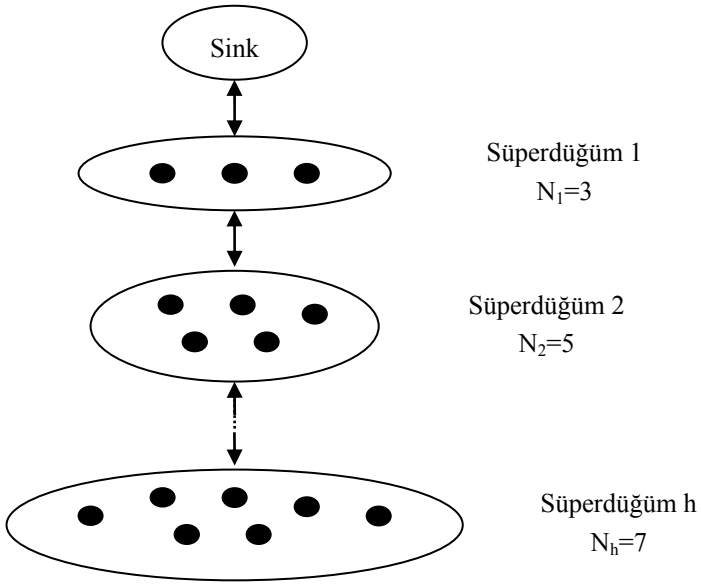
- E_{max}: Bir düğümün sahip olabileceği maksimum enerji seviyesi,
- T: Bir mesajın gönderilmesi için gereken enerji miktarı,
- k_{max}: Enerjisi E_{max} seviyesinde olan bir düğümün gönderebileceği mesaj sayısı:

$$k_{\max} = \frac{E_{\max}}{T} \tag{6.1}$$

- p: Paket seviyesindeki hata oranı (olasılığı),
- N: Gönderilecek programın paket sayısı.

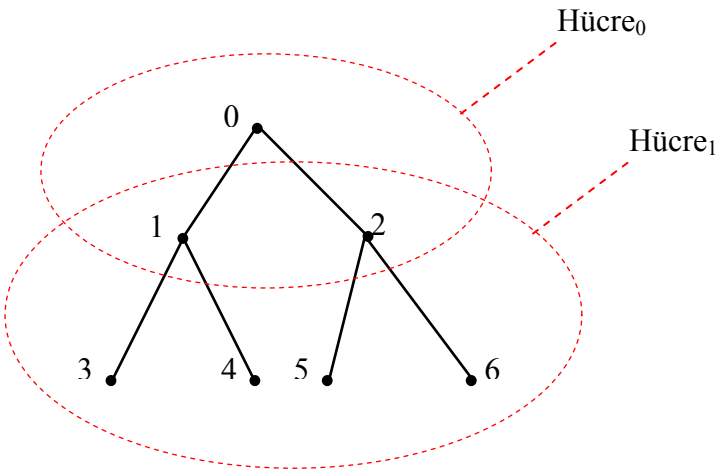
Analiz için, ağ topolojisi, kayıp paketler gibi konularda yapılmış kabuller aşağıdaki gibidir:

- Her düğümün başlangıçta sahip olduğu enerji seviyesi E_{max}'tir,



Şekil 6-1 Varsayılan ağ topolojisi örneği

- Düğümler Şekil 6-1'deki gibi, h adet "süperdüğüm"e (SD) bölünmüş durumdadır. SD_i'nin düğüm sayısı N_i'dir ve $N_1 + N_2 + N_3 + \ldots N_h$ toplamı N'ye eşittir.
- Bir süperdüğümdeki düğümler, hem kendi süperdüğümlerindeki düğümlerle, hem de komşu süperdüğümlerdeki düğümlerle aynı hücrede bulunmaktadır. Yani SD_i'deki düğümler SD_{i-1}, SD_i ve SD_{i+1}'deki düğümleri duyabilmektedir.
- Programın gönderilmesi sırasında, bir süperdüğümdeki alıcı düğümlerin kaybettikleri program paketlerinin oluşturduğu kümeler, ayrışık kümelerdir (disjoint sets); yani hepsi ayrı ayrı paketleri kaybetmektedir.



Şekil 6-2 Örnek topoloji

Analize, Şekil 6-2'deki , iki hoplu basit bir topoloji ile başlanacaktır. Kesik çizgilerle belirtilmiş olan hücrelerdeki bütün düğümler birbirlerini duyabilmektedir. Şekildeki ağın, SD_0: {0}; SD_1: {1, 2} ve SD_2: {3, 4, 5, 6} olacak şekilde, 3 süperdüğümden oluştuğu rahatlıkla görülebilir.

Bulunmaya çalışılan şey, N sayıdaki paketten oluşan bir programın, ağdaki bütün düğümlere ulaştırılabilmesi için, gönderilmesi gereken toplam mesaj sayısının beklenen değeridir. Bunun için önce, bir gönderenin (0) ve iki alıcının (1, 2) bulunduğu birinci hoptaki mesaj sayısı bulunucaktır.

6.1 1-N Analizi

Paket seviyesindeki hata oranı *p* iken, düğüm 0'ın N paketlik programın tamamını hatasız olarak düğüm 1 ve düğüm 2'ye ulaştırması için, göndermesi gereken toplam mesaj sayısının beklenen değeri, aşağıdaki gibi bulunabilir:

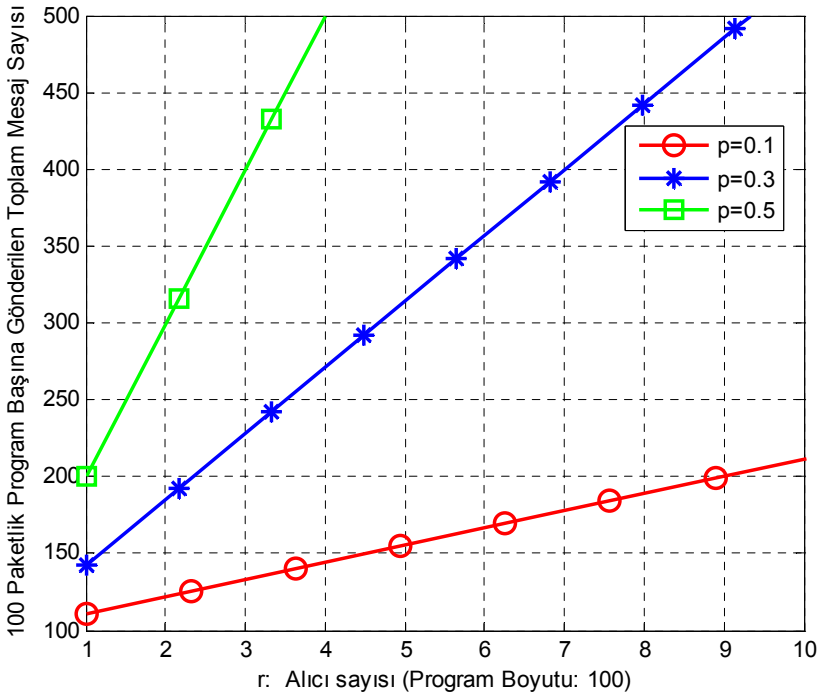
Önce düğüm 0, N paketlik programı göndermek üzere, N adet mesajı komşularına gönderir. Bir mesajla, tam olarak bir paketin gönderilebildiği varsayılmıştır. Düğüm 1, iletişim kanalındaki hatalardan dolayı (p), bu N mesajın Np tanesini kaybeder. Aynı şekilde, düğüm 2 de Np adet mesajı kaybeder. Daha önce belirtildiği gibi, iki düğümün kaybettiği mesajların, birbirinden ayrı mesajlar olduğu varsayılmıştır. Daha sonra düğüm 0, alamadıkları mesajları, bu düğümlere tekrar gönderir: Np mesajı düğüm 1'e, Np mesajı da düğüm 2'ye gönderir. İki düğüm de, bu gelen Np mesajın Np^2 tanesini kaybeder. Düğüm 0, alamadıkları mesajları

tekrar gönderir ve bu işlem iki düğüm de mesajların tamamını başarılı bir şekilde alana kadar devam eder.

Düğüm 0'ın gönderdiği mesajların sayısının beklenen değeri (N_x) böylece şu şekilde bulunabilir:

$$N_x = N + 2Np + 2Np^2 + 2Np^3 + \ldots = N\frac{1+p}{1-p} \tag{6.2}$$

(6.2) eşitliği, bir gönderen ve r adet alıcı olan durum için genelleştirilirse (6.3) elde edilir. Bu durumda, her aşamada, r alıcı, kendilerine gönderilen K mesajın Kp tanesini kaybederler (kaybedilen mesaj kümelerinin yine ayrışık olduğu kabul ediliyor).



Şekil 6-3 Bir program için gönderilen toplam mesaj sayısı

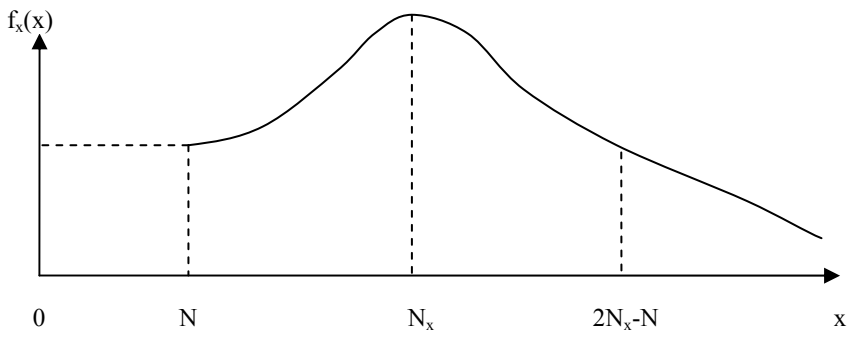
$$N_x = N + rNp + rNp^2 + rNp^3 + \ldots = N\frac{1+(r-1)p}{1-p} \tag{6.3}$$

Değişik p değerleri için bu fonksiyon çizilirse, Şekil 6-3'teki grafik elde edilir. P değeri arttıkça grafiklerin eğimlerindeki artış şekilde görülebilmektedir. Bu artış, (6.3) eşitliği (6.4) şeklinde yeniden yazılırsa, rahatlıkla görülebilir. Bu eşitlikten anlaşılabileceği gibi, grafiklerin eğimi p/(1-p) ile orantılıdır ve p arttıkça eğim üssel olarak (exponentially) artmaktadır. Örneğin, eğim p=0,5 iken N'ye; p=0,8 iken 4N'ye ve p=0,9 iken 9N'ye eşittir. Yine bu

eşitlikte, iletişim kanalındaki hata oranı arttıkça, başarılı gönderim yapmak için gönderilmesi gereken toplam mesaj sayısının üssel olarak arttığı görülmektedir.

$$N_x = N\left(\frac{p}{1-p}r+1\right) \tag{6.4}$$

(6.2) eşitliği, toplam gönderilmesi gereken mesaj sayısının beklenen değerini vermektedir. Bu sayı, mesaj kaybının söz konusu olmadığı en iyi durumda, N'ye eşittir. Diğer bir deyişle, N adet mesaj gönderimi, mesajların iki alıcı tarafından da başarılı olarak alınmasına yetecektir. En kötü durumda ise, hiçbir mesaj yerine ulaşamayacak, gönderilmesi gereken mesaj sayısı sonsuza yaklaşacaktır. Toplam gönderilen mesaj sayısını temsil eden rasgele değişkene (r.d.) x ismi verilirse, bu değişken, olasılık dağılım fonksiyonu (probability distribution function) $f_x(x)$, Şekil 6-4'teki gibi olan, beklenen değeri N_x'e eşit bir değişken olacaktır.

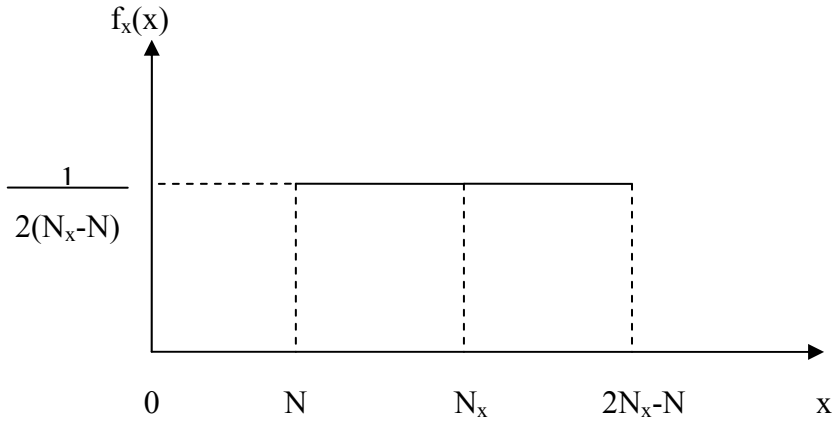


Şekil 6-4 x rasgele değişkeninin olasılık dağılım fonksiyonu

Analizin basitleştirilmesi için, x'in, olasılık dağılım fonksiyonu Şekil 6-5'te verilen ve beklenen değeri N_x olan, bir düzgün rasgele değişken (uniform random variable) olduğu varsayılmıştır. Analiz, beklenen değerler üzerinden yapıldığı için, bu varsayımın, analizin doğruluğunu önemli bir oranda etkilemediği düşünülmektedir.

x, düğüm 0'ın N paketlik bir programı, başarılı bir şekilde düğüm 1 ve 2'ye gönderebilmesi için, göndermesi gereken toplam mesaj sayısını temsil etmektedir. 0 düğümünün, enerjisi bitene kadar, bu iki alıcısına gönderebileceği toplam program sayısını temsil eden y rasgele değişkeni ise, (6.5) eşitliğinde verilmiştir.

$$y = \frac{k_{max}}{x} \tag{6.5}$$



Şekil 6-5 Düzgün rasgele değişken x'in olasılık dağılım fonksiyonu

Bulunması gereken, y değişkeninin beklenen değeridir. y = g(x) şeklinde, başka bir rasgele değişkenin fonksiyonu olarak verilen bir rasgele değişkenin beklenen değeri $\int_{-\infty}^{\infty} g(x) f_x(x) dx$ formülüyle bulunur. Buradan, y rasgele değişkeninin beklenen değeri aşağıdaki gibi hesaplanır:

$$E(y) = \int_{-\infty}^{+\infty} \frac{k_{\max}}{x} f_x(x) dx = \int_{N}^{2N_X - N} \frac{1}{2(N_X - N)} \frac{k_{\max}}{x} dx \tag{6.6}$$

$$E(y) = \frac{k_{\max}}{2(N_x - N)} \ln x \Big|_{N}^{2N_x - N} \tag{6.7}$$

$$E(y) = \frac{k_{\max}}{2(N_x - N)} \ln(2\frac{N_x}{N} - 1) \tag{6.8}$$

Bu eşitlikte, N_x, (6.3)'teki değeriyle değiştirilirse,

$$E(y) = \frac{k_{\max}}{2N(\frac{1+(r-1)p}{1-p} - 1)} \ln(2\frac{1+(r-1)p}{1-p} - 1) \tag{6.9}$$

$$E(y) = \frac{k_{\max}}{2N(\frac{pr}{1-p})} \ln(\frac{1+(2r-1)p}{1-p}) \tag{6.10}$$

sonucu elde edilir.

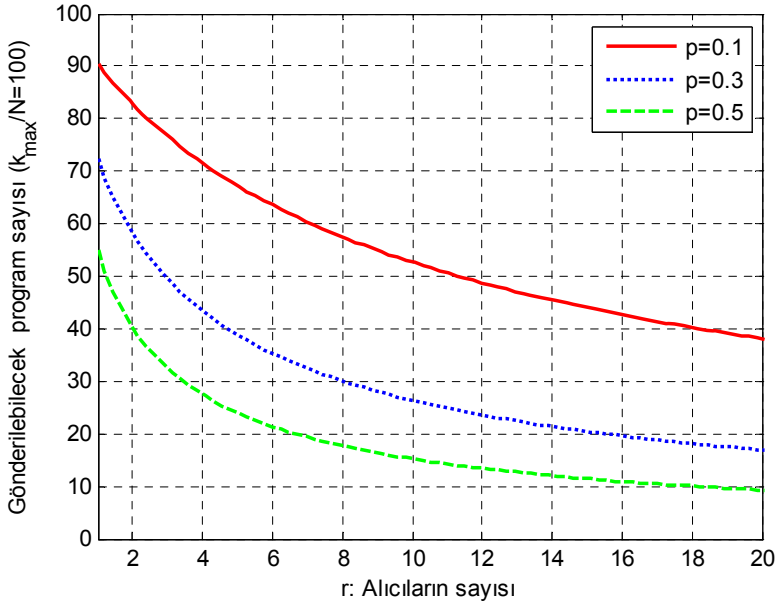
Analizin devamında, örnek olarak k_{max}=10000 ve N=100 olarak kabul edilmiştir. (6.10)

eşitliği, bu değerler kullanılarak, değişik p değerleri için tekrar yazılırsa, aşağıdaki fonksiyonlar elde edilir:

$$p = 0.1 \Rightarrow E(y) = \frac{450}{r} \ln\left(\frac{2}{9}r + 1\right) \tag{6.11}$$

$$p = 0.3 \Rightarrow E(y) = \frac{350}{3r} \ln\left(\frac{6}{7}r + 1\right) \tag{6.12}$$

$$p = 0.5 \Rightarrow E(y) = \frac{50}{r} \ln(2r + 1) \tag{6.13}$$



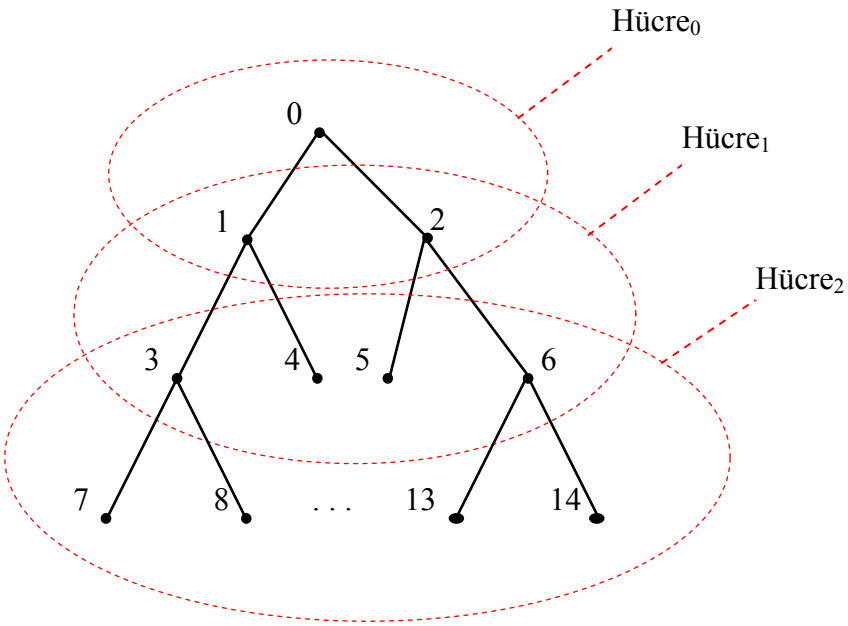
Şekil 6-6 Değişik p değerleri için, toplam gönderilebilecek program sayıları

Bu fonksiyonlar Şekil 6-6'da çizilmiştir. Alıcı sayısı arttıkça, gönderilebilecek toplam program sayısının, her durumda, üssel olarak azaldığı görülmektedir. Bu azalma, alıcı sayısının fonksiyonlarda paydada olmasından kaynaklanmaktadır. Alıcı sayısının artması, paket kaybeden düğüm sayısının artması demektir. Bu da, gönderenin, daha çok sayıda düğüme, kaybettikleri paketleri tekrar göndermek zorunda kalacağı anlamına gelir. Alıcı sayısı arttıkça, gönderilebilecek toplam program sayısında meydana gelen azalma, normalde bu analizde bulunandan daha az olacaktır. Çünkü bu analizde, kaybedilen paket kümelerinin ayrışık olduğu kabul edilmiştir. Normal şartlar altında, çok sayıda alıcının kaybettiği paketlerin kesişim kümeleri boş olmayacak; böylece gönderici, bir mesajla birden çok alıcıya eksik kalan paketlerini gönderebilecektir. Kaybedilen paket kümelerinin ayrışık olmaması

durumu, simülasyon aşamasında ele alınacaktır.

6.2 N-N Analizi

Düğüm 0, programı hatasız olarak bir sonraki hoptaki düğümlerin tamamına gönderdikten sonra, aynı işlem bir sonraki hopta tekrarlanacaktır.



Şekil 6-7 Örnek topoloji (devamı)

Yeni programı alan, düğüm 1 ve düğüm 2, bunu, 3, 4, 5 ve 6 düğümlerinden oluşan bir sonraki süperdüğüme göndermeye başlar. Bu aşamada, önceki aşamadaki gibi 1 gönderen değil, 2 gönderen vardır. Kod yayma algoritmasına göre, düğümler, ihtiyaç duydukları program paketlerini, "ata"ları olup olmadıklarına bakmaksızın, istedikleri düğümlerden alabilmektedirler. Ağın ömrünü uzatmak amacıyla, gönderici süperdüğüm içindeki düğümler, program paketlerini, dönüşümlü olarak göndermektedir. Sürekli aynı düğümün program göndermesi durumunda, diğer düğümün enerjisi henüz bitmemişken, program gönderen düğüm ölecek ve ağın ömrü sona ermiş olacaktır. Oysa, belli aralıklarla, enerjisi yüksek olanın program gönderme işini üstlenmesi, ağın ömrünü, iki düğümün enerjisi bitene kadar uzatmış olacaktır.

Bu durumda, bu iki düğüm, bir sonraki süperdüğümdeki düğümlere toplam kaç program gönderebilecektir? Bu sayının bulunabilmesi için, düğümlerin program gönderme işini, dönüşümlü değil de, sırayla yaptıkları varsayılmıştır. Yani, önce düğüm 1, enerjisi bitene

kadar program paketlerini göndermekte; o ölünce, bu işlevi, enerjisi bitene kadar düğüm 2 yerine getirmektedir. Bu varsayım, hesaplama sonucunu değiştirmeyecektir. Böylece bu aşama, biri ilk düğümün, diğeri de ikinci düğümün ölümlerine kadar geçen süreleri kapsayan iki bölümden oluşacak hale gelir.

Her iki bölümde de 1 gönderen ve 4 alıcı bulunmaktadır. (6.3) eşitliğinde, r yerine 4 konulursa, bir program için gönderilmesi gereken mesaj sayısının beklenen değeri olan N_x aşağıdaki gibi bulunur:

$$N_x = N\frac{1+(r-1)p}{1-p} = N\frac{1+3p}{1-p} \tag{6.14}$$

Düğüm 1'in, N paketlik bir programı başarılı olarak bir sonraki süperdüğüme gönderebilmesi için, göndermesi gereken toplam mesaj sayısını temsil eden rasgele değişkene x_2 ismi verilirse (beklenen değeri N_x olan düzgün rasgele değişken),

$$y_2 = \frac{k_{max}}{x_2} \tag{6.15}$$

ile verilen rasgele değişken, düğüm 1'in gönderebileceği program sayısını verecektir. Süperdüğüm içinde iki düğüm olduğu için, süperdüğümün gönderebileceği toplam program sayısını temsil eden rasgele değişken, **z** olarak isimlendirilen yeni bir değişkendir. Bu değişken için,

$$z = y_2 + y_2 \tag{6.16}$$

eşitliği geçerli olacaktır.

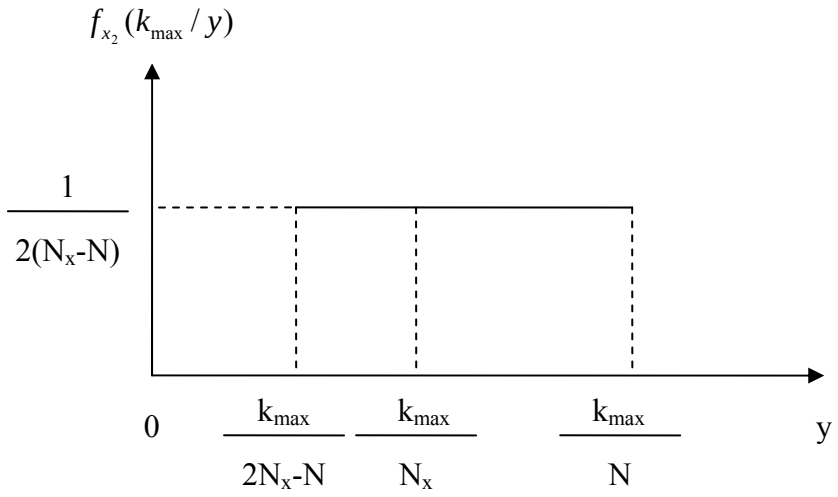
y = g(x) şeklindeki bir rasgele değişkenin olasılık dağılım fonksiyonu aşağıdaki gibidir ($x = g^{-1}(y) = h(y)$) :

$$f_y(y) = f_x(x)\left|\frac{dx}{dy}\right| = f_x[h(y)]\left|\frac{dh(y)}{dy}\right| \tag{6.17}$$

Bu durumda, y_2 r.d.'nin olasılık dağılım foksiyonu (6.18)'deki gibi bulunabilir.

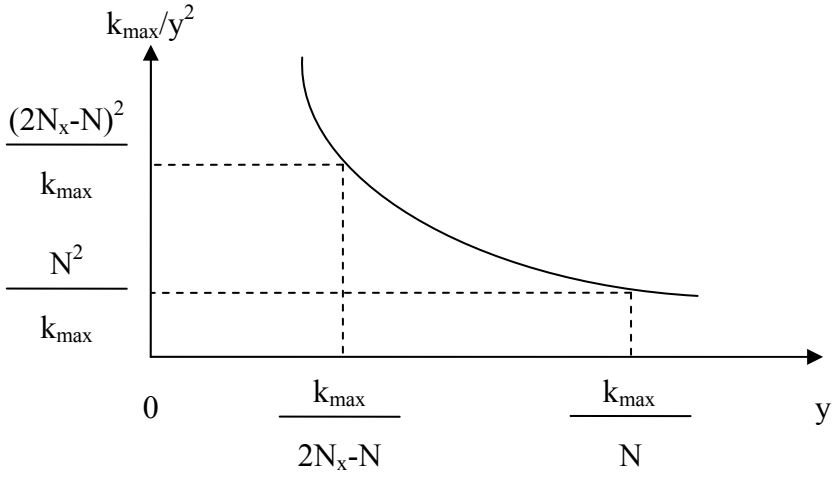
$$f_{y_2}(y) = f_{x_2}(\frac{k_{max}}{y})\left|\frac{d(\frac{k_{max}}{y})}{dy}\right| = f_{x_2}(\frac{k_{max}}{y})\frac{k_{max}}{y^2} \tag{6.18}$$

Şekil 6-5'teki x_2 r.d.'nin olasılık dağılım fonksiyonundan yararlanılarak $f_{x_2}(\frac{k_{max}}{y})$ fonksiyonu çizilirse, Şekil 6-8 elde edilir.



Şekil 6-8 $f_x(1/y)$ fonksiyonu

k_{max}/y^2 fonksiyonu ise Şekil 6-9'daki gibidir.

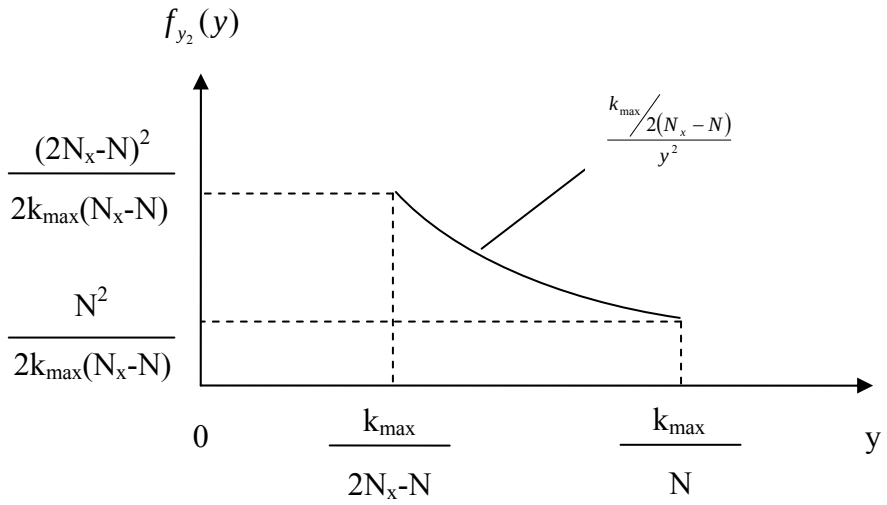


Şekil 6-9 k_{max}/y^2 fonksiyonu

Bu iki fonksiyonun çarpımı, y_2 r.d.'nin olasılık dağılım fonksiyonunu verir (6.18). Bu fonksiyon, Şekil 6-10'da görülmektedir.

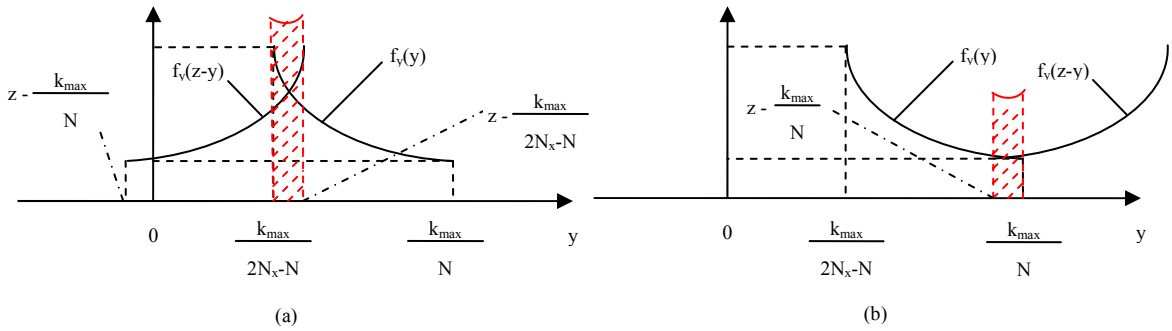
(6.16) ile y_2+y_2 şeklinde verilen z r.d.'nin beklenen değeri E(z), değişik şekillerde bulunabilir:

I) Önce, z'nin olasılık dağılım fonksiyonu $f_z(z)$, daha sonra $\int_{-\infty}^{\infty} z f_z(z) dz$ formülü ile, beklenen değer olan E(z) bulunur. Z rasgele değişkeni, y_2 değişkeninin kendisiyle toplanmasından elde edildiği için, olasılık dağılım fonksiyonu $f_z(z)$, y_2'nin dağılım fonksiyonunun, (6.19) eşitliğindeki gibi, kendisiyle konvolüsyonundan elde edilir (Şekil 6-11).



Şekil 6-10 y_2 rasgele değişkeninin olasılık dağılım fonksiyonu

$$f_z(z) = \int_{-\infty}^{\infty} f_y(y) f_y(z-y) dy \tag{6.19}$$

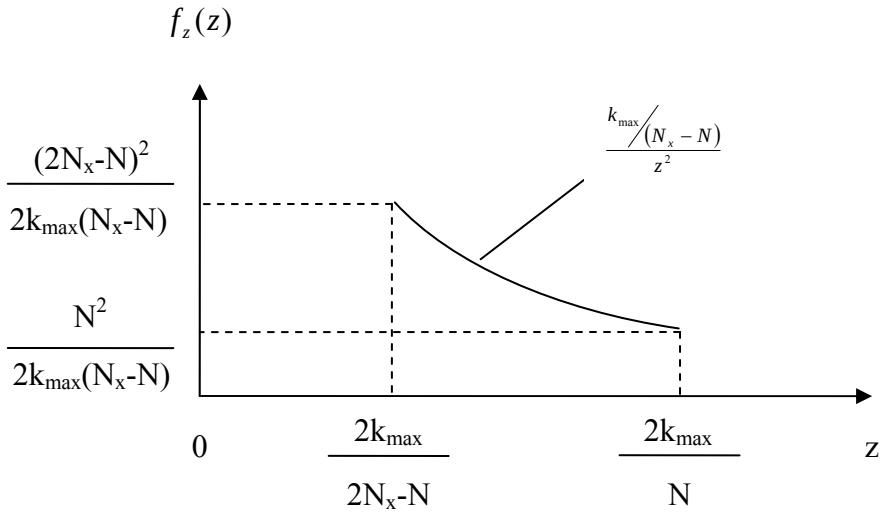


Şekil 6-11 Olasılık dağılım fonksiyonunun, konvolüsyon yöntemiyle bulunması

II) Z'nin olasılık dağılım fonksiyonu olan $f_z(z)$, $z = y_2+y_2$ eşitliğinin $z = 2y_2$ şeklinde yazılabileceği dikkate alınarak ve (6.17) kullanılarak aşağıdaki gibi bulunabilir,

$$f_z(z) = f_{y_2}(y)\left|\frac{dy}{dz}\right| = f_{y_2}\left(\frac{z}{2}\right)\left|\frac{d(z/2)}{dz}\right| = \frac{1}{2} f_{y_2}\left(\frac{z}{2}\right) \tag{6.20}$$

Bu fonksiyonun grafiği, Şekil 6-12'deki gibidir.



Şekil 6-12 z rasgele değişkeninin olasılık dağılım fonksiyonu

$E(z)=\int_{-\infty}^{\infty} z f_z(z)dz$ formülü kullanılarak, z'nin beklenen değeri aşağıdaki gibi bulunabilir:

$$E(z)=\int_{-\infty}^{\infty} z f_z(z)dz=\int_{2k_{max}/(2N_x-N)}^{2k_{max}/N}\frac{k_{max}}{(N_x-N)z}dz=\frac{k_{max}}{(N_x-N)}\ln z\Big|_{2k_{max}/(2N_x-N)}^{2k_{max}/N} \tag{6.21}$$

$$E(z)=\frac{k_{max}}{(N_x-N)}\ln\frac{(2N_x-N)}{N}=\frac{k_{max}}{(N_x-N)}\ln(2\frac{N_x}{N}-1) \tag{6.22}$$

(6.8)'den anlaşılabileceği gibi, bulunan sonuç, y_2 r.d.'nin beklenen değerinin 2 katına eşittir.

III) y = g(x) şeklinde, başka bir rasgele değişkenin fonksiyonu olarak verilen bir rasgele değişkenin beklenen değeri $\int_{-\infty}^{\infty} g(x)f_x(x)dx$ olarak bulunabilmektedir. Bu bilgi ve $z = 2y_2$ bilgisi dikkate alınırsa, z'nin beklenen değeri (6.23)'teki gibi de bulunabilir:

$$E(z)=\int_{-\infty}^{\infty} y f_{y_2}(y)dy=\int_{k_{max}/(2N_x-N)}^{k_{max}/N}\frac{2k_{max}}{2(N_x-N)y}dy=\frac{k_{max}}{(N_x-N)}\ln z\Big|_{2k_{max}/(2N_x-N)}^{2k_{max}/N} \tag{6.23}$$

Elde edilen sonuç (6.21) ile aynıdır.

IV) x ve y iki rasgele değişken olmak üzere,

$$E(x+y)=E(x)+E(y) \tag{6.24}$$

eşitliğinin geçerli olduğu bilinmektedir. Yani, iki rasgele değişkenin toplamından meydana gelen bir rasgele değişkenin beklenen değeri, bu iki rasgele değişkenin ayrı ayrı beklenen değerlerinin toplamına eşittir. O halde, **z** rasgele değişkeninin beklenen değeri,

$$E(z) = E(y_2) + E(y_2) = 2E(y_2) \tag{6.25}$$

ve (6.8)'deki E(y) değeri kullanılırsa, E(z) aşağıdaki gibi bulunur:

$$E(z) = \frac{k_{\max}}{(N_x - N)} \ln(2\frac{N_x}{N} - 1) \tag{6.26}$$

Düğümlerin herhangi bir düğümden program almalarına izin verilmemiş olsaydı, yani düğümler sadece kendi ata düğümlerinden program alabiliyor olsalardı, yukarıda bulunmuş olan değer, en iyi durumda, bu değerin yarısı kadar olabilecekti. En iyi durum, yavru düğümlerin eşit olarak paylaşıldığı durumdur. Bu durumda, düğümlerin yavrularına gönderebilecekleri program sayıları eşit olacak, ancak gönderdikleri programları sadece kendi yavruları alabildiği için, süperdüğümün gönderebildiği program sayısı, bu sayıların toplamına değil, sayılardan birine eşit olacaktır. Yavru düğümlerin eşit paylaşılmadığı durumda ise, bu sayılardan küçük olan belirleyici olacaktır.

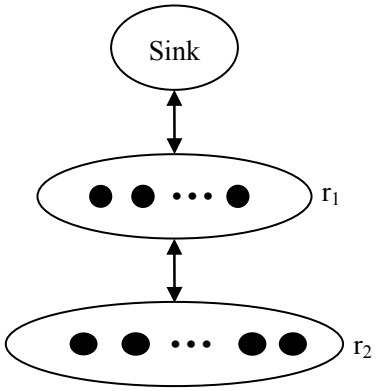
6.3 En İyi Topoloji Analizi

Düğüm sayısı verilmiş bir ağda, en iyi topolojiyi bulma işlemi için yukarıdaki analizden yararlanılabilir. Bu işlemde hedeflenen şey, ağın, program gönderme anlamında en uzun ömre sahip olması için, en uygun hop sayısının ve hoplardaki düğüm sayılarının bulunmasıdır.

Analize, şu sorunun yanıtını bulmaya çalışmakla başlanabilir: Birinci hoptaki düğüm sayısı belli iken, ağın yaşam süresini azaltmayacak şekilde, ikinci hoptaki düğüm sayısı ne kadar artırılabilir? Yani bu hopta en çok kaç düğüm olabilir? Bu sayı, ikinci hopta gönderilebilecek program sayısı, birinci hoptakine eşit olduğu zaman, yani

$$r_1 E(r_2) = E(r_1) \tag{6.27}$$

eşitliği geçerli iken ortaya çıkacaktır.



Şekil 6-13 İki hoplu topoloji örneği

Bu eşitlikte, r_1 ve r_2, Şekil 6-13'teki gibi, sırasıyla birinci ve ikinci hoptaki düğüm sayılarını temsil etmektedir. E(r) ise, bir tane gönderen ve r tane alıcı olması durumunda, gönderilebilecek program sayısının beklenen değerini göstermektedir.

(6.10), (6.27)'de yerine konulursa, aşağıdaki eşitlikler elde edilir:

$$r_1 \frac{k_{\max}}{2N\frac{pr_2}{1-p}} \ln\frac{1+(2r_2-1)p}{1-p} = \frac{k_{\max}}{2N\frac{pr_1}{1-p}} \ln\frac{1+(2r_1-1)p}{1-p} \tag{6.28}$$

$$\frac{1}{r_2} \ln\frac{1+(2r_2-1)p}{1-p} = \frac{1}{r_1^2} \ln\frac{1+(2r_1-1)p}{1-p} \tag{6.29}$$

$$r_2 \ln\left(\frac{2p}{1-p} r_1 + 1\right) = r_1^2 \ln\left(\frac{2p}{1-p} r_2 + 1\right) \tag{6.30}$$

Değişik p değerleri için (6.30)'un alacağı şekiller aşağıdaki gibidir:

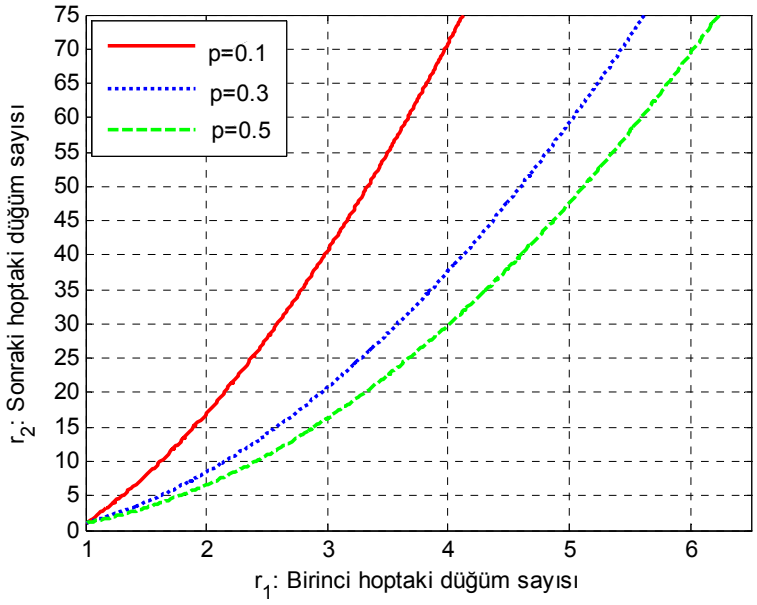
$$p = 0.1 \Rightarrow r_2 \ln(\frac{2}{9} r_1 + 1) = r_1^2 \ln(\frac{2}{9} r_2 + 1) \tag{6.31}$$

$$p = 0.3 \Rightarrow r_2 \ln(\frac{6}{7} r_1 + 1) = r_1^2 \ln(\frac{6}{7} r_2 + 1) \tag{6.32}$$

$$p = 0.5 \Rightarrow r_2 \ln(2r_1 + 1) = r_1^2 \ln(2r_2 + 1) \tag{6.33}$$

Bu fonksiyonlar çizilirse, Şekil 6-14'teki grafikler elde edilir. Grafiklerde de görüleceği gibi, birinci hoptaki düğüm sayısı arttıkça, ikinci hoptaki düğüm sayısı, aşırı denebilecek düzeyde,

çok hızlı artmaktadır. Örneğin, p=0,1 durumu için, birinci hopta 3 veya 4 düğüm varsa, ikinci hoptaki düğüm sayısı sırasıyla 41 ve 70 olmaktadır. Bu düğüm yoğunlukları, [Xue, 2004]'te ortaya konulan ve yaklaşık olarak 5 log(n) olan, en iyi bağlanırlık sayısını (connectivity number) ihlâl etmektedir.

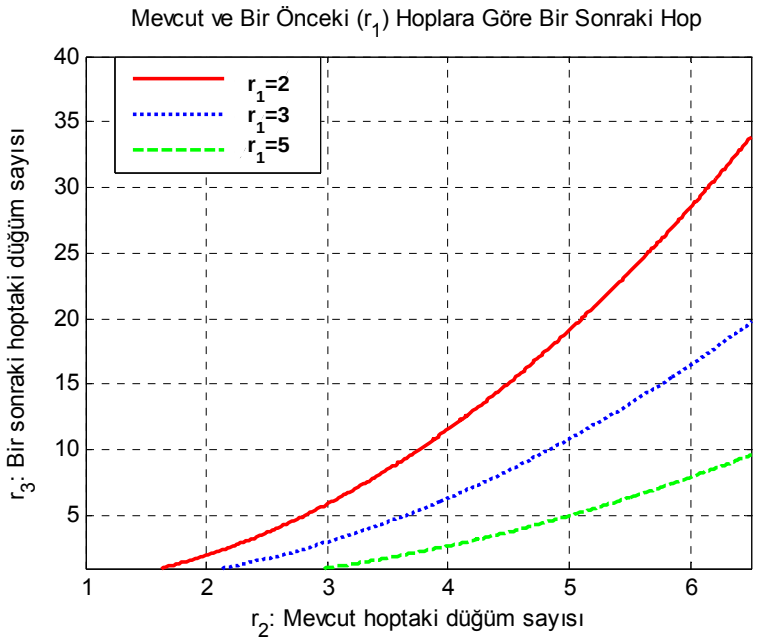


Şekil 6-14 İkinci hoptaki en uygun düğüm sayıları

Yukarıdaki analiz, bir veya daha fazla hop ilerletilirse, düğüm sayıları daha da artacaktır. Ancak, ikinci ve daha sonraki hoplar için Şekil 6-14'teki grafikler kullanılamaz. Çünkü, bu grafikler, birinci hopta bir gönderen var iken (sink) geçerli olan grafiklerdir. Oysa bir sonraki aşamada, bir değil birden çok gönderen mevcuttur. Birden çok gönderenin ve birden çok alıcının olması durumunda, bir sonraki hoptaki en uygun düğüm sayısının bulunması için, (6.27) eşitliği, aşağıdaki gibi değiştirilerek kullanılabilir.

$$r_2 E(r_3) = r_1 E(r_2) \tag{6.34}$$

Denklem üç bilinmeyenli olduğu için, çizilmesi ve çizilen grafiklerin anlaşılması zordur. Bu yüzden, değişik r_1 değerleri alınarak, bu değerler için ortaya çıkacak fonksiyonların çizilmesi daha faydalı olacaktır. Örneğin, birinci hopta 3 düğüm olduğu kabul edilerek (r_1=3), ikinci ve üçüncü hoptaki düğüm sayıları arasındaki ilişkiyi gösteren grafik çizilebilir.



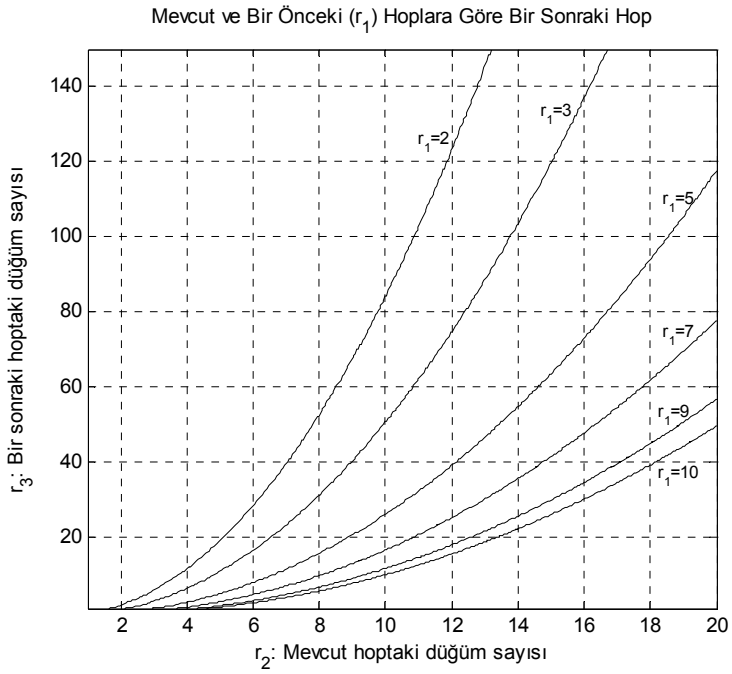
Şekil 6-15 Değişik r_1 değerleri için sonraki düğüm sayıları (Bölüm I)

Şekil 6-15 ve Şekil 6-16'da bu grafikler görülmektedir. Grafiklerde rahatlıkla görülebileceği gibi, hop sayısı arttıkça, en uygun düğüm sayısı üssel olarak artmaktadır. Çizelge 6-1'de, bu grafiklerden elde edilen, bazı en uygun düğüm sayıları görülmektedir.

Çizelge 6-1 Süperdüğümlerdeki en uygun düğüm sayıları

Sink	SD_1	SD_2	SD_3	SD_4
1	2	6	28	200
1	3	16	140	2000
1	4	30	360	6200
1	5	47	670	13000
1	6	70	1280	30000
1	7	90	1700	44000

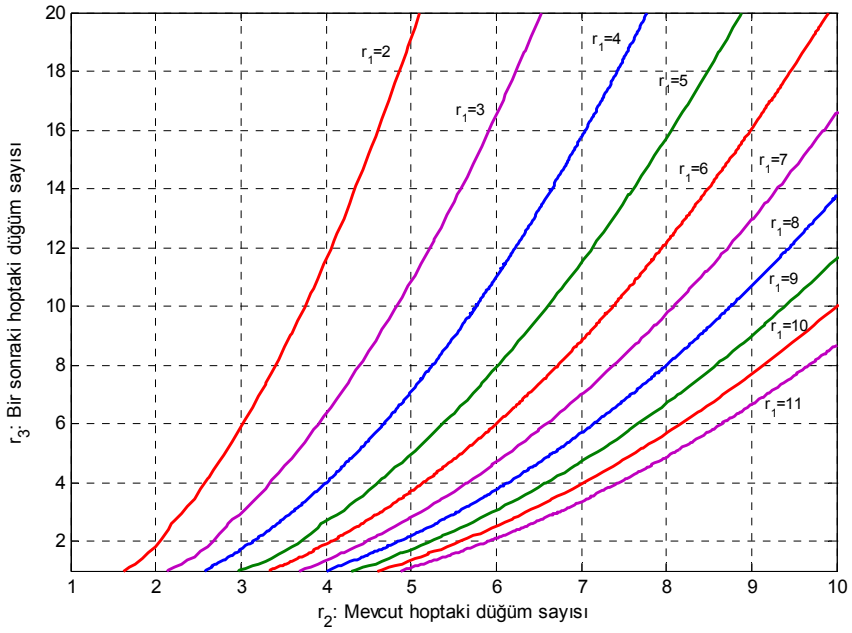
Düğüm yoğunluğu kriterlerine uymak için, düğüm sayılarında bir düzeltme yapılması gerekir. Sayıların bu kadar büyük çıkmasının en önemli sebebi, birinci hopta 1 gönderen olmasıdır. Analize, 1 gönderen ve r_1 alıcı ile başlandığı zaman, bir sonraki hoptaki düğüm sayıları, daha önce de belirtildiği gibi, çok büyük olmaktadır. Bunu engellemek için, analize ikinci hoptan başlanabilir. Yani, analizin ilk aşamasında r_1 gönderen ve r_2 alıcı olacak, bir sonraki aşamadaki düğüm sayısı olan r_3 ve sonraki aşamalardaki düğüm sayıları bulunmaya çalışılacaktır.



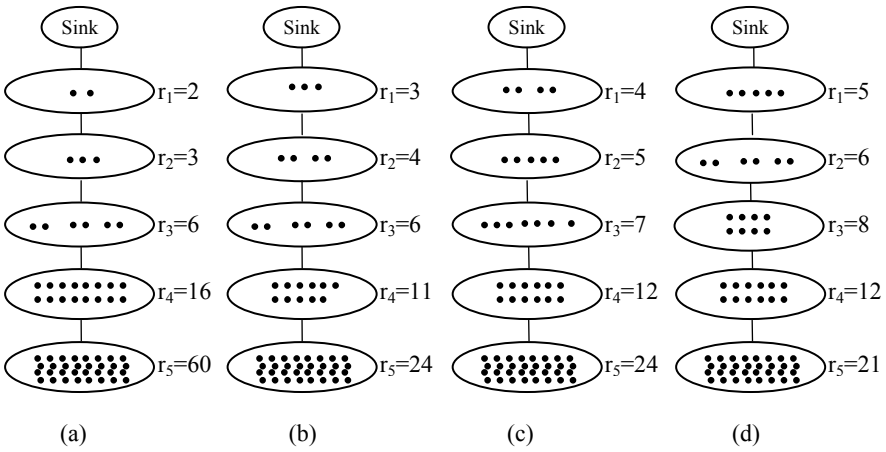
Şekil 6-16 Değişik r_1 değerleri için sonraki düğüm sayıları (Bölüm II)

Şekil 6-16'daki grafikler 2'den 11'e kadar bütün r_1 değerleri için çizilirse, Şekil 6-17'deki grafikler elde edilir. Bu grafikler kullanılarak, birinci ve ikinci hoptaki belirli düğüm sayıları için, diğer hoplardaki en uygun düğüm sayıları bulunabilir. Birinci ve ikinci hoptaki düğüm sayılarının ($\{r_1, r_2\}$) örnek olarak sırasıyla {2, 3},{3, 4},{4, 5} ve {5, 6} olması durumunda ortaya çıkacak ağlar, Şekil 6-18'de görülebilir. Şekilden de anlaşılabileceği gibi, iki hoptaki düğüm sayıları birbirine yakın ise, bir sonraki hoptaki düğüm sayısı da önceki hoplardakine yakın olmaktadır. Ancak, iki hopun düğüm sayıları arasındaki fark açıldıkça, bir sonraki hopun düğüm sayısı hızla büyümektedir.

Bu durum ortaya çıktığı zaman bir düzeltme yapılması kaçınılmazdır. Aksi halde, sonraki hoplarda, düğüm sayıları aşırı derecede büyümektedir. Düğüm yoğunluğu [Xue, 2004]'te ortaya konulan en iyi bağlanırlık sayısı olan 5 log(n)'i geçtiği zaman, düğüm sayıları buna uyacak şekilde düzeltilecektir.



Şekil 6-17 Değişik r_1 değerleri için sonraki düğüm sayıları (Bölüm III)

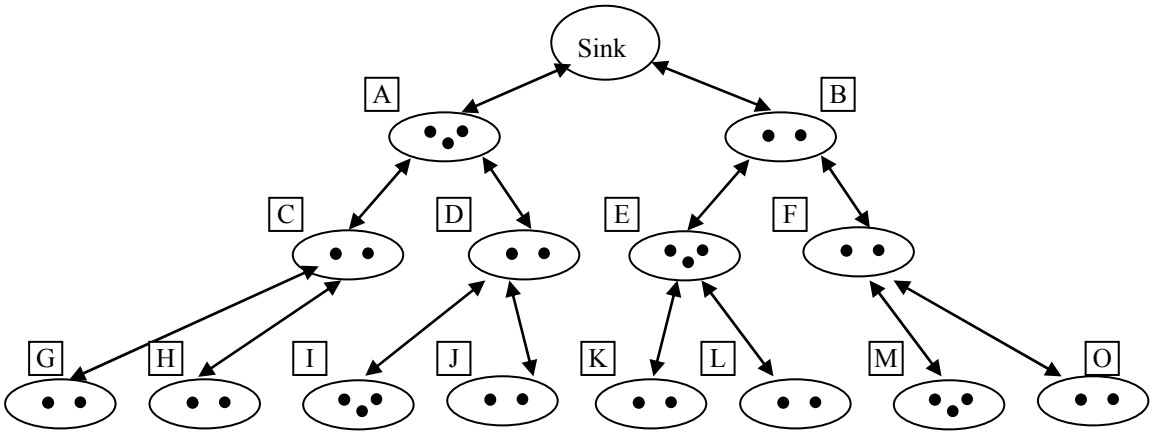


Şekil 6-18 Değişik başlangıç durumları için ortaya çıkan topolojiler

6.3.1 Süperdüğüm Varsayımının Gevşetilmesi

Sonraki hoplarda, düğüm sayılarının çok yüksek çıkmasının diğer önemli bir sebebi, topoloji konusunda yapılmış olan varsayımdan kaynaklanmaktadır. Bir süperdüğümdeki bütün düğümler, programı bir sonraki süperdüğümdeki düğümlerin tamamına gönderebilmektedir. Gerçek ortamlarda, bu tarz bir topolojinin ortaya çıkması çok düşük bir olasılıktır. Gerçek ortama biraz daha yaklaşılması için, süperdüğüm topoloji varsayımı bir adım gevşetilmiş ve analiz yeni bir topoloji üzerinden sürdürülmüştür.

Yeni topolojinin Şekil 6-19'daki gibi olduğu varsayılmıştır. Bir hoptaki bütün düğümlerin bir süperdüğüm oluşturması yerine, yarı yarıya iki süperdüğüme dağıldığı kabul edilmiştir. Süperdüğümler, metin içinde daha kolay atıfta bulunabilmek için, "A" harfinden başlanarak, alfabetik olarak isimlendirilmiştir. Her süperdüğüme bağlı bir sonraki hoptaki düğümler de, aynı şekilde yarı yarıya iki süperdüğüme dağılmıştır.



Şekil 6-19 Yeni ağ topolojisi

(6.3)'te verilmiş olan N_x açısından, birinci hopta, önceki topolojiden farklı bir şey olmayacaktır. Yine sink, önce N adet mesaj gönderecek ve birinci hopta bulunan A ve B süperdüğümlerindeki düğümlerden oluşan r tane alıcının her biri, bu mesajların Np tanesini kaybedecektir. Sonra, bu kaybedilen mesajları sink tekrar gönderecek ve bu işlem program başarıyla gönderilene kadar devam edecektir. N_x değeri (6.3)'te olduğu gibi, aşağıdaki gibi olacaktır:

$$N_x = N + rNp + rNp^2 + rNp^3 + \ldots = N\frac{1+(r-1)p}{1-p} \tag{6.35}$$

İkinci hopta ise, durum önceki analizden farklı olacaktır. Bu sefer, iki süperdüğüm, programı ayrı ayrı gönderecektir. Çünkü, A'nın gönderdiği mesajları, B'ye bağlı olan E ve F süperdüğümleri; B'nin gönderdiği mesajları ise, A'ya bağlı olan C ve D süperdüğümleri duymayacaktır. Önce, A ve B, N adet mesaj gönderecek; her birinin gönderdikleri mesajların Np tanesi alıcılar tarafından doğru alınamayıp tekrar istenecektir. İkinci hoptaki toplam düğüm sayısına r_2 denilirse, bunların $r_2/2$ tanesi A'nın altında, $r_2/2$ tanesi de B'nin altında olacaktır. Bu durumda N_x değeri aşağıdaki gibi olur:

$$N_x = N + \frac{r_2}{2}Np + \frac{r_2}{2}Np^2 + \frac{r_2}{2}Np^3 + ... = N\frac{1+(r_2/2-1)p}{1-p} \tag{6.36}$$

Bu hopun toplam gönderebileceği program sayısı ise, önceki analizden farklı olarak, bir düğümün toplam gönderebileceği program sayısının "r_1" katı değil, "$r_1/2$" katı olacaktır. Çünkü, A ve B süperdüğümleri, alıcılarına programı ayrı ayrı göndermektedir ve birinin alıcıları diğerini duymamaktadır.

Bir sonraki hopta ise, ayrı ayrı gönderen süperdüğüm sayısı 4'tür. Bunlar; C, D, E ve F süperdüğümleridir. Bunların altındaki r_3 adet alıcı, 4'e bölünmüş olarak dağıtılmıştır. Bu hopta gönderilebilecek toplam program sayısı aşağıdaki gibidir:

$$\frac{r_2}{4}E(\frac{r_3}{4}) \tag{6.37}$$

Bu durumda, en iyi topoloji analizi için kullanılacak eşitlik,

$$\frac{r_1}{2}E(\frac{r_2}{2}) = \frac{r_2}{4}E(\frac{r_3}{4}) = \frac{r_3}{8}E(\frac{r_4}{8}) = ... = \frac{r_h}{2^h}E(\frac{r_{h+1}}{2^h}) \tag{6.38}$$

şeklindedir. Burada "h", hop sayısını; r_h ise, h numaralı hoptaki düğüm sayısını temsil etmektedir. Birinci ve ikinci hop için geçerli olan eşitlik, N_x'in (6.36)'daki değeri yerine konulup tekrar yazılırsa aşağıdaki eşitlik elde edilir:

$$\frac{r_1}{2}\frac{k_{\max}}{N(\frac{pr_2}{1-p})}\ln(\frac{1+(r_2-1)p}{1-p}) = \frac{r_2}{4}\frac{k_{\max}}{\frac{N}{2}(\frac{pr_3}{1-p})}\ln(\frac{1+(\frac{r_3}{2}-1)p}{1-p}) \tag{6.39}$$

$$r_1 r_3 \ln(\frac{p}{1-p}r_2 + 1) = {r_2}^2 \ln(\frac{p/2}{1-p}r_3 + 1) \tag{6.40}$$

Bu eşitlik bir sonraki hop için,

$$r_2 r_4 \ln(\frac{p/2}{1-p}r_3 + 1) = {r_3}^2 \ln(\frac{p/4}{1-p}r_4 + 1) \tag{6.41}$$

şeklinde olmaktadır. Genel olarak h^{nci} hop için bu eşitlik aşağıdaki gibidir:

$$r_{h-1} r_{h+1} \ln(\frac{p/2^{h-2}}{1-p}r_h + 1) = {r_h}^2 \ln(\frac{p/2^{h-1}}{1-p}r_{h+1} + 1) \tag{6.42}$$

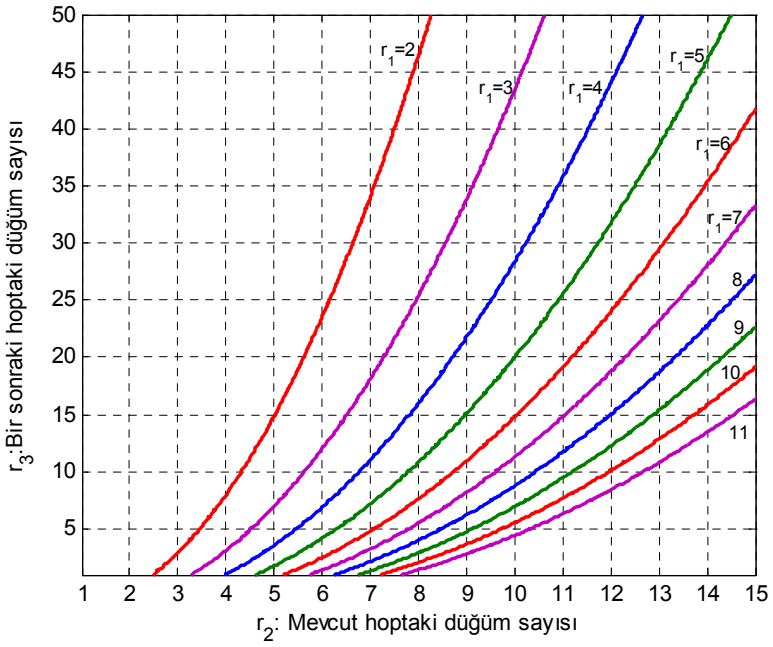
p=0,5 için aşağıdaki eşitlikler elde edilir:

$$r_1 r_3 \ln(r_2 + 1) = r_2^{\,2} \ln(\frac{1}{2} r_3 + 1) \tag{6.43}$$

$$r_2 r_4 \ln(\frac{1}{2} r_3 + 1) = r_3^{\,2} \ln(\frac{1}{4} r_4 + 1) \tag{6.44}$$

$$r_{h-1} r_{h+1} \ln(2^{h-2} r_h + 1) = r_h^{\,2} \ln(2^{h-1} r_{h+1} + 1) \tag{6.45}$$

Eşitlikler üç değişkenli olduğu için, grafikleri çizilirken daha önce uygulanan yöntem uygulanmıştır. Bir önceki hoptaki düğüm sayısı sabitlenip, diğer iki hop arasındaki eşitlik çizilmiş ve değişik değerler için bu işlem tekrarlanmıştır. Bu durumda, r_1'in değişik değerleri için (6.43) çizilirse, Şekil 6-20 elde edilir.



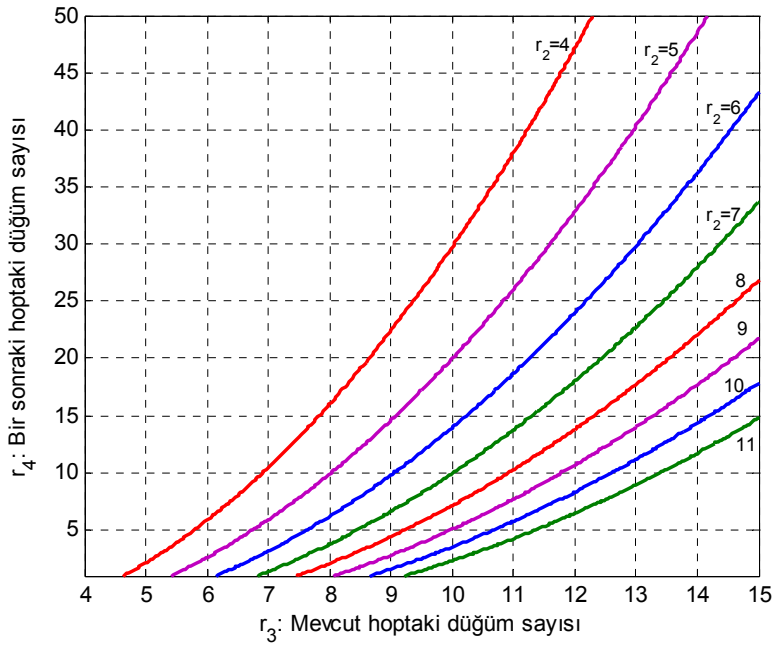
Şekil 6-20 Değişik r_1 değerleri için sonraki düğüm sayıları (yeni topoloji)

Bu grafiklerle, Şekil 6-17'deki, bir önceki topoloji için bu grafiklerin karşılığı olan grafikler karşılaştırılırsa, yeni topolojinin uygun düğüm sayılarına etkisi görülebilir. Bazı karşılaştırmalı düğüm sayıları Çizelge 6-2'de verilmiştir. Çizelgede, r_1 ve r_2'nin değişik değerleri için r_3'ün önce eski topolojideki, sonra yeni topolojideki değerleri verilmiştir. Çizelgede rahatlıkla görülebileceği gibi, üçüncü hoptaki düğüm sayısı, r_1 ve r_2'nin her durumu için, önceki topolojiye göre daha düşük çıkmıştır.

Çizelge 6-2 Karşılaştırmalı düğüm sayıları (r_3: eski|yeni)

	r_2=3		r_2=4		r_2=5		r_2=6		r_2=7		r_2=8	
r_1=2	6	3	12	8	19	15	27	23	40	35	52	46
r_1=3			6	3	11	7	16	12	23	18	30	25
r_1=4					7	3	11	7	16	11	21	16
r_1=5							8	4	12	7	16	11
r_1=6									9	5	12	8

Aynı grafikler bir sonraki hop için çizilince, Şekil 6-21 elde edilmiştir. Dikkatle incelenirse, bir önceki grafiklere çok yakın sonuçlar elde edildiği görülebilir. İki grafik arasındaki tek fark, karşılık gelen eşitlikler içindeki ln(…) fonksiyonları içindeki r_h ve r_{h+1} değişkenlerinin katsayılarının, ikinci grafiktekiler için ½ ile çarpılmış olmasıdır.



Şekil 6-21 Değişik r2 değerleri için sonraki düğüm sayıları (yeni topoloji)

Bu grafikler, sonraki hoplar için de çizilip düğüm sayıları elde edilmiş ve ortaya çıkan ağların Şekil 6-18'dekilere göre daha düşük düğüm yoğunluklarına sahip oldukları görülmüştür. Çizelge 6-3'te, bazı örnek düğüm sayıları verilmiştir. Analiz için örnek alınan topoloji, normal şartlara ne kadar yaklaşırsa, analiz o kadar karmaşıklaşacak; ama elde edilen sonuçlar da o kadar gerçeğe yakın olacaktır.

Çizelge 6-3 Yeni topolojideki en uygun düğüm sayıları

r_1	r_2	r_3	r_4	r_5
2	4	8	16	32
2	5	15	58	300
3	6	12	24	48
3	7	18	48	150
4	8	16	32	64
4	9	22	46	100

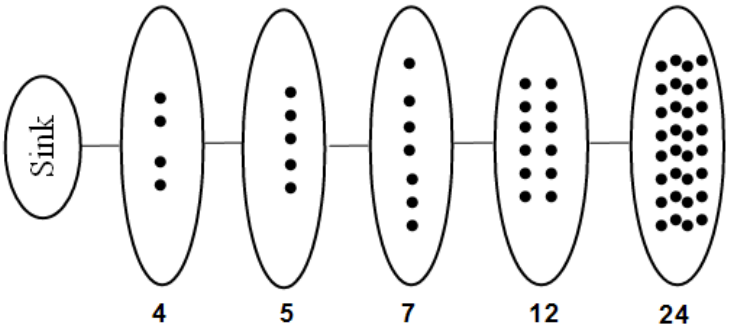
6.4 Düğüm Sayılarında Düzeltme

Tekrar Şekil 6-1'de verilen, önceki topoloji üzerinden yapılan analize dönülürse; Şekil 6-18'de, değişik başlangıç durumları için ortaya çıkan topolojiler bulunmuş ve düğüm sayıları aşırı artmaya başladığı zaman, düğüm yoğunluğu kriterlerine uymak için, bir düzeltme yapılması gerektiği ortaya konulmuştu. Topoloji oluşturulurken, öncelikle yapılması gereken, ilk iki hopta kaç düğüm olacağına karar verilmesidir. Düğüm yoğunluğu, n toplam düğüm sayısı olmak üzere, 5 log(n) civarında tutulmak istenmektedir.

Toplam düğüm sayısı örnek olarak 1000 alınırsa, 5 log(n) sayısı 15'e eşit olacaktır. Örnek alınan topolojide, ağdaki herhangi bir süperdüğümdeki düğümler; hem bulundukarı süperdüğümdeki düğümleri, hem de bir üst ve bir alt hoplarda bulunan süperdüğümlerdeki düğümleri duyabilmektedir. Bu nedenle, süperdüğümlerde olması gereken düğüm sayısı, düğüm yoğunluğunun üçte biri kadar olmalıdır. N=1000 için bu sayı 5'tir. Bu durumda, başlangıçta, birinci ve ikinci hopa sırasıyla 4 ve 5 düğüm yerleştirmek uygun olacaktır. Sonraki hoplarda analizin öngördüğü düğüm sayıları Şekil 6-18'den alınırsa, Şekil 6-22'de verilmiş olan ağ elde edilir. Üçüncü hopta düğüm sayısı 7 iken, dördüncü hopta 12, beşinci hopta ise 24'tür. Bu düğüm sayıları, düğüm yoğunluğu kriterlerine uymadığı için, düzeltme yapılması gerekir. Dördüncü hopa 12 düğüm yerine 4 düğüm, beşinci hopa da 24 düğüm yerine 5 düğüm yerleştirilirse sorun giderilmiş olur. Bu durumda, dört hopta bir düzeltme yapılması gerektiği görülür.

Aynı düzeltme analizi, Şekil 6-19'da kabul edilen yeni ağ topolojisi için yapılırsa, ortaya daha farklı bir sonuç çıkar. Çünkü bu topolojide, ağdaki düğümler, bulundukları hoptaki veya komşu hoplardaki bütün düğümleri duymamaktadır. Örneğin, "C" süperdüğümündeki

düğümler, bulundukları hoptaki diğer süperdüğümler olan “D”, “E” ve “F” süperdüğümlerini duymamaktadırlar. Aynı şekilde bu düğümler, bir sonraki hopta bulunan “G”den “O”ya sekiz adet süperdüğümden, sadece ikisini duyabilmektedir. Bu düğümlerin bağlanırlık sayısı “A”, “C”, “G” ve “H” süperdüğümlerindeki düğümlerin sayısına eşit olacaktır.



Şekil 6-22 Düzeltme gerektiren düğüm sayıları

“A”daki düğümlerin sayısı, bulunduğu hoptaki, yani 1. hoptaki düğümlerin ½’sine, “C”deki düğümlerin sayısı 2. hoptaki düğümlerin ¼’üne ve “G ve H”deki düğümlerin sayısı da 3. hoptaki düğümlerin ¼’üne eşittir. Yani bu topolojide, h^{nci} hoptaki bir düğümün bağlanırlık sayısı “b_h”,

$$b_h = \frac{r_{h-1}}{2^{h-1}} + \frac{r_h}{2^h} + \frac{r_{h+1}}{2^h} \tag{6.46}$$

olmaktadır. Çizelge 6-3’te verilen düğüm sayılarına bakılırsa, 1000 düğümlük bir ağda, düğüm sayılarında nerdeyse hiçbir düzeltme yapılmasına gerek olmadığı görülebilir. Bağlanırlık sayınının çizelgedeki pek çok durum için 5 log(n)’den, yani 15’ten düşük olduğu, çok az durum için bu değerin üstüne çıktığı görülebilir. Örneğin, çizelgenin 4. satırındaki, { r_3=18, r_4=48, r_5=150} değerleri dikkate alınırsa, 4. hop için b_4 değeri,

$$b_4 = \frac{18}{8} + \frac{48}{16} + \frac{150}{16} \cong 16 \tag{6.47}$$

olacaktır. Aynı şekilde, çizelgenin 2. satırındaki, { r_3=15, r_4=58, r_5=300} değerleri kullanılırsa 4. hop için b_4 değeri,

$$b_4 = \frac{15}{8} + \frac{58}{16} + \frac{300}{16} \cong 24 \tag{6.48}$$

olacaktır. Bu son durumda bir düzeltme yapılması gerektiği açıkça görülmektedir. Çizelge dikkatle incelendiği zaman, örnek olarak ele alınan bu iki durumda da, 2. hoptaki düğüm sayılarının 1. hoptakinin iki katından büyük olduğu görülebilir. İkinci hoptaki düğüm sayısı, birinci hoptaki düğüm sayısının iki katı civarında olursa, ortaya çıkacak düğüm sayılarında, bağlanırlık açısından bir düzeltme yapmaya gerek kalmamaktadır.

SONUÇ VE ÖNERİLER

Geleneksel bilgisayar sistemlerinden özellikle kaynakların çok kısıtlı olması bağlamında ayrışan duyarga ağları, pek çok yeni araştırma konusuna kaynaklık etmektedir. Şimdiye kadar bilgisayar sistemlerinde ve ağlarında geliştirilmiş olan pek çok teknoloji ve çözüm, duyarga ağlarının ihtiyaçlarına cevap verememekte, duyarga ağlarına uygun tasarımlar gerektirmektedir.

Sistemin programlanması, duyarga ağlarını geleneksel sistemlerden ayıran önemli konulardan biridir. Yerleştirildikten sonra, fiziksel olarak erişilmesi çoğu durumda verimsiz ve hatta imkânsız olan duyarga ağlarının, uzaktan programlanması gerekmektedir. Bu ise, kaynakları zaten çok kısıtlı olan duyarga ağlarına ek bir yük getirmektedir. Bu nedenle, uzaktan programlama işleminin kaynak kullanımı açısından en verimli şekilde yapılması çok önemlidir.

Bu tezin konusu, telsiz duyarga ağlarının uzaktan programlanması ile ilgilidir. Önce, duyarga ağları üzerinde kullanılan işletim sistemleri incelenmiş (bunların arasında, TinyOS işletim sistemi, duyarga ağları için standart işletim sistemi olma yolunda hızla ilerlemektedir) daha sonra duyarga ağlarının uzaktan programlanması konusunda yapılmış olan sınırlı sayıda çalışma üzerinde durulmuştur. İncelenen bu çalışmaların hepsinde, programlama işleminin "sürekli" (continuous) olduğu gözlenmiş ve bunun sisteme gereksiz bir yük getirdiği saptanmıştır. Güncellenmiş bir ağda bile, güncel olmayan düğümlerin var olup olmadığının belirlenmesi için, sürekli bir mesaj gönderimi yoluyla, hem enerji hem de bant genişliği israfına neden olunmaktadır.

Tezin literatüre olan temel katkıları aşağıdaki gibidir:

- Sürekli olmayan bir kod yayma mekanizması,
- Kod yayma öncesi en uygun ağ altyapısının belirlenmesi,
- Düğümler arasında bir sorumluluk ağacı kavramının geliştirilmesi.

Bu çalışmada, duyarga ağlarının uzaktan programlanması için, özellikle enerji ve bant genişliği açısından, varolan algoritmalardan daha verimli bir algoritma geliştirilmiştir. Varolan algoritmaların enerji açısından verimli olmamasının en önemli sebebi, algoritmaların sürekli çalışıyor olmasıdır. Bu çalışmada ise, kod yayımı öncesi düğümler arasında oluşturulan bir sorumluluk ağacı aracılığıyla, bu "süreklilik" ortadan kaldırılmaktadır. Ağaçtaki her düğüm, kendi yavrularının güncellenmesinden sorumlu olmaktadır. Ağa yeni bir program yüklendiği zaman, düğümlerin hepsi, yavrularının yeni programı aldıklarını tespit

ettiklerinde, mesaj gönderme işlemi durmaktadır.

Oluşturulan ağaç, enerji yükü, düğümlere dengeli olarak dağılacak şekilde tasarlanmıştır. Ayrıca, ağaç yapısı dinamiktir ve sürekli güncellenebilmektedir. Bir düğümün enerjisi tükenince, bu düğüme bağlı yavru düğümlerin başka bir düğüme bağlanması ve bu yavruların güncellenmesi sorumluluğunun, bu yeni düğüme aktarılması gerekir. Ortaya konulan algoritmanın sonuçları, simülasyon ortamında gözlenmiş ve varolan algoritmalara göre önemli iyileştirmeler sağladığı gözlenmiştir.

Kod yayma işleminin ve genel olarak telsiz duyarga ağlarında çalışan bütün uygulamaların, mümkün olduğunca verimli çalışabilmesi için, uygulamaya en uygun ağ altyapısının mevcut olması gerekir. Ağdaki en uygun hop sayısı, her hopta bulunacak düğüm sayısı gibi parametrelerin titizlikle belirlenmesi, kaynakların verimli kullanılabilmesi açısından çok önemlidir. Ağın ömrü, ağ altyapısı ile yakından ilgilidir.

Bu çalışmada ayrıca, en iyi ağ altyapısı konusunda matematiksel bir analiz geliştirilmiştir. Önce basit bir topoloji ile, sonra daha karmaşık topolojilerle geliştirilen analiz sayesinde, kod yayma uygulaması için en uygun ağ altyapısı belirlenmekte ve elde edilen bu bilgiler kod yayma algoritmasında kullanılmaktadır.

Analizden elde edilen bu bilgiler ve Bölüm 5.5'te yapılan iyileştirme çalışmalarından elde edilen sonuçlar kullanılarak geliştirilmiş olan kod yayma algoritmasının, daha karmaşık ağ topolojilerinde, simülasyon ve gerçek ortamlarda performansının ölçülmesi yapılacak işler arasındadır. İleriye yönelik diğer çalışmalar ise, duyarga düğümlerinin hareket halinde olabildikleri hareketli duyarga ağlarında ağaç oluşturma, ağaç bakımı ve uzaktan programlama algoritmalarının geliştirilmesidir.

KAYNAKLAR

Abrach H., Carlson J, Dai H., Rose J., Sheth A., Shucker B. ve R. Han R., (2003), "MANTIS: System Support for MultimodAL NETworks of In-situ Sensors", In Proceedings of 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), Eylül 2003, San Diego, CA.

Akyildiz I.F., Su W., Sankarasubramaniam Y. ve Cayirci E., (2002), "Wireless sensor networks: a survey", Computer Networks, 38(4):393{422}.

An L., Young-Hyun O. ve Peng N., (2008), "Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks Using Deluge", "International Conference on Information Processing in Sensor Networks, 2008. (IPSN '08).

Boulis A., Han C.-C. ve Srivastava M. B., (2003), "Design and Implementation of a Framework for Efficient and Programmable Sensor Setworks", In Proceedings of the First International Conference on Mobile Systems, Applications, and Services, ACM Pres, 187-200.

Cerpa A., Elson J., Hamilton M., Zhao J. (2001), "Habitat monitoring: application driver for wireless communications technology", ACM SIGCOMM'2000, Costa Rica.

Crossbow (web sitesi), http://www.xbow.com/Products/productdetails.aspx?sid=156.

Çimen Ç. (web sitesi), "Yaklaşan Teknoloji: Sensör Ağlar", http://temagem.sdu.edu.tr/index.php?dosya=b2&tur=2.

Deng J., Han R. ve Mishra, S., (2006), "Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks", Proceedings of The 5th International Conference On Information Processing in Sensor Networks (IPSN '06).

Dunkels A., Grönvall B. ve Voigt T., (2004), "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", IEEE Emnets.

Dutta P. K., Hui J. W., Chu D. C. ve Culler D. E. (2006), "Securing the Deluge Network Programming System", Proceedings of the Fifth International Conference on Information Processing in Sensor Networks, 19-21 Nisan 2006, Nashville, Tennessee, ABD.

eCos, The eCos Operating System, http://ecos.sourceware.org.

Estrin D., Girod L., Pottie G., Srivastava M.(2001), "Instrumenting the world with wireless sensor networks", International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001), Salt Lake City, Utah.

Fok C., Roman G.-C. ve Lu C., (2004), "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications", Tech. Rep. WUCSE-04-59, Washington University, Department of Computer Science and Engineering, St. Louis.

Gay D., Levis P., Behren R., Welsh M., Brewer E. ve Culler D., (2003), "The nesC Language: A Holistic Approach to Networked Embedded Systems", Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, 1–11.

Han C.-C., Kumar R., Shea R., Kohler E. ve Srivastava M, (2005), "A Dynamic Operating System for Sensor Nodes, Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, 06-08 Haziran, 2005, Seattle, Washington.

He T., Stankovic T.A., Lu C., Abdelzaher T., (2002), “SPEED: A Real-Time Routing Protocol For Sensor Networks”, Technical Report CS-2002-09, University of Virginia.

Hedetniemi S., Liestman A., (1988), “A survey of gossiping and broadcasting in communication networks”, Networks 18 (4) 319–349.

Hill J., Szewczyk R., Woo A., Hollar S., Culler D. E. ve Pister K., (2000), “System Architecture Directions for Networked Sensors”, In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, 93-104, 2000, Cambridge, MA.

Hui J. W. ve Culler D., (2004) “The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale”, The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04).

Jeong J., Culler D. (2004) “Incremental Network Programming for Wireless Sensors”, Secon 04: Proceedings of the The First IEEE Communications Society Conference.

Karl H. ve Willig A., (2006), Protocols and Architectures for Wireless Sensor Networks, John Wiley & Sons, New York.

Karp B., Kung H. T. GPSR (2000) “GPSR: Greedy Perimeter Stateless Routing for Wireless Networks”, In Proceedings of International Conference on Mobile Computing and Networking (Mobicom), Boston, Massachusetts, USA.

Köpke A., Handziski V., Hauer J. H. Ve Karl H., (2004), “Structuring the Information Flow in Component-Based Protocol Implementations for Wireless Sensor Nodes”, In Proceedings of the Work-in-Progress Session of the 1st European Workshop on Wireless Sensor Networks (EWSN), Ocak 2004, Berlin, Almanya.

Kulkarni S., Wang L. (2005), “MNP: Multihop Network Programming for Sensor Networks”, International Conference on Distributed Computing Systems, Columbus OH.

Lanigan P. E., Gandhi R. ve Narasimhan P., (2005), “Secure Dissemination of Code Updates in Sensor Networks”, Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, 02-04 Kasım 2005, San Diego, California, ABD.

Levis P., Culler D., (2002), “Maté: a tiny virtual machine for sensor networks”, ACM SIGPLAN Notices, v.37 n.10.

Levis P., Lee N., Welsh M. ve Culler D., (2003), " Tossim: Accurate and Scalable Simulation of Entire Tinyos Applications.", Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003), Kasım 2003, Los Angeles, ABD.

Levis P., (2003), “Viral Code Propagation in Wireless Sensor Networks”, University of California, Berkeley.

Levis P., Patel N., Shenker S. ve Culler D., (2004), "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks", Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation .

Li S. F., Sutton R. Ve Rabaey J., (2001), “Low Power Operating System for Heterogeneous Wireless Communication Systems”, In Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques (PACT 01), Eylül 2001, Barselona, İspanya.

Liu T., Martonosi M, (2003), “Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems”, In PPoPP ’03: Proceedings of the ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, New York, NY, USA.

Lorin H ve Deitel H. M., (1981), Operating Systems, Reading, Massachusetts:Addison-Wesley Publishing Company, Inc.

Mutluoglu M. B., Ergin M. O. ve Baydere S., (2007), “TAP: Tree-Based Code Propagation in Wireless Sensor Networks”, World Congress in Computer Science, Computer Engineering, and Applied Computing - WORLDCOMP’07, 25-28 Haziran 2007, Las Vegas, ABD.

Perkins C. E. Royer E. M. (1999) “Ad Hoc On-Demand Distance Vector Routing”, In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), New Orleans, Louisiana, USA.

Reijers N., Langendoen K., (2003), “Efficient Code Distribution in Wireless Sensor Networks”, WSNA 03: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications, NewYork,NY,USA.

Stann F. Heidemann J. (2003) “RMST: Reliable Data Transport in Sensor Networks”, In Proceedings of The First International Workshop on Sensor Net Protocols and Applications (SNPA'03), Anchorage, AK, USA.

Stathopoulos T., Heidemann J., Estrin D., (2003), “A Remote Code Update Mechanism for Wireless Sensor Networks”, Technical report, UCLA, Los Angeles, CA, USA.

Tanenbaum A. S. ve Woodhull A. S., (1997), Operating Systems: Design and Implementation, Prentice Hall.

Tekin U., (2006), “Kablosuz Duyarga Ağlarında Etkili Yönlendirme ve Enerji Problemleri”, Seminer raporu, GYTE Bilgisayar Mühendisliği Bölümü, Gebze, Kocaeli.

Tian D., Georganas N. D., (2003), “Low-Cost, Reliable Data Delivery in Large Wireless Sensor Networks”, Technical Report, University of Ottawa.

Xue F. ve Kumar P. R., (2004), “The Number of Neighbors Needed for Connectivity of Wireless Networks", Wireless Networks, 10(2): 169-181.

ÖZGEÇMİŞ

Doğum Tarihi	17.10.1973	
Doğum Yeri	Trabzon	
Lise	1983-1990	Trabzon Anadolu Lisesi
Lisans	1990-1994	Bilkent Üniversitesi Mühendislik Fakültesi Elektrik-Elektronik Mühendisliği Bölümü
Yüksek lisans	1994-1999	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Bilimleri ve Mühendisliği Bölümü
Doktora	2001-2008	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Bölümü

Çalıştığı kurum

1996-Devam ediyor	Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü Donanım Anabilim Dalı Araştırma Görevlisi