EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

TELSİZ DUYARGA AĞLARI İÇİN HAKİM KÜME ALGORİTMALARI

Özkan ARAPOĞLU

Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN

Uluslararası Bilgisayar Anabilim Dalı

Bilim Dalı Kodu: 619.02.04
Sunuş Tarihi: 05.01.2015

Bornova-İZMİR

2015

Özkan ARAPOĞLU tarafından YÜKSEK LİSANS tezi olarak sunulan "Telsiz Duyarga Ağları İçin Hakim Küme Algoritmaları" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 05.01.2015 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: Doç. Dr. Orhan DAĞDEVİREN**	**.....**
Raportör Üye	**: Prof. Dr. Mehmet Emin DALKILIÇ**	**.....**
Üye	**: Yrd. Doç. Dr. Hasan BULUT**	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "Telsiz Duyarga Ağları İçin Hakim Küme Algoritmaları" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

06 / 01 / 2015

İmzası

Adı-Soyadı

Özkan ARAPOĞLU

ÖZET

TELSİZ DUYARGA AĞLARI İÇİN HAKİM KÜME ALGORİTMALARI

ARAPOĞLU, Özkan

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN
Ocak 2015, 80 sayfa

Telsiz duyarga ağları (TDAlar) algılama yapabilen ve telsiz haberleşebilen düğümlerin oluşturdukları altyapısız bir ağdır. TDA üzerinde çalışan uygulamaların paketlerinin çıkış düğümüne doğru iletilmesi önemli bir problemdir. Ağı kümelemek bu problemin çözümlerindendir. Hakim küme oluşturma, hata toleranslı kümeleme yöntemlerinden bir tanesidir. Hakim kümenin içindeki düğümlerin hiçbiri birbirinin komşusu değilse bu küme bağımsız küme olur. Bağımsız küme oluşturma, küme liderlerini belirlemek için kullanılan önemli bir yöntemdir.

Öz kararlılık bir düğümün sadece komşularının durumlarına göre karar vermesini sağlar. Öz kararlılık dağıtık ve hata toleranslı çalışma için çok uygundur. Bu yöntemde düğümlerin iyi tanımlanmış kuralları çalıştırması gereklidir. Bu tezde öz kararlı dağıtık maksimal bağımsız küme algoritmaları üzerine çalışılmıştır. Literatürdeki algoritmaların teorik ve pratik değerlendirmesi yapılmış, literatürdeki algoritmalara göre daha etkin olduğu kanıtlanmış bir algoritma tasarlanmıştır. Önerilen algoritma teorik olarak analiz edilmiştir, benzetim ortamında gerçeklenmiştir ve Iris düğümler üzerinde test edilmiştir.

Anahtar sözcükler: Telsiz Duyarga Ağları, Bağımsız Küme, Hakim Küme, Öz Kararlılık, Dağıtık Algoritma

ABSTRACT

DOMINATING SET ALGORITHMS FOR WIRELESS SENSOR NETWORKS

ARAPOĞLU, Özkan

MSc in Department of International Computer
Supervisor: Assoc. Prof. Dr. Orhan DAĞDEVİREN
January 2015, 80 pages

Wireless sensor networks (WSNs) are infrastructureless network of nodes which are capable of sensing and wireless communication. Relaying the packets of applications running on WSN is an important problem. Clustering the network is a solution for this problem. Dominating set construction is a fault tolerant clustering method. If any of two dominators are not neighbors of each other then the set of dominators is called independent set. Construction of an independent set is an important method to elect cluster leaders.

Self-stabilization provides a decision making mechanism for a node about its state by just checking its neighbors state. Self-stabilization is a very suitable method for distributed and fault tolerant processing. In this technique, nodes should execute well defined rules. In this thesis, distributed self-stabilizing maximal independent set algorithms are studied. Theoretical and practical evalutions of the previous work are made, an algorithm which is shown to be effective than the previous work, is designed. Proposed algorithm is analyzed theoretically, implemented in the simulation environment and tested on Iris nodes.

Keywords: Wireless Sensor Networks, Independent Set, Dominating Set, Self-Stabilization, Distributed Algorithm

TEŞEKKÜR

Yüksek Lisans tezimde benden bilgisini, deneyimlerini ve desteğini esirgemeyen, bana yol gösteren, tez danışmanım Sn. Doç. Dr. Orhan DAĞDEVİREN'e çok teşekkür ederim.

TOSSIM simülatörünü öğrenmeme büyük katkısı olan, benden desteğini ve bilgilerini esirgemeyen Sn. Vahid Khalilpour Akram'a çok teşekkür ederim.

Yüksek Lisans eğitimim boyunca bilgilerini ve desteğini esirgemeyen Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü'ndeki tüm saygıdeğer hocalarıma çok teşekkür ederim.

Hayatımın her anında yanımda olan sevgili eşim Tuğba ARAPOĞLU'na sonsuz teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ÇİZELGELER DİZİNİ

Çizelge Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
id_i	i düğümünün kimlik numarası
s_i	i düğümünün durum bilgisini tutan yerel değişken
$\forall j \in N_i$	i düğümünün bütün komşuları için
$\exists j \in N_i$	i düğümünün komşularından en az biri için
d	Durum sayısı
SİYAH	Maksimal bağımsız kümeye dahil olan düğümün s_i değeri (Siyah renkli düğümle gösterilir)
BEYAZ	Maksimal bağımsız kümeye dahil olmayan düğümün s_i değeri (Beyaz renkli düğümle gösterilir)
BEKLE	Maksimal bağımsız kümede olup olmadığına henüz karar vermemiş düğümün s_i değeri (Gri renkli düğümle gösterilir)

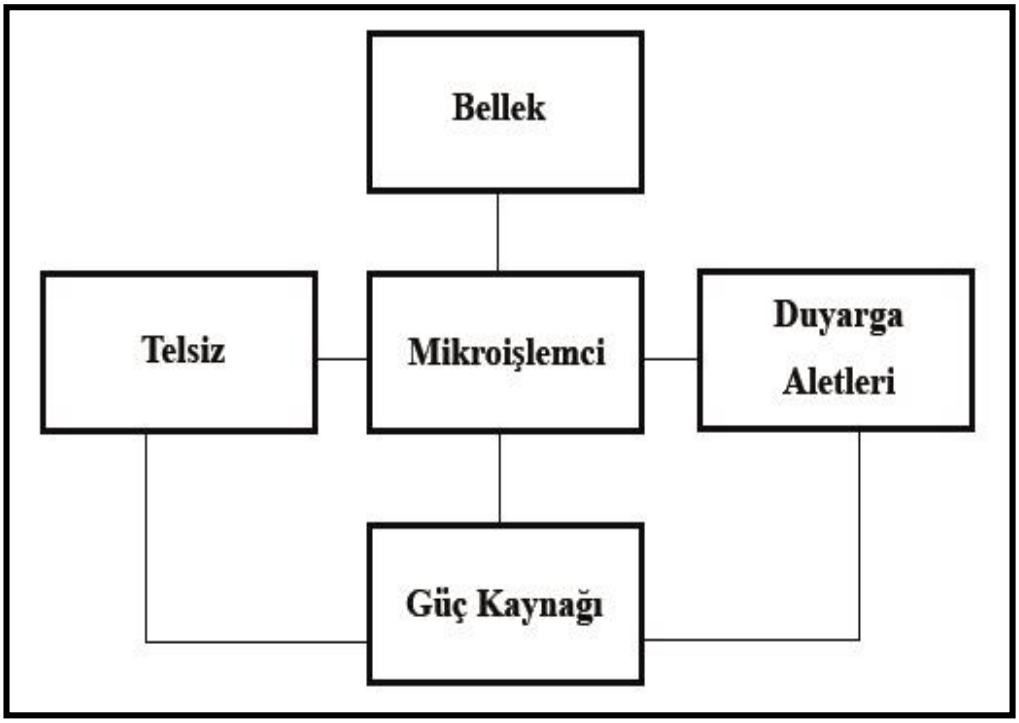
Kısaltmalar

TDA	Telsiz Duyarga Ağı
MBK	Maksimal Bağımsız Küme

1. GİRİŞ

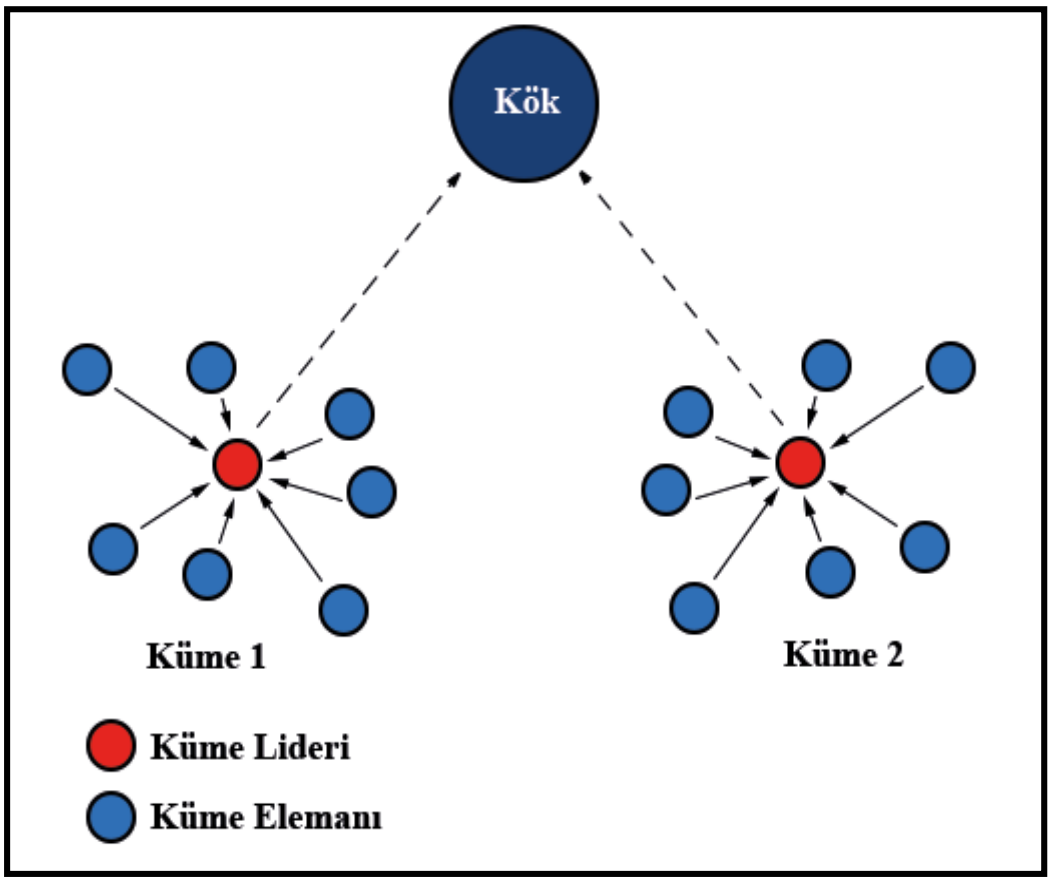
Telsiz Duyarga Ağı (TDA), fiziksel ortam ile dijital dünya arasında iletişim sağlayabilen ve fiziksel ortamdan topladığı verileri işleyebilen birden fazla duyarga (Ing. *Sensor*) düğümünün (Ing. *Node*) sabit bir altyapı olmadan bir araya gelmesiyle oluşan ağdır. Duyarga ağlarının, ortamdaki sıcaklık, basınç, nem, hareket, ses ve ışık gibi değerleri ölçebilen duyarga düğümlerine sahip olması, farklı uygulama alanlarında kullanılabilmelerini sağlar (Li et al., 2008). TDA, askeriyede sınır güvenliğinin sağlanması, tarımda üretimin artırılabilmesi için toprağın nem, su, sıcaklık değerlerinin ölçülmesi, akıllı ev sistemlerinde bazı işlerin otomatik olarak yapılması gibi birçok alanda kullanılmaktadır. TDA'nın kullanım alanının geniş olması, bu alandaki çalışmalar üzerindeki ilgiyi artırmıştır ve TDA'nın popüler olmasını sağlamıştır (Sohraby et al., 2007). TDA günümüzde de popülerliğini korumaktadır.

TDA içindeki bir duyarga düğümü, mikroişlemci, telsiz (Ing. *Transceiver*), bellek, güç kaynağı ve duyarga aletlerinden oluşur. Bir duyarga düğümünün yapısı Şekil 1.1'de yer almaktadır. Duyarga düğümü, fiziksel ortamdaki verileri, olayları algılayabilir, analiz edebilir ve verileri komşu düğümlere gönderebilir. Bir kök düğüm (Ing. *Sink*) ile ağ izlenebilir, ağdaki veriler elde edilebilir ve çalıştırılan algoritmaya göre istenilen bilgilere ulaşılabilir. Veri iletimi kablosuz ortamda radyo dalgaları ile gerçekleştirilir. Bir duyarga düğümünün, komşu bir düğüme veri gönderebilmesi için komşu düğümün kendi kapsama alanının (Ing. *Transmission Range*) içine girmesi gerekir. İzlenen bir alanın birden fazla duyarga düğümünün kapsama alanına girmesi ağın enerji verimliliğini (Ing. *Energy Efficiency*) azaltır ve maliyetini artırır. Duyarga düğümleri en fazla enerjiyi veri gönderirken veya alırken harcar. Duyargalar enerji tasarrufu için uyku haline geçirilebilir. Uyku durumunda iken duyarga düğümü veri paketi alamaz veya gönderemez. Duyarga uyku durumuna geçerken ya da uyandırılırken ek enerji harcar. Bir düğümün ne zaman uyku konumuna getirileceği, ne kadar uykuda kalacağı ve ne zaman uyandırılması gerektiği iyi düşünülmelidir. Bir duyarga düğümü kendi kendini yönetebilir. Bu özellikleri sayesinde ağdaki bir duyarga düğümünde hata oluştuğunda, diğer düğümler hatayı belirli ölçüde telafi edebilir. Bir ağın bazı düğümlerinde hata olduğu halde sistemin düzgün çalışmaya devam edebilmesine "Hata Toleransı (Ing. *Fault Tolerance*)" denir. Kısıtlı bellek alanına, işlem gücüne ve enerjiye sahip olan bir duyarga düğümü üzerinde çalışan algoritmanın yüksek enerji verimliliği ve hata toleransı sağlaması, ağın yaşam süresini uzatır.



Şekil 1.1 Bir duyarga düğümünün yapısı

Çevreden toplanan verilerin enerji verimliliğini sağlayacak şekilde çıkış düğümüne gönderilmesini sağlayacak etkin bir ağ topolojisi kurmak TDA'daki en önemli problemlerden biridir. Etkin bir ağ topolojisi oluşturmanın önemli yöntemlerinden birisi de kümeleme (Ing. *Clustering*) işlemidir (Abbasi et al., 2007). Kümeleme, bir ağı bağlı alt kümelere bölme ve bölünen her alt kümeye bir küme lideri (Ing. *Cluster Head*) atama işlemidir. Küme lideri kendi kümesi içindeki düğümlerden bilgileri toplar ve bilgiyi doğrudan kök düğüme ya da komşu alt kümenin liderine gönderir. Şekil 1.2'de örnek bir kümeleme şeması yer almaktadır. Kümeleme yöntemi ile TDA'da veri birleştirme (Ing. *Data Aggregation*), yönlendirme (Ing. *Routing*), yük dengesinin sağlanması (Ing. *Load Balancing*), zaman senkronizasyonu (Ing. *Time Synchronization*), topoloji kontrolü (Ing. *Topology Control*), hata toleransı, güvenli iletişimin sağlanması ve ağın yaşam süresinin uzatılması işlemleri daha etkin ve kolay şekilde yapılır. TDA'da etkin bir ağ topolojisi kurmanın diğer yolu ise sanal omurga (Ing. *Virtual Backbone*) oluşturmaktır. Sanal omurgaya sahip olan bir ağın hata toleransı ve yönlendirme esnekliği yüksektir. Kümeleme ve sağlam omurga oluşturma işlemleri birbirinden bağımsız değildir. Sağlam bir omurga oluşturabilmek için kümeleme problemini çözen etkin bir algoritmaya ihtiyaç vardır.

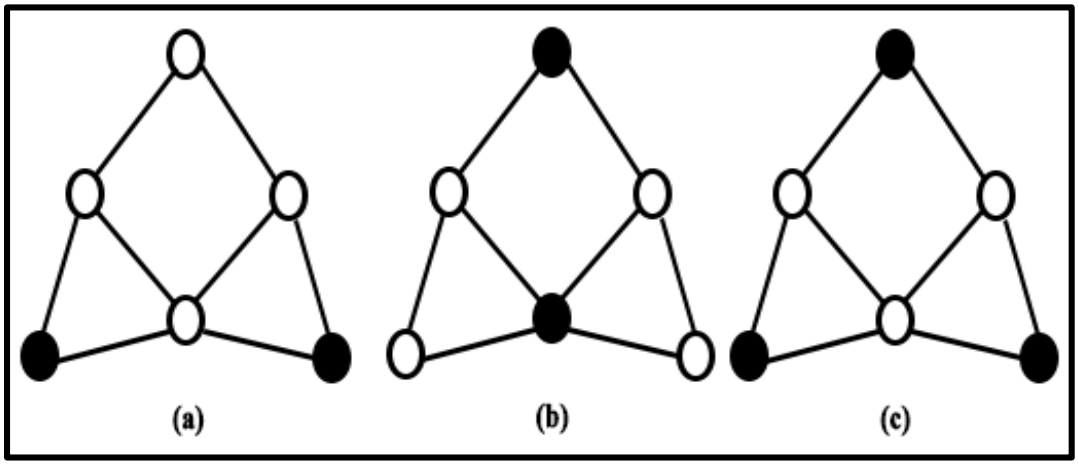


Şekil 1.2 Örnek bir kümeleme şeması

Bir *G* çizgesi, *V* düğümlerin kümesi, *E* kenarların kümesi olmak üzere *G*=(*V*,*E*) ikilisi olarak ifade edilebilir. TDA, çizge yapıları kullanılarak modellenebilmektedir. *V* kümesi içindeki bütün düğümler eğer *V* kümesinin alt kümesi olan ve boş küme olmayan bir *V'* kümesinin elemanı ise veya *V'* kümesinin içinde bulunan herhangi başka bir düğüme komşu ise *V'* kümesi hakim kümedir. *V'* kümesi içindeki düğümler kendi aralarında bağlı ise *V'* kümesi bağlı hakim kümedir (Wu et al., 2007).

Maksimal bağımsız küme (Ing. *Maximal Independent Set*), hakim küme yapılarının örneklerinden bir tanesidir. *V* düğümlerine ve *E* kenarlarına sahip olan bir *G* çizgesinin, herhangi iki düğümü arasında hiçbir bağlantı kenarı bulunmayan *V'* alt kümesine "Bağımsız Küme (Ing. *Independent Set*)" denir. *G* çizgesinin bağımsız kümelerinden herhangi birinin alt kümesi olmayan bağımsız kümeye ise "Maksimal Bağımsız Küme (MBK)" denir. MBK'nın boyutu içerdiği düğüm sayısı kadardır ve $\boldsymbol{\alpha(G)}$ simgesi ile gösterilir. G çizgesinin boyutu en fazla olan maksimal bağımsız kümesine, maksimum bağımsız küme denir. Maksimum bağımsız kümeyi bulmak "NP-Zor" problemdir.

Şekil 1.3a'da örnek bir bağımsız kümeyi, Şekil 1.3b'de örnek bir maksimal bağımsız kümeyi, Şekil 1.3c'de ise örnek bir maksimum bağımsız kümeyi görebiliriz. Siyah renkli düğümler bağımsız küme içerisindeki düğümleri göstermektedir. MBK, telsiz duyarga ağlarında kümeleme problemini çözmek, yeni ağ yapıları inşa etmek, gezgin (Ing. *Mobility*) ağlarda değişen ağ yapısına göre sistemin yeniden kurulmasını sağlamak, enerji verimliliğini ve hata toleransını artırarak ağın yaşam ömrünü uzatmak gibi birçok problemin çözümünde kullanılır (Erciyes, 2013).



Şekil 1.3 a) Bağımsız küme b) Maksimal bağımsız küme c) Maksimum bağımsız küme

Bir sistemin başlangıç durumuna (Ing. *State*) bakılmaksızın sınırlı adımlar sonunda kararlı duruma geçmesine ve dışsal bir müdahale olmadığı sürece kararlı durumda kalmasına "Öz Kararlılık (Ing. *Self Stabilization*)" denir. (Dijkstra, 1974) Öz kararlı algoritmalar bir ağdaki hata toleransını artırmayı hedefler (Schneider, 1993; Dolev, 2000). Gezgin ağlarda ağın öz kararlı olması sistemin düzgün çalışmasını sağlar. Duyarga düğümleri kendi kendini yönetebilen cihazlar olduğu için özellikle TDA'daki dağıtık sistemlerde öz kararlı algoritmalar geliştirmek büyük önem taşımaktadır.

TDA'da telsiz en fazla enerji harcayan birimdir. Bu yüzden ağdaki gereksiz mesaj gönderimlerinin azaltılması ağın yaşam süresini artırmaktadır. Öz kararlı algoritmalarda bir düğümün kuralının tetiklenmesi sonucunda düğümün durumunu değiştirmesi işlemine hareket (Ing. *Move*) denir. Her bir durum değişikliğinden sonra düğümün komşularının haberdar olması gerekir. Ağdaki bir düğümün durumunun değişmesi komşu düğümlerin durumunun değişmesine

neden olabilir. Bu açıdan sistemdeki hareket sayısı ne kadar azsa sistemin enerji verimliliği ve buna bağlı olarak ağın yaşam süresi o kadar artar.

Bu tezde, TDA'da ağın yaşam ömrünü uzatan yeni dağıtık öz kararlı maksimal bağımsız küme algoritması tasarlandı. Önerilen algoritmanın ("OZKAN") literatürdeki dağıtık öz kararlı MBK algoritlarından daha etkin çalıştığı gösterilmiştir. Bölüm 2'de literatürdeki maksimal bağımsız küme algoritmaları incelendi. Merkezi sistemde çalışan ilk öz kararlı MBK algoritması olan Shukla ve arkadaşlarının (Shukla et al., 1995) algoritması ("SHUKLA"), dağıtık sistemde çalışan ilk dağıtık öz kararlı MBK algoritması olan Ikeda ve arkadaşlarının (Ikeda et al., 2002) algoritması ("IKEDA") ve dağıtık öz kararlı MBK algoritması olan Turau'nun (Turau, 2007) algoritması ("TURAU") ayrıntıları ile anlatıldı. Bölüm 3'de OZKAN'ın genel fikri ve birim disk çizge (Ing. *Unit Disk Graph*) üzerinde örnek çalışması detayları ile anlatıldı. Bölüm 4'de OZKAN'ın teorik analizi, uygulamaların değerlendirmesi ve benzetimlerin değerlendirmesi yapıldı. Algoritmalar TOSSIM simülatöründe ve TinyOS işletim sistemini kullanan Iris düğümler üzerinde gerçek uygulama ile test edildi. Test sonuçları grafikler ile ayrıntılı anlatıldı. Bölüm 5'de ise sonuç bölümü yer almaktadır.

2. İLGİLİ ÇALIŞMALAR

Gradinariu ve arkadaşı (Gradinariu and Tixeuil, 2000), dağıtık sistemde tekil (Ing. *Unique*) süreç *id* değerlerine sahip olan ağlar için geliştirdiği düğüm renklendirme (Ing. *Vertex Coloring*) algoritması ile MBK'nın elde edilebileceğini göstermiştir. MBK'nın bulunabilmesi için karşılıklı dışlama (Ing. *Mutual Exclusion*) tekniğini (Beaquier et al., 2002) ve rasgelelik (Ing. *Randomization*) tekniğini (Turau and Weyer., 2006) kullanan algoritmalar geliştirildi. Wu ve arkadaşları (Wu et al., 2005), MBK'yı bulmak için birim disk çizgeler üzerinde çalışmıştır. Goddard ve arkadaşları (Goddard et al. 2003), senkron MBK algoritması tasarladı. Guellati ve arkadaşı (Guellati and Kheddouci, 2010), yaptıkları araştırmada öz kararlı bağımsız küme algoritmalarını incelemişlerdir ve algoritmalar çalıştırıldığı ortam (dağıtık – merkezi), kullanılan topoloji, karmaşıklık ve anonimlik yönünden karşılaştırılmıştır. Öz kararlı algoritmalar genellikle verilen kurallara göre çalışır. (Hedetniemi et al., 2003)

Shukla ve arkadaşları (Shukla et al., 1995) tarafından ilk öz kararlı MBK algoritması tasarlandı. Bu algoritma merkezi sistemde (Ing. *Central System*), rasgele topolojide, yönsüz, bağlı çizgede ve anonim ağda (Ing. *Anonymous Network*) çalışmaktadır. Algoritmanın amacı ağdaki MBK'yı bulmaktır. Ağdaki her bir düğümün s_i değeri ya SİYAH ya da BEYAZ durum değerlerini alabilir. Bütün düğümler iki kurala göre kendi durum değerine karar verir. Bir kuralın çalışması için ön koşullarının sağlanması gerekir. Bir kuralın çalışması ve düğümün s_i değerinin değişmesi işlemine bir adım (Ing. *Step*) denir. SHUKLA'nın kuralları Şekil 2.1'de gösterilmiştir.

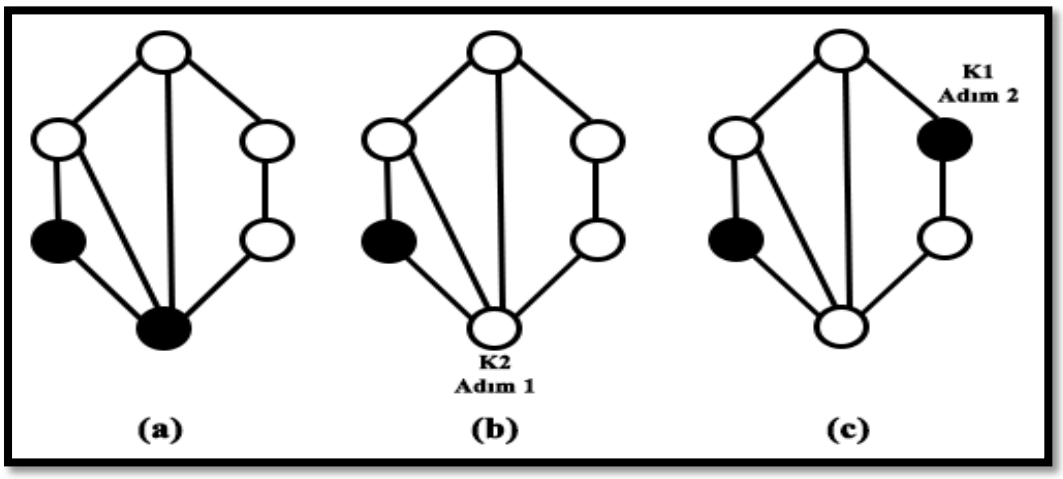
Kural 1: Eğer s_i = BEYAZ ve $\forall j \in N_i$ [s_j = BEYAZ] ise s_i = SİYAH olur.

Kural 2: Eğer s_i = SİYAH ve $\exists j \in N_i$[s_j = SİYAH] ise s_i = BEYAZ olur.

Şekil 2.1 SHUKLA'nın kuralları

SHUKLA'nın çalışmasının bir örneği Şekil 2.2'de gösterilmiştir. Şekil 2.2a'da G çizgesindeki düğümlerin başlangıç durumları verilmiştir. Algoritma merkezi sistemde çalıştığı için aynı anda sadece bir düğüm durumunu güncelleyebilir. Şekil 2.2b'de siyah düğümlerden bir tanesi Kural 2'yi çalıştırır ve

s_i değeri BEYAZ olur. Şekil 2.2c'de ise beyaz düğümlerden bir tanesi Kural 1'i çalıştırır ve s_i değeri SİYAH olur. MBK siyah düğümlerden oluşur ve sistem öz kararlı kalır. SHUKLA doğrusal, yönsüz ve bağlı çizge üzerinde çalıştırıldığında en kötü "n/2" adımda sistem öz kararlı olur (Shukla et al., 1995). Zaman karmaşıklığı, -n adım sayısı olmak üzere - O(n)'dir.



Şekil 2.2 SHUKLA'nın çalışma örneği

İlk öz kararlı dağıtık MBK algoritması Ikeda ve arkadaşları (Ikeda et al., 2002) tarafından yazılmıştır. Bu algoritma dağıtık sistemde (Ing. *Distributed System*), anonim ağda, yönsüz ve bağlı çizge üzerinde çalışmaktadır. Algoritma paylaşımlı bellek modelini kullanmaktadır. Bir düğüm komşularının durumlarını okuyabilir fakat; sadece kendi durum değerini güncelleyebilir. IKEDA'nın iki kuralı vardır. Bir kuralın ön koşullarının sağlanması ile her bir düğüm bir süreç başlatır. Her bir sürecin kromatik (Ing. *Chromatic*) tekil *id* değeri vardır. Turau (Turau et al., 2007), IKEDA'nın ana fikrinin bir düğümün Kural 1'i çalıştırdıktan sonra aynı tur içinde Kural 2'yi de çalıştırması olduğunu belirtmiştir. IKEDA'nın kuralları Şekil 2.3'te gösterilmiştir.

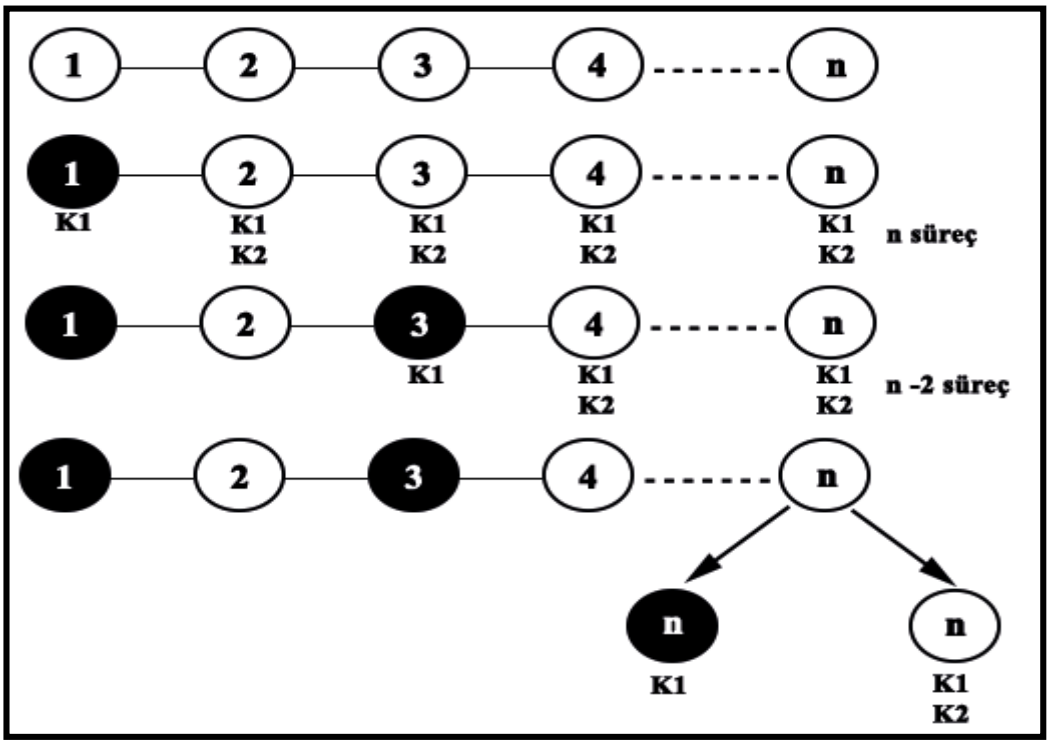
Kural 1: Eğer s_i = BEYAZ ve $\forall j \in N_i$ [s_j = BEYAZ] ise s_i = SİYAH olur.

Kural 2: Eğer s_i = SİYAH ve $\exists j \in N_i$ [s_j = SİYAH ve $id_j < id_i$] ise

s_i = BEYAZ olur.

Şekil 2.3 IKEDA'nın kuralları

IKEDA'nın çalışmasının bir örneği Şekil 2.4'te yer almaktadır. Başlangıçta bütün düğümlerin s_i değeri BEYAZ'dır. Topoloji, doğrusal, yönsüz ve bağlı çizgedir. Kural 1'i çalıştıran bağlı bileşenlerin (Ing. *Component*) süreç *id* değerleri kromatiktir ve artan ardışık sayılardan oluşmaktadır. Çizgede n adet düğüm vardır ve n değeri 1'den büyüktür. Birinci adımda bütün düğümler Kural 1'in ön koşullarını sağladığı için s_i değerini SİYAH yapar ve en büyük bağlı bileşeni oluşturur. Topolojideki en az bir düğüm, en küçük süreç *id* değerine sahip olduğu için bu düğüm Kural 2'yi çalıştıramaz ve öz kararlı olur. Diğer düğümler aynı tur (Ing. *Round*) içinde sırası ile Kural 2'yi çalıştırır ve s_i değerini tekrar BEYAZ yapar. Öz kararlı olan düğümün en az bir komşusu Kural 1'i tekrar çalıştıramayacağı için bir turda en az iki düğüm öz kararlı olur. İkinci adımda n-2 düğüm Kural 1'i çalıştırır ve aynı tur içinde n-3 tanesi tekrar Kural 2'i çalıştırır. Bu işlemler bütün sistem öz kararlı hale gelene kadar döngüsel devam eder. Ikeda algoritması, en kötü durumda Şekil 2.4'teki çizge üzerinde çalışır ve -n süreç sayısı olmak üzere- en fazla "(n+1)(n+2)/4" süreçte sistem öz kararlı olur. Bu yüzden zaman karmaşıklığı $O(n^2)$'dir. IKEDA'nın zaman karmaşıklığı formülü (Ikeda et al., 2002) Şekil 2.5'te gösterilmiştir .



Şekil 2.4 IKEDA'nın çalışma örneği

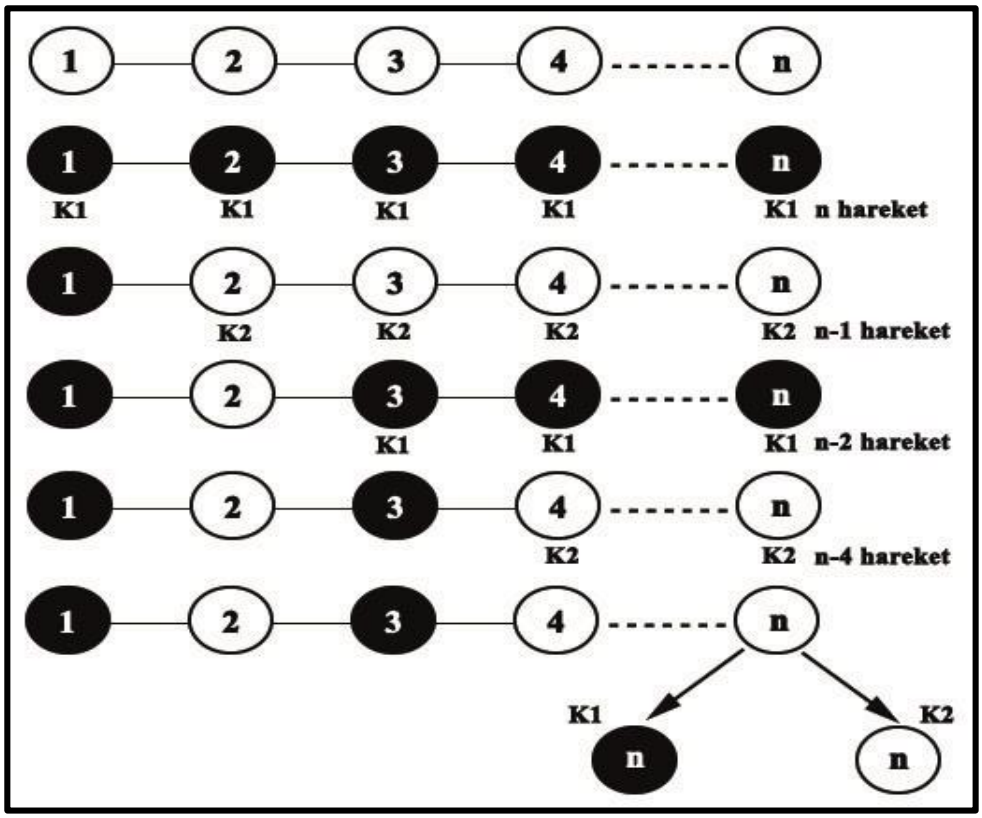
$$
\begin{aligned}
&(1+n-1)+(1+n-3)+(1+n-5)+\cdots \\
&\quad = (n)+(n-2)+(n-4)+\cdots \\
&\quad = \tfrac{1}{2}\{\,2(n)+2(n-2)+2(n-4)+\cdots\} \\
&\quad < \tfrac{1}{2}\{\,(n+1)+(n)+(n-1)+(n-2)+ \\
&\quad = \tfrac{1}{2}\sum_{i=1}^{n+1} i \\
&\quad = \tfrac{1}{4}(n+2)(n+1)
\end{aligned}
$$

Şekil 2.5 IKEDA'nın zaman karmaşıklığı formülü

TDA'da öz kararlı algoritmalar geliştirmenin kısıtları vardır (Lagemann et al., 2009). Eğer IKEDA TDA'da tasarlansaydı, n hareket sayısını göstermek üzere "n(n+1)/2" harekette sistem öz kararlı hale gelirdi. Paylaşımlı bellek modelinde öz kararlı olan düğüm durum bilgisini paylaşımlı belleğe yazmaktadır. Bu yüzden öz kararlı düğüm komşusuna durum bilgisini göndermek için bir hareket yapmaz. TDA'da düğümler arasında paylaşımlı bellek olmadığı için düğümler her kuralı çalıştırdığında yeni durum bilgisini komşularına göndermek zorundadır. Ayrıca düğümler birbirlerinin süreç *id*'lerini göremeyeceği için sistem ancak tekil *id* değerine sahip düğümlerden oluşabilir. Her düğüm mesaj paketi ile kendi *id* ve s_i değerlerini komşu düğümlere yayın (Ing. *Broadcast*) yaparak göndermek zorundadır. Her mesajda *id* ve s_i değerleri gönderildiği için bir mesajın boyutu 2 bayt'tır. Bu yüzden paylaşımlı bellek modelinde IKEDA'nın alan karmaşıklığı (Ing. *Space Complexity*) 1 bit iken, TDA modelinde 2 bayt'tır.

IKEDA'nın TDA modelinde çalışma örneği Şekil 2.6'da yer almaktadır. Üzerinde çalışılan topoloji doğrusal, yönsüz, bağlı ve tekil *id* değerleri ardışık artan çizgeden oluşmaktadır. Çizgede düğüm sayısı 1'den büyük olmak üzere n adet düğüm vardır. Başlangıçta bütün düğümlerin s_i değeri BEYAZ'dır. Bütün düğümler birbirlerine "Merhaba (Ing. *Hello*)" mesaj paketi ile *id* ve s_i değerlerini gönderir. Bütün düğümlerde Kural 1'in ön koşulları sağlandığı için her düğüm Kural 1'i çalıştırır ve s_i değerlerini SİYAH yapar. Durum bilgisi değişen her düğüm komşularına yeni s_i değerini ve tekil *id* değerini yayın yaparak gönderir. İkinci turda en küçük *id* değerine sahip en az bir düğüm (id_1) Kural 2'yi çalıştıramayacağı için öz kararlı olur. Bu durumda n-1 düğüm Kural 2'yi çalıştırır ve en az bir düğüm (id_2), id_1'den dolayı Kural 1'i çalıştıramayacağı için öz kararlı olur. Her adımda hareket sayısı bir azalarak sistem "n(n+1)/2" harekette öz kararlı

hale gelir. Zaman karmaşıklığı –n hareket sayısı olmak üzere- $O(n^2)$'dir. IKEDA'nın TDA modelinin zaman karmaşıklığı formülü Şekil 2.7'de gösterilmiştir.



Şekil 2.6 IKEDA'nın TDA modelinde çalışma örneği

$$T(n) = n + (n-1) + (n-2) + \ldots\ldots + 1$$
$$= \sum_{i=1}^{n} i$$
$$= n(n+1)/2$$

Şekil 2.7 IKEDA'nın TDA modelinin zaman karmaşıklığı formülü

TURUA, TDA'da çalışmaktadır. Sistem dağıtıktır ve sistemdeki düğümlerin her biri tekil *id* değerine sahiptir. Herhangi bir düğümün s_i değeri değiştiğinde komşu düğümleri bilgilendirmek için yayın yapmaktadır. TURAU'nun dört kuralı vardır. Bir kuralın ön koşulları sağlandığında s_i değerini değiştirmesi işlemine bir hareket denir. Sistem dağıtık olduğu için birden fazla düğüm aynı anda s_i değerini değiştirebilir. Algoritmanın üzerinde çalıştığı çizge yönsüz ve bağlıdır. Turau

(Turau, 2007), sınırlı kaynağa sahip kablosuz sistemlerde hareket sayısının azaltılmasının en az tur sayısı kadar önemli olduğunu belirtmiştir. Çünkü, s_i değeri değişen her düğüm yayın yapmaktadır. TDA'da iletişim en fazla enerji tüketen unsur olduğu için, hareket sayısının azaltılması ağın yaşam süresini uzatır. TURAU'nun kuralları Şekil 2.8'de gösterilmiştir.

Kural 1: Eğer s_i = BEYAZ ve $\exists j \in N_i$ [$s_j \neq$ SİYAH] ise s_i = BEKLE olur.

Kural 2: Eğer s_i = BEKLE ve $\exists j \in N_i$ [s_j = SİYAH] ise s_i = BEYAZ olur.

Kural 3: Eğer s_i = BEKLE ve $\exists j \in N_i$ [$s_j \neq$ SİYAH] ve

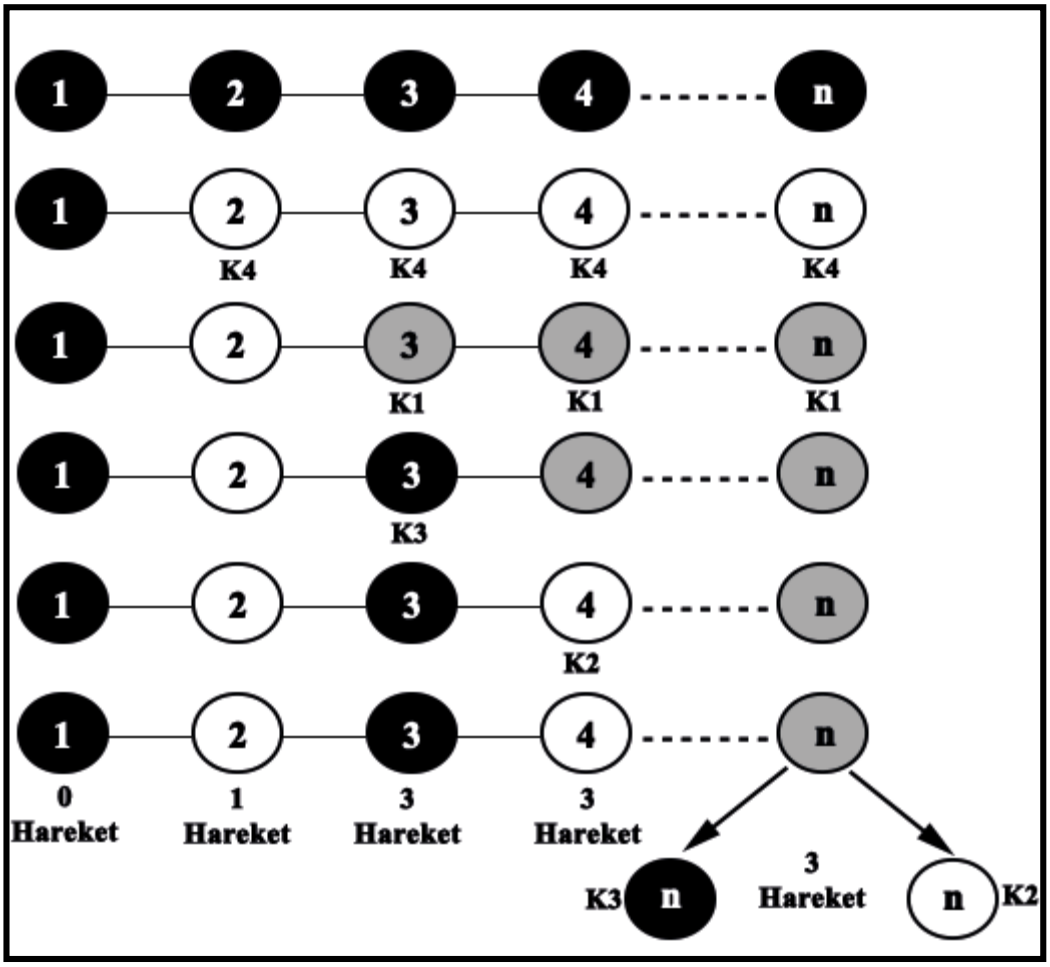
$\forall j \in N_i$[s_j = BEKLE ve $id_i < id_j$] ise s_i = SİYAH olur.

Kural 4: Eğer s_i = SİYAH ve $\exists j \in N_i$ [s_j = SİYAH ve $id_j < id_i$] ise

s_i = BEYAZ olur.

Şekil 2.8 TURAU'nun kuralları

TURAU'nun çalışmasının bir örneği Şekil 2.9'da yer almaktadır. Başlangıçta bütün düğümlerin s_i değeri SİYAH'tır. Topoloji, doğrusal, yönsüz, bağlı ve tekil *id* değerleri ardışık artan çizgeden oluşmaktadır. Bütün düğümler birbirlerine "Merhaba" mesaj paketi ile *id* ve s_i değerlerini gönderir. Sistemdeki en az bir düğümün *id* değeri en küçüktür. Bu durumda id_1 dışındaki bütün düğümler Kural 4'ü çalıştırır ve s_i değerini BEYAZ yapar. Birinci turun sonunda id_1 öz kararlı olur. Durum bilgisi değişen diğer her bir düğüm komşularına yeni s_i değerini ve tekil *id* değerini gönderir. Öz kararlı olan düğümün (id_1) en az bir komşusu (id_2) Kural 4'ü çalıştırdıktan sonra öz kararlı olur. Dolayısı ile id_1 hiç hareket yapmadan; id_2 sadece bir hareket yaparak öz kararlı olmuştur. İkinci turda 3'den n'ye kadar olan düğümler Kural 1'i çalıştırır ve BEKLE durumuna geçer. Üçüncü turda BEKLE durumundaki id_3, Kural 3'ü çalıştırır, SİYAH durumuna geçer ve öz kararlı olur. Dördüncü turda BEKLE durumundaki id_4, Kural 2'yi çalıştırır, BEYAZ durumuna geçer ve öz kararlı olur. İşlemler bütün sistem öz kararlı olana kadar döngüsel devam eder. Sistem, en fazla -n hareket sayısı ve 5'den büyük olmak üzere- "3n-5" hareket sayısı ile öz kararlı hale gelir (Turau, 2007). Bu yüzden, zaman karmaşıklığı O(n)'dir.



Şekil 2.9 TURAU'nun çalışma örneği

3. ÖNERİLEN ALGORİTMA

3.1 Genel Fikir

OZKAN, yönsüz çizge veya rasgele birim disk çizgelesiyle modellenebilen dağıtık sistemlerde, öz kararlı maksimal bağımsız küme oluşturmak için tasarlanmıştır. Bu açıdan TDA için çok uygundur. Ağdaki bütün düğümler tekil *id* değerine sahiptir. Her düğümün var olan yerel değişkeni s_i, üç farklı durum değeri alabilir. Bu durum değerleri: SİYAH, BEYAZ ve BEKLE'dir. Eğer bir düğüm MBK kümesine dahil ise s_i değeri SİYAH'tır, MBK kümesine dahil değilse BEYAZ'dır, MBK kümesine dahil olup olmadığına henüz karar vermemişse BEKLE'dir. Sistemdeki düğümler başlangıçta üç durumdan herhangi birine sahip olabilir. Sistem öz kararlı hale geldiğinde sistemde sadece MBK kümesinde yer alan SİYAH ve MBK kümesinde yer almayan BEYAZ durumlarına sahip düğümler yer alır. Ayrıca, öz kararlı hale gelen sistemde her BEYAZ düğümün mutlaka bir SİYAH komşusu vardır. Dışsal bir müdahale olmadığı sürece sistem öz kararlı kalmaya devam eder.

Başlangıçta bütün düğümler birbirlerine "Merhaba" mesaj paketi ile *id* ve s_i değerlerini gönderir. Her düğüm komşusundan "Merhaba" mesaj paketini aldıktan sonra, herhangi bir kuralın ön koşulları sağlanmışsa s_i değerini güncelleyebilir. Sistem dağıtık olduğu için aynı anda birden fazla düğüm s_i değerini değiştirebilir. Herhangi bir düğümün s_i değeri değiştiğinde komşu düğümleri bilgilendirmek için yayın yapmaktadır. Sistemde sadece s_i değeri değişen düğümler mesaj göndereceği için durumu değişmeyen yani öz kararlı olan düğümler gereksiz yere mesaj paketi göndermemiş olur. TDA'da telsiz en fazla enerji harcayan birim olduğu için ağdaki gereksiz mesaj gönderimlerinin azaltılması ağın yaşam süresini artırmaktadır. OZKAN'ın kuralları Şekil 3.1'de gösterilmiştir. Şekil 3.2'de ise OZKAN'ın kurallarının sonlu durum makinası (Ing. *FSM*) ile gösterimi yer alır.

OZKAN'ın en fazla hareket sayısı ile öz kararlı hale geldiği çalışma örneği Şekil 3.3'de yer almaktadır. Topoloji, doğrusal, yönsüz, bağlı ve ardışık artan tekil *id* değerine sahip çizgeden oluşmaktadır. Sistemde n sayısı 1'den büyük olmak üzere n adet düğüm vardır. Siyah düğümler SİYAH durum değerini, beyaz düğümler BEYAZ durum değerini ve gri düğümler BEKLE durum değerini ifade etmektedir. Başlangıçta bütün düğümlerin durum değerleri BEYAZ'dır. Sistemde en az bir düğümün *id* değeri en küçüktür (id_1). Bu düğüm Kural 1'in ön koşullarını sağladığı için bir hareket yaparak s_i değerini SİYAH yapar. Diğer her

düğüm kendinden küçük *id* değerine sahip en az bir komşu düğüme sahip olduğu için Kural 2'yi çalıştır ve s_i değerini BEKLE yapar. Her düğüm bir hareket yaptığı için sistemde n tane hareket yapılmış olur. Kural 1'i çalıştıran düğüm (id_1) başka hiç bir kuralı çalıştıramayacağı için öz kararlı hale gelir. Bu düğümün en az bir komşusu olacağı için komşusu olan düğüm (id_2), Kural 3'ü çalıştırır ve s_i değerini BEYAZ yapar. BEKLE durumundaki düğümlerden en az bir tanesi en küçük *id* değerine sahiptir (id_3). Bu düğüm Kural 3'ü çalıştırmadığına göre SİYAH komşusu yoktur. Bu durumda BEKLE durumundaki en küçük *id* değerine sahip olan düğüm (id_3), Kural 4'ü çalıştırır ve s_i değerini SİYAH yapar. Sistemdeki BEKLE durumundaki bütün düğümler döngüsel olarak ya Kural 3'ü çalıştırır BEYAZ durumuna geçer ya da Kural 4'ü çalıştırır SİYAH durumuna geçer. Sistem öz kararlı hale geldiğinde sistemde sadece SİYAH ve BEYAZ durumda düğümler vardır. Her BEYAZ düğümün en az bir tane SİYAH komşusu vardır. Böylece öz kararlı sistemde MBK oluşur. Sistemde id_1 bir hareket ile, diğer düğümler ortalama iki hareket ile öz kararlı olur. Bu yüzden sistem, en fazla –n hareket sayısı olmak üzere- "2n-1" hareket yaparak öz kararlı hale gelir.

Kural 1: Eğer s_i = BEYAZ ve $\forall j \in N_i$ [s_j = BEYAZ] ve

$\forall j \in N_i$ [s_j = BEYAZ ve $id_i < id_j$] ise s_i = SİYAH olur.

Kural 2: Eğer s_i = BEYAZ ve $\forall j \in N_i$ [s_j = BEYAZ] ve

$\exists j \in N_i$ için [s_j = BEYAZ ve $id_j < id_i$] ise s_i = BEKLE olur.

Kural 3: Eğer s_i = BEKLE ve $\exists j \in N_i$ [s_j = SİYAH] ise s_i = BEYAZ olur.

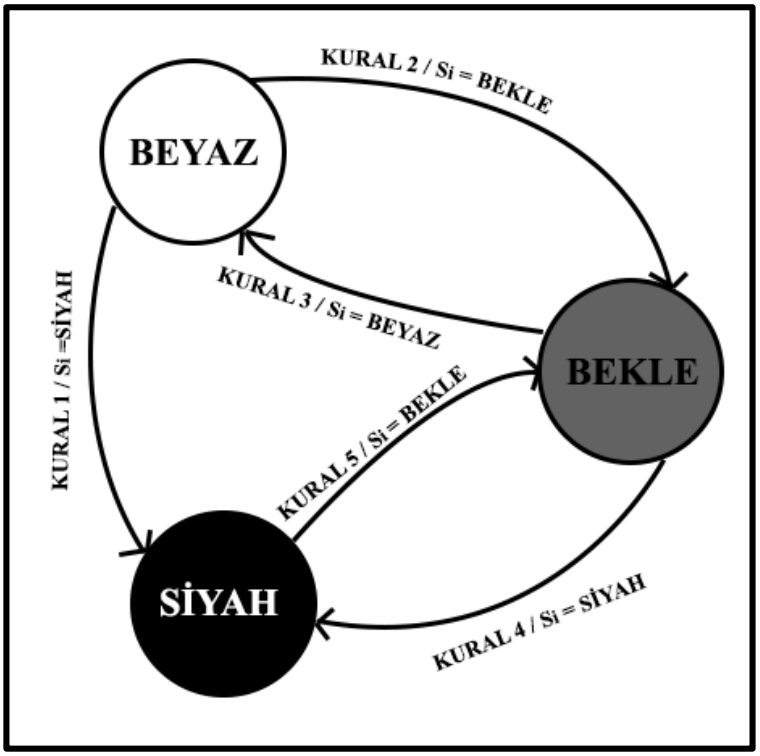
Kural 4: Eğer s_i = BEKLE ve $\exists j \in N_i$ [$s_j \neq$ SİYAH] ve

$\forall j \in N_i$ [s_j = BEKLE ve $id_i < id_j$] ise s_i = SİYAH olur.

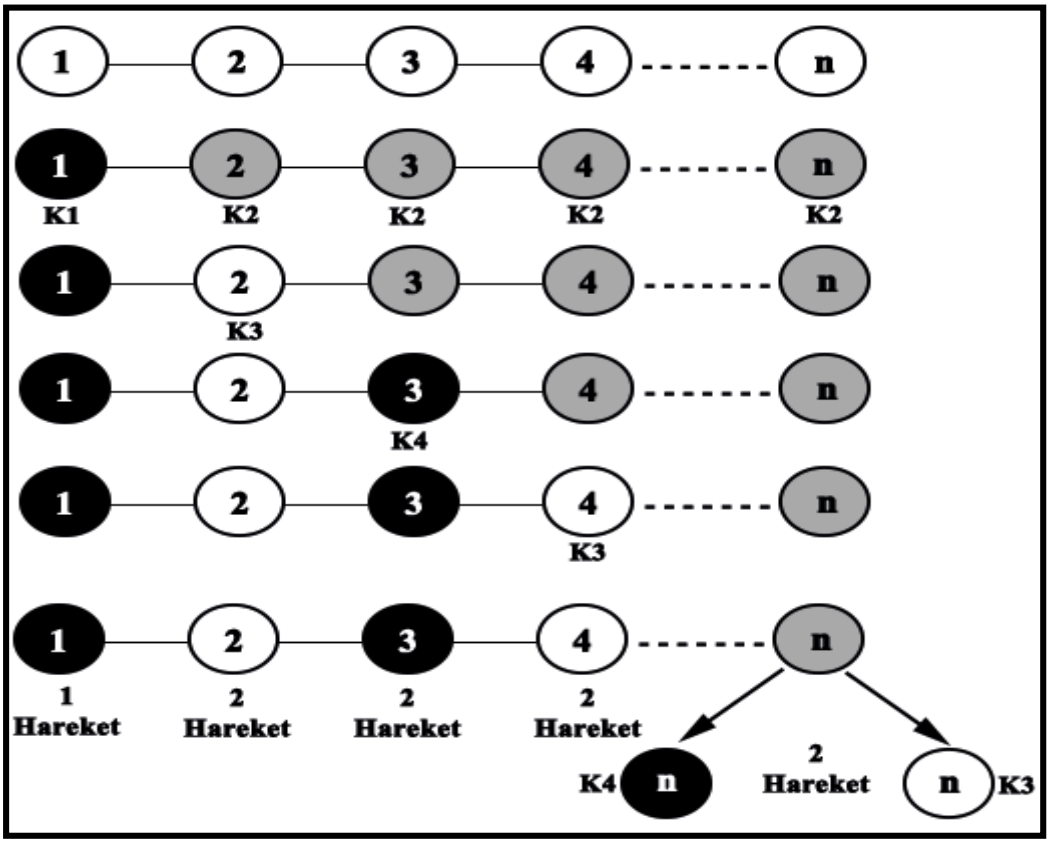
Kural 5: Eğer s_i = SİYAH ve $\exists j \in N_i$ [s_j = SİYAH ve $id_j < id_i$] ise

s_i = BEKLE olur.

Şekil 3.1 OZKAN'ın kuralları



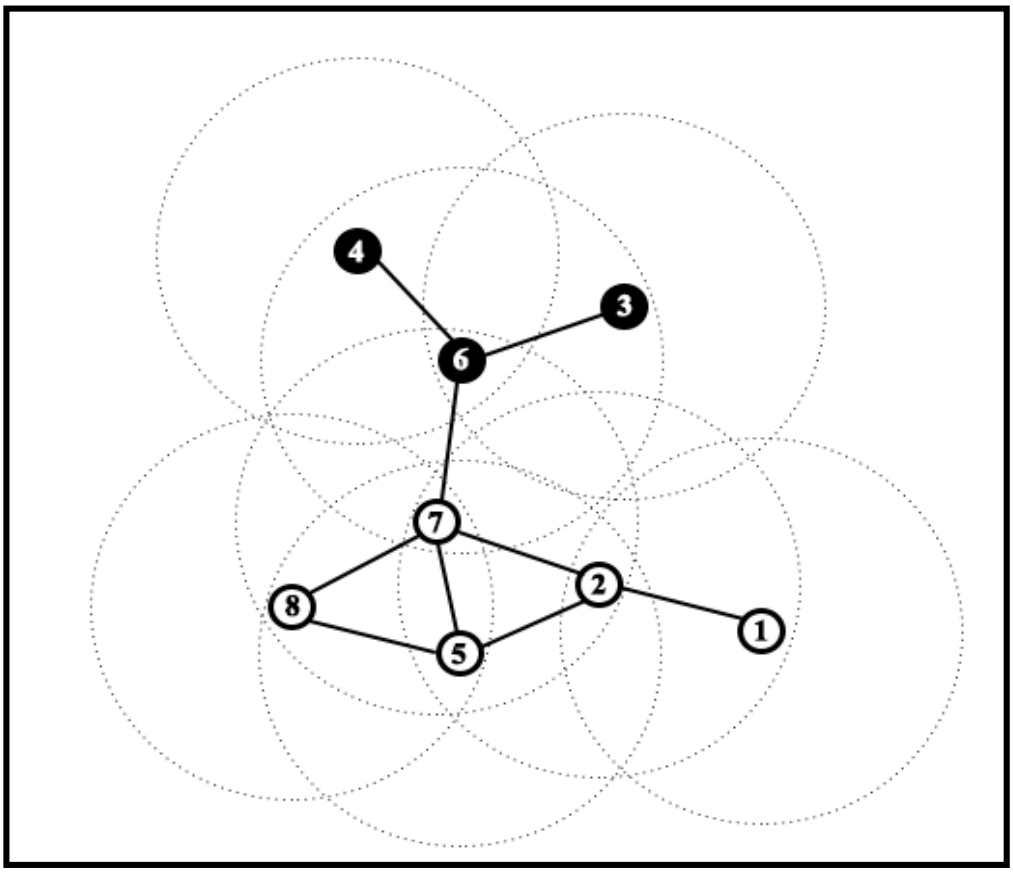
Şekil 3.2 OZKAN'nın sonlu durum makinası ile gösterimi



Şekil 3.3 OZKAN'nın en fazla hareket sayısı ile çalışma örneği

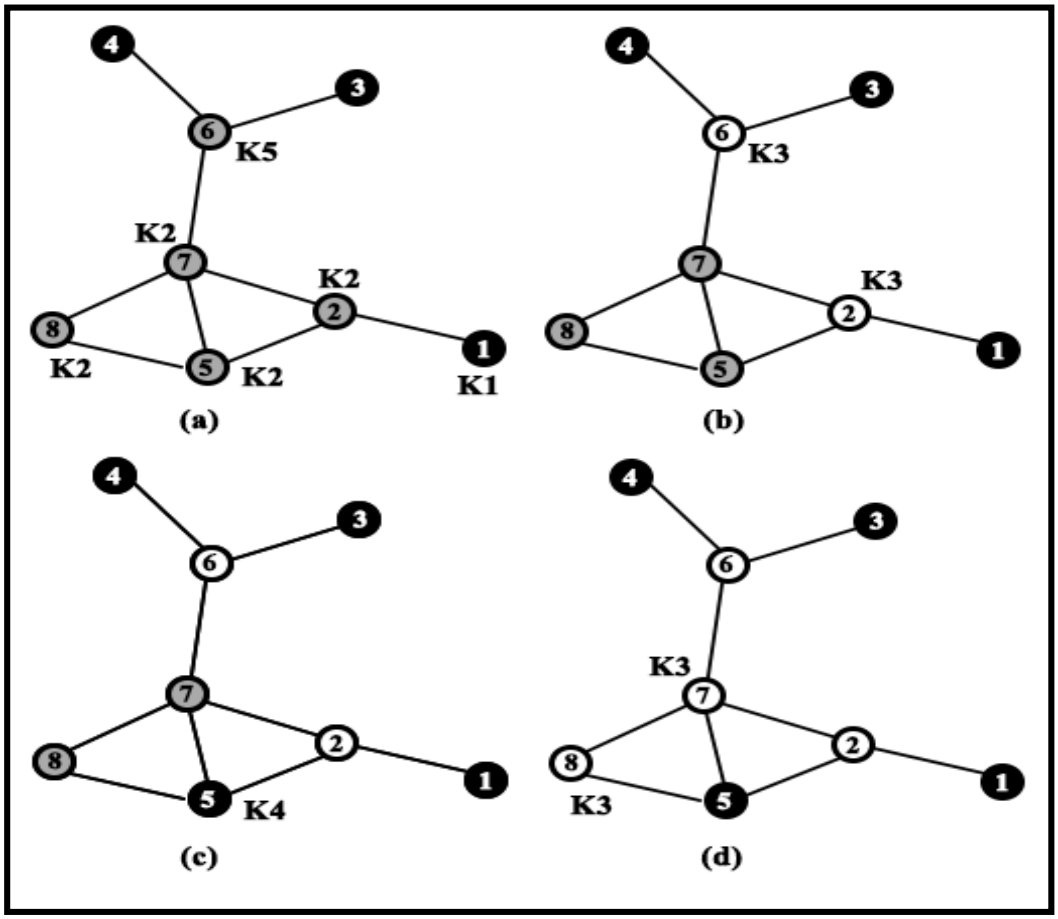
3.2 Örnek İşlem

Şekil 3.4'te OZKAN'ın birim disk çizge üzerinde çalışmasının bir örneği yer almaktadır. Topoloji, yönsüz, bağlı ve tekil *id* sayıları içeren birim disk çizgedir. Sistem dağıtıktır ve TDA'da çalışmaktadır. Bu yüzden birden fazla düğüm aynı anda s_i değerini değiştirebilir. Düğümlerin *id* değeri 1'den 8'e kadar olan ardışık sayılardan oluşmaktadır. Başlangıçta id_1, id_2, id_5, id_7 ve id_8 düğümlerinin s_i değeri BEYAZ, id_3, id_4 ve id_6 düğümlerinin ise s_i değeri SİYAH'tır. Bütün düğümler ilk olarak birbirlerine "Merhaba" mesaj paketi ile *id* ve s_i değerlerini gönderir. Çizgedeki her bir düğüm komşularının *id* ve s_i değerlerini elde ettikten sonra OZKAN'ın beş kuralından herhangi birinin ön koşullarının sağlanıp sağlanmadığını kontrol eder. Herhangi bir kuralının ön koşulları sağlanan düğümler s_i değerini günceller ve komşularına tekrar *id* ve s_i değerlerini yayın yapar. Sistem öz kararlı hale geldiğinde, sistemde sadece SİYAH ve BEYAZ durum değerlerine sahip düğümler vardır. SİYAH durumuna sahip olan düğümler MBK'yı oluşturur. Ayrıca, her bir BEYAZ düğüm mutlaka MBK içindeki bir düğümün komşusudur.



Şekil 3.4 OZKAN'ın birim disk çizge üzerinde çalışmasının başlangıç durumu

Şekil 3.5'te OZKAN'nın birim disk çizge üzerindeki çalışma adımları yer almaktadır. Ağdaki her bir düğüm bütün komşularından "Merhaba" mesajını aldıktan sonra algoritmayı çalıştırır. Birinci turda id_1, Kural 1'i çalıştırır ve s_i değerini SİYAH olarak günceller; id_2, id_5, id_7, id_8 Kural 2'yi çalıştırır ve s_i değerlerini BEKLE olarak değiştirir; id_6 ise Kural 5'i çalıştırır ve s_i değerini BEKLE olarak günceller. Durumları değişen bu düğümler, id ve yeni s_i değerini komşularına yayın yaparak gönderir. Şekil 3.3a'da sistemin birinci turun sonundaki görüntüsü yer almaktadır. İkinci turda id_2 ve id_6 Kural 3'ü çalıştırır ve BEYAZ durumuna geçer. Şekil 3.3b'de sistemin ikinci turun sonundaki görüntüsü yer almaktadır. Üçüncü turda sadece id_5, Kural 4'ü çalıştırır ve SİYAH durumuna geçer. Şekil 3.3c'de sistemin üçüncü turun sonundaki görüntüsü yer almaktadır. Dördüncü turda ise id_7 ve id_8 Kural 3'ü çalıştırır ve BEYAZ durumuna geçer. Şekil 3.3d'de sistemin dördüncü turun sonundaki görüntüsü yer almaktadır. Dördüncü turun sonunda sistem öz kararlı hale gelir ve MBK id_1, id_3, id_4, id_5 düğümlerinden oluşur. Sistemin öz kararlı hale gelebilmesi için toplam 11 hareket yapılmıştır.



Şekil 3.5 OZKAN'ın birim disk çizge üzerindeki çalışma adımları

4. DEĞERLENDİRME

4.1 Teorik Analiz

Bu bölümde ilk olarak literatürdeki algoritmaların hareket sayıları analiz edilmiştir. Kuram 1 ve Sonuç Kuramı 1 ile IKEDA'nın 2 durum için en az sayıda hareket sağladığı ve en az 3 durum kullanan bir algoritmanın en az 2n-1 hareket yapacağı ispat edilmiştir. Ön Kuram 1, Ön Kuram 2, Ön Kuram 3, Ön Kuram 4 ve Kuram 2 ile OZKAN'ın doğruluk analizi yapılmıştır. Sonuç Kuramı 2 ile OZKAN'ın en az hareket sayısının 2n-1 olduğu ve teorik olarak en az sayıda hareket sağladığı ispat edilmiştir.

Kuram 1. IKEDA, 2 durum için en az sayıda hareket sağlar.

İspat. N adet düğümün olduğu doğrusal bir ağ olduğunu varsayalım. Soldan sağa doğru düğüm 1'den düğüm n'ye doğru dizildiklerini, düğümlerin kimlikleri arasında $id_1<id_2<...<id_n$ ilişkisi olduğunu varsayalım. Düğüm i'nin durumu s_i olarak gösterilsin ve tüm düğümler için s_i = BEYAZ olsun. Yeni bir kısıt getirmeden küçük kimliğin öncelikli olduğunu varsayalım. Böylece aşağıdaki basamakların işletilmesi gereklidir.

Basamak 1. Bağımsız kümenin maksimal olabilmesi için düğüm 1 veya düğüm 2'nin hareket yapması gerekmektedir. Küçük kimliğe öncelik verdiğimiz için düğüm 1 hareket yapacak ve s_1=SİYAH olacaktır. Düğüm 2 hareket yapmaz ve BEYAZ durumunda kalır. Bu aşamadan itibaren bağımsız kümenin maksimal olabilmesi için düğüm 3 veya düğüm 4'ün hareket yapması gerekmektedir. Düğüm 3 veya düğüm 4'ü hareket yaptıracak kural aşağıdadır:

Kural 1. Eğer BEYAZ durumundaysam ve tüm komşularım BEYAZ ise SİYAH durumuna geçiş yap.

Basamak 2. Kural 1'in işletilmesinden sonra tüm düğümler SİYAH durumuna geçer. Toplamda n hareket yapılır. Bu aşamadan itibaren kimliği küçük olanların SİYAH önceliği vardır ve kimliği büyük olanlar bağımsız kümenin sağlanabilmesi için BEYAZ durumuna geçiş yapmalıdırlar. Bu geçişi sağlayan kural aşağıdadır:

Kural 2. Eğer SİYAH durumundaysam ve benim kimliğimden daha küçük SİYAH komşum varsa BEYAZ durumuna geçiş yap.

Kural 1 ve Kural 2 sıralı ve tekrarlı olarak bu ağ üzerinde çalıştırıldığında her bir kural çifti için 2 düğümün durumu kesinleşir. Bu işlem tüm düğümlerin durumları kesinleşene kadar devam eder ve toplamda n(n+1)/2 hareket yapılır. Kural 1 ve Kural 2, IKEDA'nın kurallarıdır, IKEDA 2 durum için en az sayıda hareket yapar.

Sonuç Kuramı 1. d >= 1 için 2+d durumlu herhangi bir öz kararlı MBK algoritması için en az hareket sayısı 2n-1'dir.

İspat. Kuram 1'de kullandığımız ağı ele alalım. d>=1 olduğundan dolayı en az yeni bir durum tanımlamamız gerekmektedir, bu durumu BEKLE olarak isimlendirelim. Düğümlerin Kuram 1'de verilen Kural 1 ve Kural 2'yi sıralı ve tekrarlı çalıştırmamaları için Kural 1'i aşağıdaki gibi iki parçaya bölebiliriz:

Kural 1.1 Eğer BEYAZ durumundaysam ve tüm komşularım BEYAZ ise ve kimliğim en küçük ise SİYAH durumuna geçiş yap.

Kural 1.2 Eğer BEYAZ durumundaysam ve tüm komşularım BEYAZ ise ve kimliğim en küçük değilse BEKLE durumuna geçiş yap.

Kural 1.1 ve Kural 1.2 çalıştırıldığında toplamda n hareket yapılacaktır. Bu aşamada düğüm 1 SİYAH durumunda, geride kalan tüm düğümler ise BEKLE konumundadır. Bu noktadan itibaren tasarlanabilecek en etkili algoritmanın yapabileceği BEKLE durumundaki düğümleri tek hareket kullanarak SİYAH veya BEYAZ durumuna geçirebilmektir. Bir başka ifadeyle n-1 hareket daha gereklidir. Sonuç olarak toplamda 2n-1 hareket gereklidir.

Ön Kuram 1. OZKAN'da herhangi bir düğümün kuralı aktif olamadığı zaman SİYAH düğümlerin oluşturduğu küme maksimal bağımsız kümedir.

İspat. BEKLE durumunda bekleyen bir v düğümü bulunduğunu farz edelim ve hiçbir kural aktif olmasın. Eğer v düğümünün SİYAH komşusu varsa Kural 3 aktif olur ve BEYAZ durumuna geçiş yapar. Eğer SİYAH komşusu yoksa ve BEKLE durumundaki komşuları içinde en küçük kimliğe sahipse Kural 4 aktif olur SİYAH durumuna geçiş yapar. Eğer SİYAH komşusu yoksa ve BEKLE

durumundaki komşuları içinde en küçük kimliğe sahip değilse BEKLE durumundan kurtulmak için Kural 3 veya Kural 4'ün aktif olmasını bekler. Kural 3'ün aktif olmaması için SİYAH komşusu olmaması gereklidir. SİYAH komşu olmaması için bu komşuların SİYAH komşularının olması gereklidir. Düğüm v'nin komşuları bu durumda Kural 3'ü çalıştıracak ve BEYAZ duruma geçecektir. Düğüm v'nin BEYAZ olmayan diğer komşuları BEKLE durumundadır. Bu aşamada Kural 4 ya düğüm v için ya da onun BEKLE durumundaki komşularından biri için aktif olacaktır. Her olasılık için Kural 3 veya Kural 4 aktiftir, kuramdaki varsayımımız ile çelişiyoruz, bir düğümün BEKLE durumunda kalması mümkün değildir. Kural 5 aktif olmadığından dolayı SİYAH durumundaki düğümler bağımsızdır. Kural 1 ve Kural 2 aktif olmadıklarından dolayı küme genişletilemez.

Ön Kuram 2. OZKAN'da eğer herhangi bir düğüm Kural 4'ü çalıştırırsa bir daha kural çalıştırmaz. Bu düğümün komşuları en fazla 1 kural çalıştırır ve bu kural, Kural 3'tür.

İspat. Düğüm v'nin Kural 4'ü çalıştırdığını farz edelim. Bu olay gerçekleşecek ise düğüm v'nin komşularının BEYAZ olması veya kimliği büyük olup BEKLE durumunda olması gerekmektedir. Bundan dolayı düğüm v'nin komşularının Kural 4 veya Kural 3'ü çalıştırması mümkün değildir. Bir düğümün Kural 1 veya Kural 2'yi çalıştırması için tüm komşularının BEYAZ olması gereklidir, bundan dolayı düğüm v'nin komşuları Kural 1 veya Kural 2'yi çalıştıramaz. Düğüm v'nin komşuları için çalıştırılabilir tek kural, Kural 3'tür. Böylece düğüm v'nin tüm komşuları BEYAZ olur. Düğüm v, Kural 4'ü çalıştırdıktan sonra SİYAH olacağından dolayı tek çalıştıracağı kural, Kural 5'tir. Kural 5'in çalıştırabilmesi için düğüm v'nin SİYAH komşusu olması gereklidir ki bu olasılığın gerçekleşmesi mümkün değildir.

Ön Kuram 3. OZKAN'da eğer bir düğüm Kural 1'i çalıştırırsa bir daha başka kural çalıştırmaz. Bu düğümün komşuları en fazla 2 kural (Kural 2 ve Kural 3'ü) çalıştırır.

İspat. Düğüm v'nin Kural 1'i çalıştırabilmesi için tüm komşularının BEYAZ olması gereklidir ve düğüm v'nin kimliğinin komşu düğümler içinde en küçük olması gereklidir. Düğüm v'nin Kural 1'i çalıştırdığını farz edelim. Düğüm v, SİYAH durumuna geçecektir ve hiçbir kural onun SİYAH durumunu

değiştirmeyecektir. Düğüm v'nin komşusu düğüm u için aşağıdaki durumlar mümkündür:

Düğüm u, BEYAZ durumundadır ve düğüm u'nun tüm komşuları BEYAZ durumundadır, düğüm u Kural 2'yi çalıştırıp BEKLE durumuna geçer, daha sonra düğüm v SİYAH olduğu için Kural 3'ü çalıştırıp BEYAZ durumuna geçer.

Düğüm u, BEYAZ durumundadır ve düğüm u'nun tüm komşuları BEYAZ değildir. BEYAZ kalmaya devam eder.

Ön Kuram 4. OZKAN çalışırken sadece aşağıdaki beş durum sıralamaları ve onların son ekleri (Ing. *Suffixes*) mümkündür:

SİYAH BEKLE SİYAH
SİYAH BEKLE BEYAZ
SİYAH BEKLE BEYAZ BEKLE SİYAH
SİYAH BEKLE BEYAZ BEKLE BEYAZ
BEYAZ SİYAH

İspat. Öncelikle düğüm v'nin SİYAH olduğunu kabul edelim. Düğüm v'nin komşusu, düğüm v'den kimliği küçük ve SİYAH durumda olan en az bir düğüm olduğunu kabul edelim. Böylece, düğüm v Kural 5'i çalıştırıp BEKLE konumuna geçecektir. Düğüm v'den kimliği küçük olan komşuların önce BEKLE daha sonra ise BEYAZ duruma geçtiğini varsayalım. Bu işlemler gerçekleştirdiğinde düğüm v Kural 4'ü çalıştırıp SİYAH duruma geçer ve Ön Kuram 2'ye göre bir daha kural çalıştırmaz. Bu işlemlerin sonunda, düğüm v SİYAH BEKLE SİYAH durumlarına geçer.

Yukarıda anlatılan duruma benzer düğüm v'nin SİYAH BEKLE durumlarına geçtiklerini varsayalım. Düğüm v BEKLE durumundayken SİYAH bir komşusu olduğunu varsayalım. Böylece düğüm v Kural 3'ü çalıştırıp BEYAZ konumuna geçer ve SİYAH BEKLE BEYAZ geçişleri olur. Bu aşamadan itibaren düğüm v sadece Kural 1 ve Kural 2'yi çalıştırabilir. Düğüm v, SİYAH durumundan BEKLE durumuna geçtiği için komşularından birinin kimliğinin ondan küçük olması gerekir, böylece düğüm v, Kural 1'i çalıştıramaz. Düğüm v, Kural 2'yi çalıştırırsa BEKLE durumuna geçer. Böylece düğüm v, Kural 3 ve Kural 4'ü çalıştırabilir. Eğer Kural 4'ü çalıştırırsa SİYAH olur ve Ön Kuram 2'ye göre bir daha kural çalıştırmaz, SİYAH BEKLE BEYAZ BEKLE SİYAH

geçişleri olur. Eğer Kural 3'ü çalıştırırsa buna sebep olan komşu düğümün Kural 4'ü çalıştırmış olması gereklidir. Düğüm v, Ön Kuram 2'ye göre bir daha kural çalıştırmaz, SİYAH BEKLE BEYAZ BEKLE BEYAZ geçişleri olur.

Düğüm v'nin BEKLE durumunda başladığını varsayalım. Önceki 2 paragrafta anlatıldığı üzere BEKLE SİYAH, BEKLE BEYAZ, BEKLE BEYAZ BEKLE SİYAH ve BEKLE BEYAZ BEKLE BEYAZ geçişleri mümkündür.

Düğüm v'nin BEYAZ durumunda başladığını düşünelim. Eğer Kural 1'i çalıştırırsa BEYAZ SİYAH geçişi olur ve Ön Kuram 3'e göre başka kural işletilmez. Eğer Kural 2'yi çalıştırırsa BEKLE durumuna geçer. Düğüm v'nin Kural 2'yi çalıştırabilmesi için tüm komşularının BEYAZ olması gereklidir. Düğüm v'nin komşularından herhangi biri Kural 1'i çalıştırırsa düğüm v BEYAZ BEKLE BEYAZ geçişleri yapar ve Ön Kuram 3'e göre bu durumda kalır. Eğer düğüm v'nin komşularından biri önce Kural 2 daha sonra Kural 4'ü çalıştırırsa düğüm v Kural 3'ü çalıştırır BEYAZ BEKLE BEYAZ geçişleri yapar ve Ön Kuram 2'ye göre bu durumda kalır. Eğer düğüm v'nin kimliği küçük komşuları, BEYAZ durumunda kalırsa ve kimliği büyük olanlar BEYAZ veya BEKLE durumunda kalırlarsa düğüm v Kural 4'ü çalıştırıp SİYAH durumuna geçer, BEYAZ BEKLE SİYAH geçişleri yapılır ve Ön Kuram 2'ye göre bu durumda kalır.

Kuram 2. OZKAN'ın çalışması sonucunda maksimal bağımsız küme en fazla 2n-1 hareket ile bulunur.

İspat. OZKAN'ın maksimal bağımsız kümeyi doğru olarak bulması Ön Kuram 1, Ön Kuram 2, Ön Kuram 3 ve Ön Kuram 4'ün sonucudur. En kötü durum Kuram 1'de anlatılan ağ yapısıdır. Bu ağda OZKAN'ı çalıştıran düğüm 1 Kural 1'i çalıştırarak BEYAZ durumdan SİYAH duruma geçer. Diğer düğümler ise Kural 2'yi çalıştırarak BEYAZ durumdan BEKLE durumuna geçer. Böylece n hareket yapılmış olur. BEKLE durumundaki düğüm 2, düğüm 1 SİYAH durumunda olduğundan dolayı Kural 3'ü çalıştırır ve BEYAZ durumuna geçer. Düğüm 2 BEYAZ olduktan sonra Düğüm 3 Kural 4'ü çalıştırır ve SİYAH durumuna geçer. Bundan sonra (Düğüm 4, Düğüm 5), (Düğüm 6, Düğüm 7), (Düğüm n-1, Düğüm n) çiftleri sırasıyla Kural 3 ve Kural 4'ü çalıştırır. Böylece n-1 hareket daha yapılmış olur. Toplamda 2n-1 hareket yapılır.

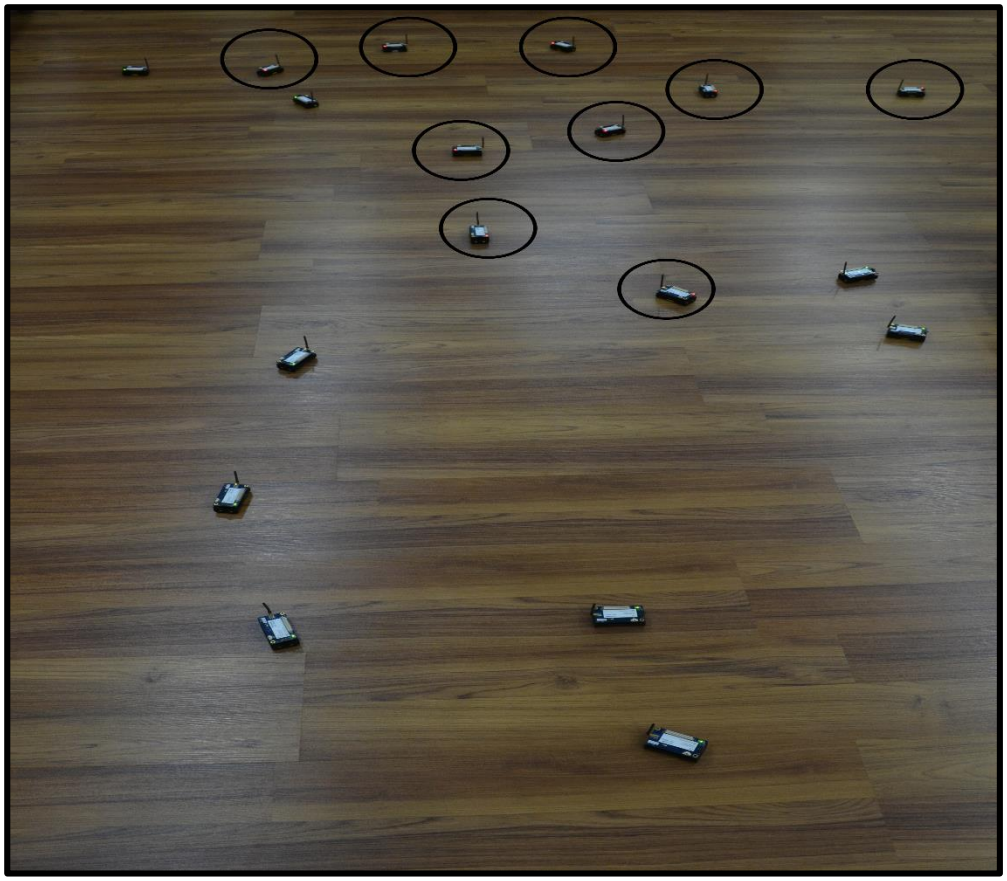
Sonuç Kuramı 2. OZKAN en kötü durumda bir dağıtık öz kararlı MBK algoritması için en az sayıda hareket yaparak maksimal bağımsız kümeyi bulur.

İspat. Kuram 2 ve Sonuç Kuramı 1'in ortak sonucudur.

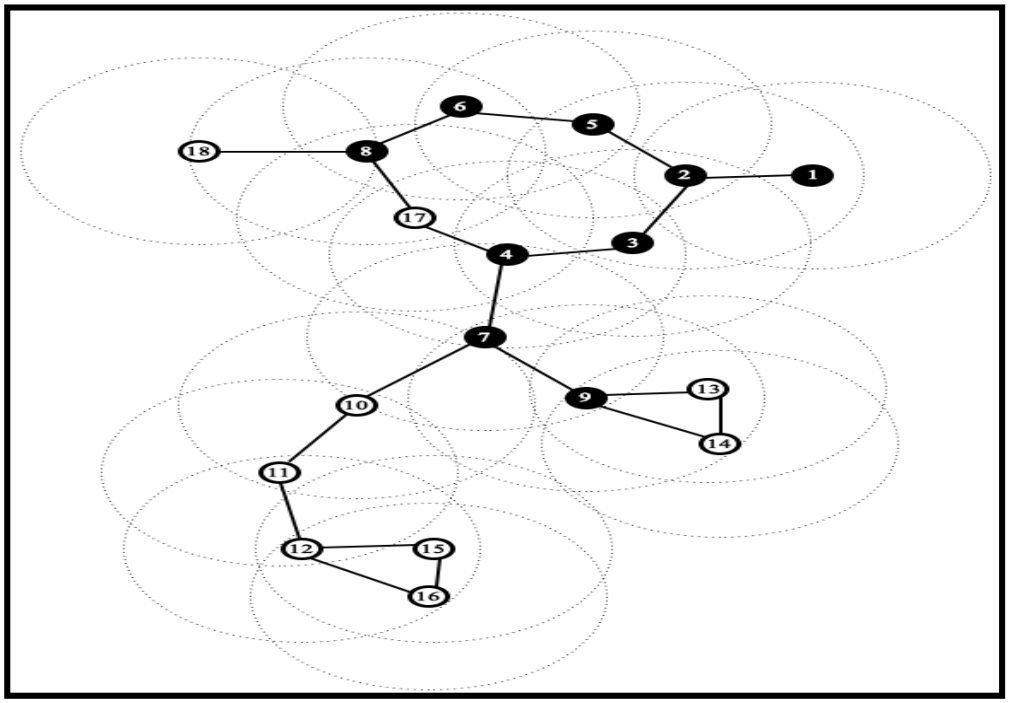
4.2 Uygulamaların Değerlendirmesi

IKEDA, TURAU ve OZKAN, TinyOS işletim sistemini kullanan Iris düğümler üzerinde çalıştırılmıştır. TOSSIM ile düğümlere yazılımlar yüklenmiştir. Uygulamada 18 adet Iris düğüm algoritmayı çalıştırmak, 1 düğüm algoritmayı başlatmak ve bir düğüm de ağdaki verileri toplamak için kök düğüm olarak kullanılmıştır. Bütün düğümlerde üç algoritma da yer almaktadır. Başlatıcı (Ing. *Starter*) düğümün gönderdiği mesaj içeriğine göre düğümler istenilen algoritmayı çalıştırır. Düğümler başlatıcıdan mesaj aldıktan sonra komşularına başlangıç *id* ve s_i değerlerini gönderirler. Gönderilen veri 2 bayt olmasına rağmen, paketin başlık (Ing. *Header*) bölümü ile birlikte her mesaj paketinin boyutu 10 bayt'tır. Yani her mesaj paketinin başlığı sisteme 8 bayt ek yük getirmiştir. Düğümlerin üzerinde kırmızı, yeşil ve sarı olmak üzere üç renk ışık yer almaktadır. Kırmızı ışık yanıyorsa düğümün s_i değeri SİYAH'tır, yeşil ışık yanıyorsa s_i değeri BEYAZ'dır ve sarı ışık yanıyorsa s_i değeri BEKLE'dir. Bir düğümün durumu değiştiğinde ışık rengi değişir. Kullanılan topolojiler bağlı, yönsüz ve tekil *id* numaralarına sahip düğümlerden oluşan birim disk çizgedir. Uygulamalar seyrek (Ing. *Sparse*) ve yoğun (Ing. *Dense*) olmak üzere iki farklı topolojide çalıştırılmıştır.

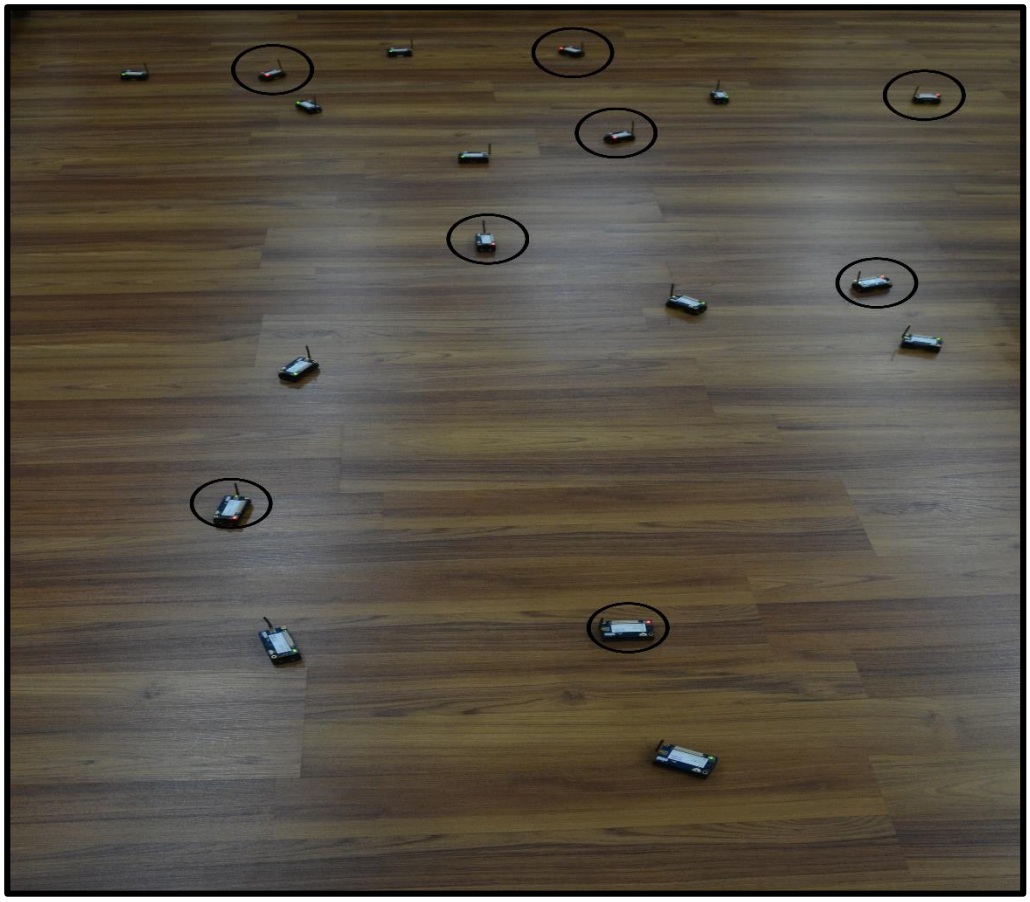
Şekil 4.1'de seyrek topolojide düğümlerin başlangıç durumlarının gerçek resmi yer almaktadır. Kırmızı ışık yanan düğümler daire içine alınmıştır. Şekil 4.2'de ise seyrek topolojide düğümlerin başlangıç durumlarının çizimi yer almaktadır. Başlangıçta düğümlerin yarısının s_i değeri BEYAZ, diğer yarısının ise SİYAH'tır. Bütün düğümler başlatıcı düğümden mesaj aldıktan sonra paketin içeriğine göre hangi algoritmayı çalıştıracaksa o algoritmayı başlatır. Başlangıçta bütün düğümler komşularına başlangıç *id* ve s_i değerlerini gönderir. Her düğüm bütün komşularından mesaj aldıktan sonra çalıştırdığı algoritmanın kurallarının ön koşullarının sağlanıp sağlanmadığını kontrol eder. Eğer bir kuralın ön koşulları sağlanmışsa düğüm durumunu değiştirir ve yeni s_i değerini komşularına yayın yapar. Sistem öz kararlı hale geldiğinde MBK oluşur. Şekil 4.3'de seyrek topolojide oluşan öz kararlı MBK'nın gerçek resmi yer almaktadır. Şekil 4.4'de ise seyrek topolojide oluşan öz kararlı MBK'nın çizimi yer almaktadır.



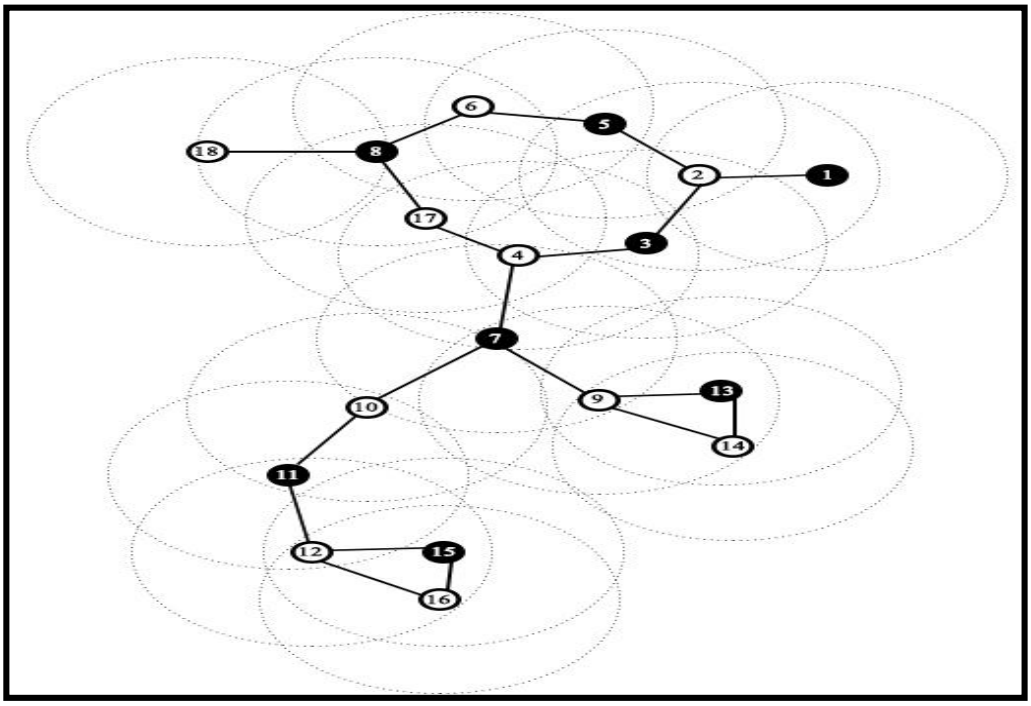
Şekil 4.1 Seyrek topolojide düğümlerin başlangıç durumu gerçek resmi



Şekil 4.2 Seyrek topolojide düğümlerin başlangıç durumu çizimi



Şekil 4.3 Seyrek topolojide öz kararlı MBK'nın gerçek resmi



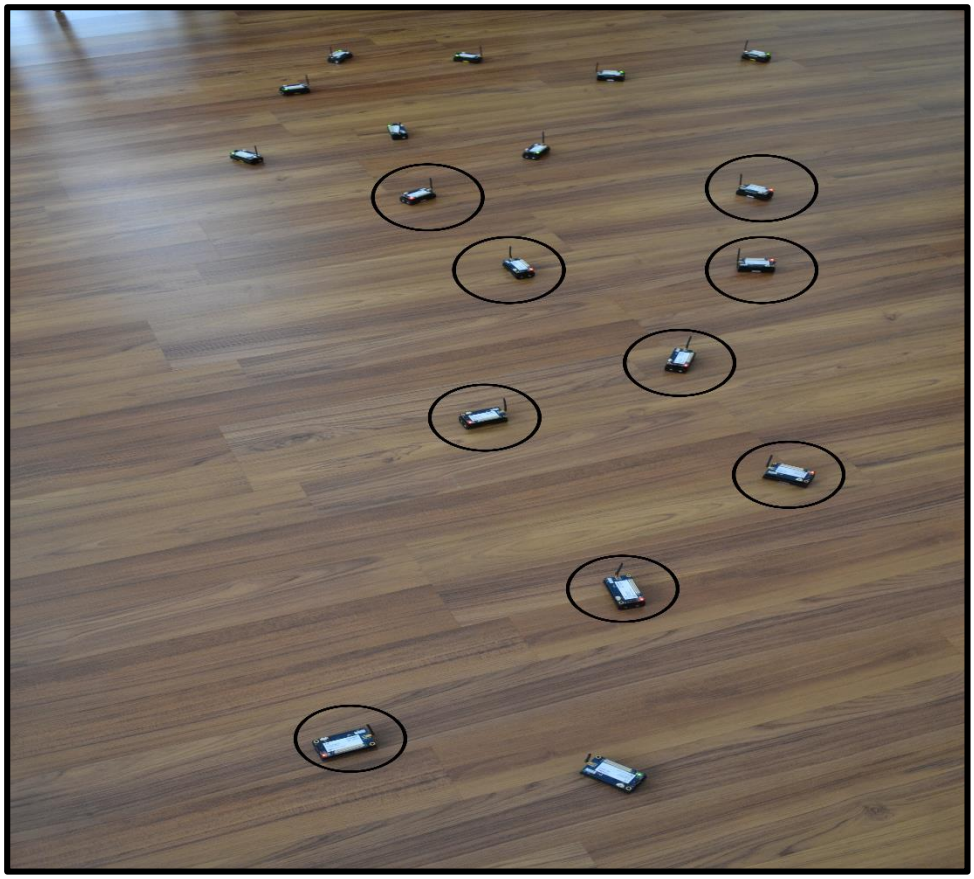
Şekil 4.4 Seyrek topolojide öz kararlı MBK'nın çizimi

Çizelge 4.1'de seyrek topolojide çalıştırılan algoritmaların kök düğüm ile elde edilen verileri yer almaktadır. Sistemin öz kararlı hale gelmesi ve MBK'nın oluşması için en fazla hareket sayısına sahip olan algoritma IKEDA'dır. Hareket sayısı fazla olduğu için gönderilen bayt sayısı, alınan bayt sayısı da daha fazladır. Gönderilen ve alınan bayt sayısı fazla olan sistemde tüketilen enerji de fazladır. TURAU, IKEDA'dan daha etkin çalışmıştır. Hareket sayısının, gönderilen bayt sayısının, alınan bayt sayısının ve tüketilen enerji miktarının en az olduğu algoritma OZKAN'dır. Bu yüzden OZKAN seyrek topolojide en etkin çalışan algoritma olmuştur.

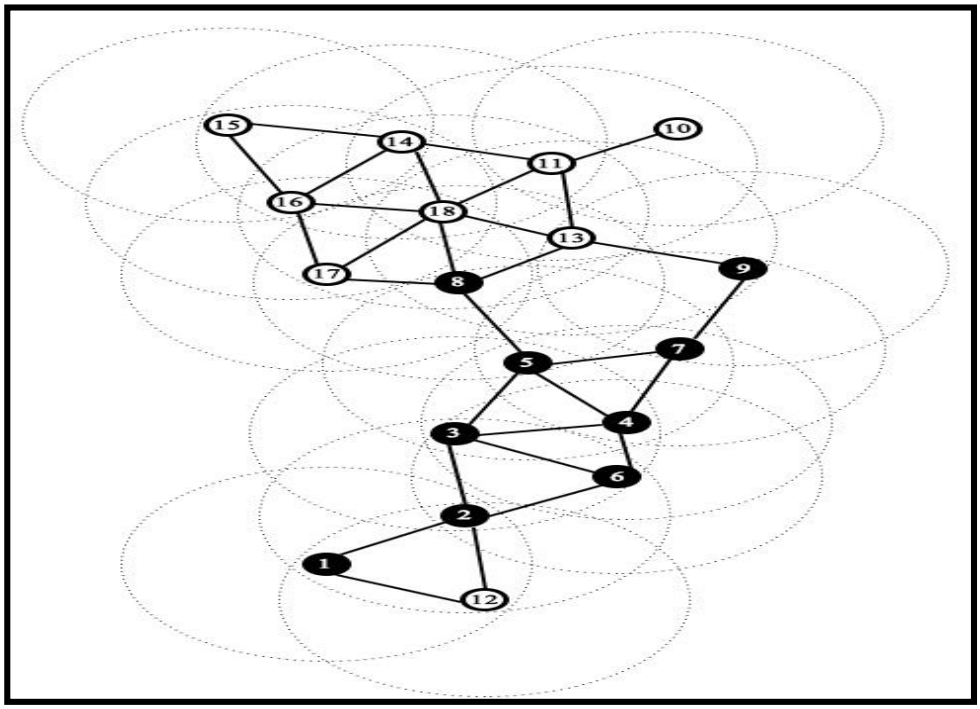
Çizelge 4.1 Seyrek topolojide kök düğüm sonuçları

	HAREKET SAYISI	**GÖNDERİLEN BAYT SAYISI**	**ALINAN BAYT SAYISI**	**TÜKETİLEN ENERJİ (mJ)**
IKEDA	53	530	1140	27,962
TURAU	40	400	880	21,384
OZKAN	28	280	660	15,598

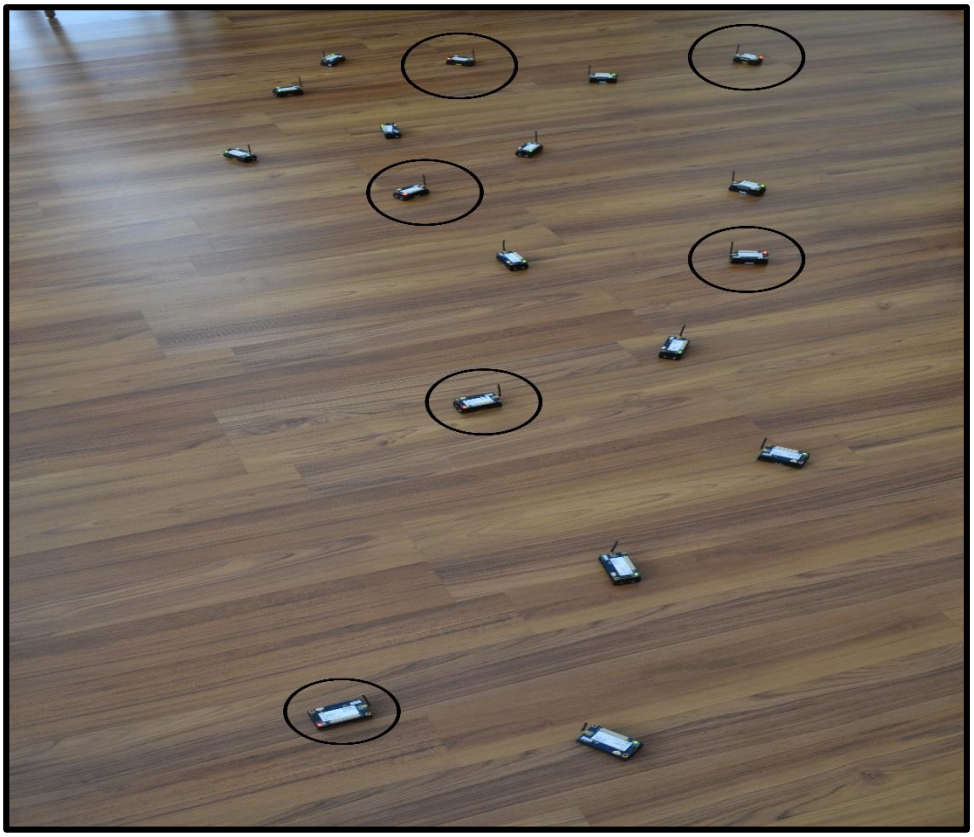
Şekil 4.5'de yoğun topolojide düğümlerin başlangıç durumlarının gerçek resmi yer almaktadır. Şekil 4.6'da ise yoğun topolojide düğümlerin başlangıç durumlarının çizimi yer almaktadır. Şekil 4.7'de yoğun topolojide oluşan öz kararlı MBK'nın gerçek resmi yer almaktadır. Şekil 4.8'de ise yoğun topolojide oluşan öz kararlı MBK'nın çizimi yer almaktadır.



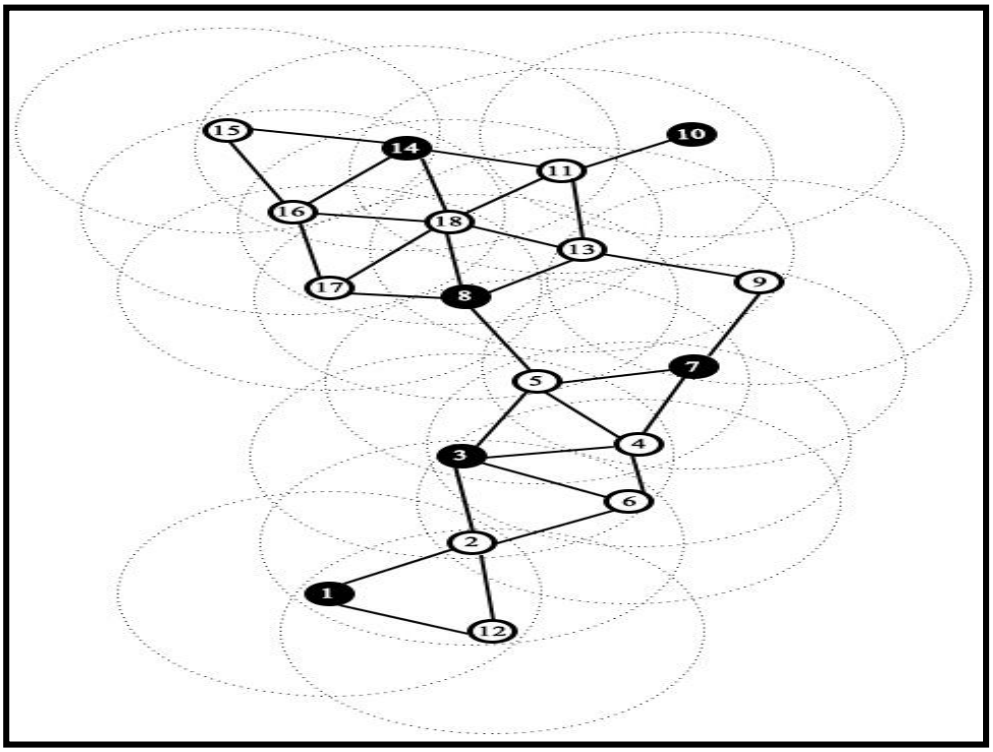
Şekil 4.5 Yoğun topolojide düğümlerin başlangıç durumu gerçek resmi



Şekil 4.6 Yoğun topolojide düğümlerin başlangıç durumu çizimi



Şekil 4.7 Yoğun topolojide öz kararlı MBK'nın gerçek resmi



Şekil 4.8 Yoğun topolojide öz kararlı MBK'nın çizimi

Çizelge 4.2'de yoğun topolojide çalıştırılan algoritmaların kök düğüm ile elde edilen verileri yer almaktadır. IKEDA sistemin öz kararlı hale gelmesi ve MBK'nın oluşması için en fazla hareket sayısına sahip olan algoritmadır. Buna bağlı olarak gönderilen bayt sayısı, alınan bayt sayısı da daha fazladır; gönderilen ve alınan bayt sayısı fazla olduğundan tüketilen enerjide daha fazladır. TURAU, seyrek topolojide olduğu gibi IKEDA'dan daha etkin çalışmıştır. En az hareket sayısına, gönderilen bayt sayısına, alınan bayt sayısına ve tüketilen enerji miktarına sahip olan algoritma OZKAN'dır. Bu yüzden OZKAN seyrek topolojide olduğu gibi yoğun topolojide de en etkin çalışan algoritma olmuştur.

Çizelge 4.2 Yoğun topolojide kök düğüm sonuçları

	HAREKET SAYISI	**GÖNDERİLEN BAYT SAYISI**	**ALINAN BAYT SAYISI**	**TÜKETİLEN ENERJİ (mJ)**
IKEDA	41	410	1360	28,468
TURAU	38	380	1200	25,52
OZKAN	25	250	850	17,655

Uygulamalarda kullanılan Iris düğümler 3,3 volta sahiptir. Iris düğümlerin veri gönderirken ve alırken ne kadar akım harcadığı dikkate alınarak (http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf, Erişim Tarihi: 11.18.2014), sistemin başlangıçtan öz kararlı duruma gelene kadar tüketilen enerji "(((Gönderilen bayt sayısı * 0,2) + (Alınan bayt sayısı * 0.13))/30)*3.3) mJ" formülü ile hesaplanmıştır. Bir düğüm kural çalıştırırken, mikroişlemcinin harcadığı enerji ihmal edilebilir (Karl and Willig, 2005).

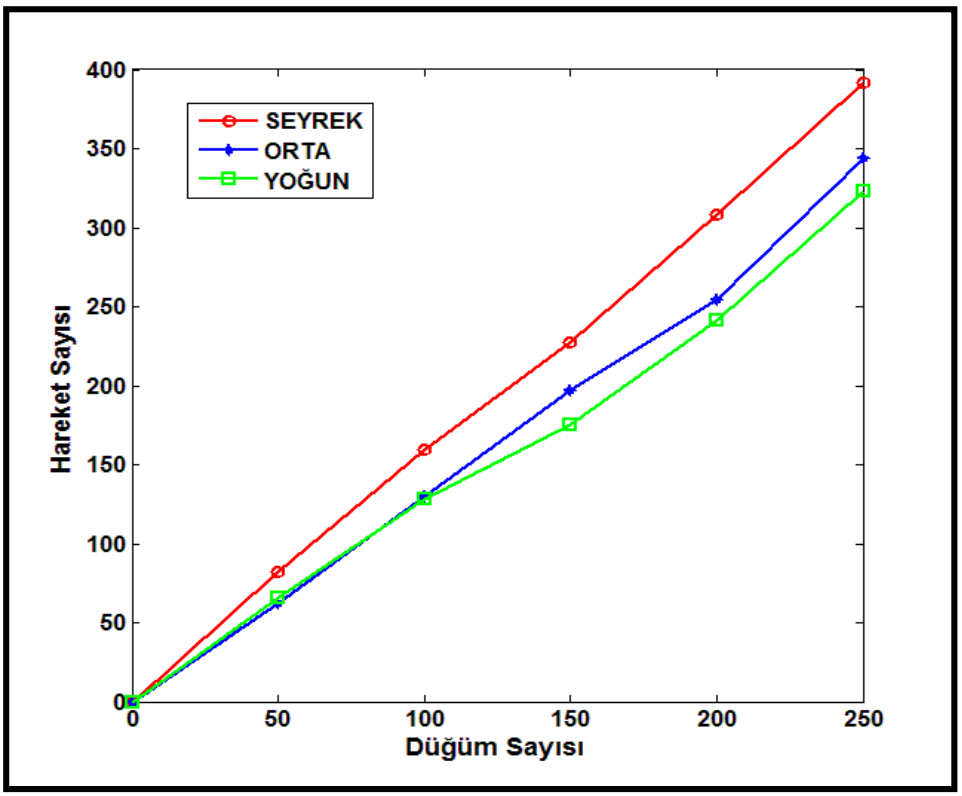
Seyrek ve yoğun topolojilerin her ikisinde de gerçek uygulamalar, hareket sayısının, gönderilen ve alınan bayt sayısının ve enerji tüketiminin en fazla olduğu algoritmanın IKEDA olduğunu göstermiştir. TURAU'nun sonuçları IKEDA'dan daha etkindir. OZKAN ise hareket sayısının, gönderilen ve alınan bayt sayısının, tüketilen enerji miktarının en az olması sebebiyle her iki topolojide de en etkin çalışan algoritma olmuştur. Algoritmaların yüksek sayıda düğümlerden oluşan topolojiler üzerinde test işlemi TOSSIM simülatöründe gerçekleştirilmiştir. Gerçek uygulamaların videosu (http://www.youtube.com/watch?v=VytqV07mbss, Erişim Tarihi: 19.12.2014) internet bağlantısında yer almaktadır.

4.3 Benzetimlerin Değerlendirmesi

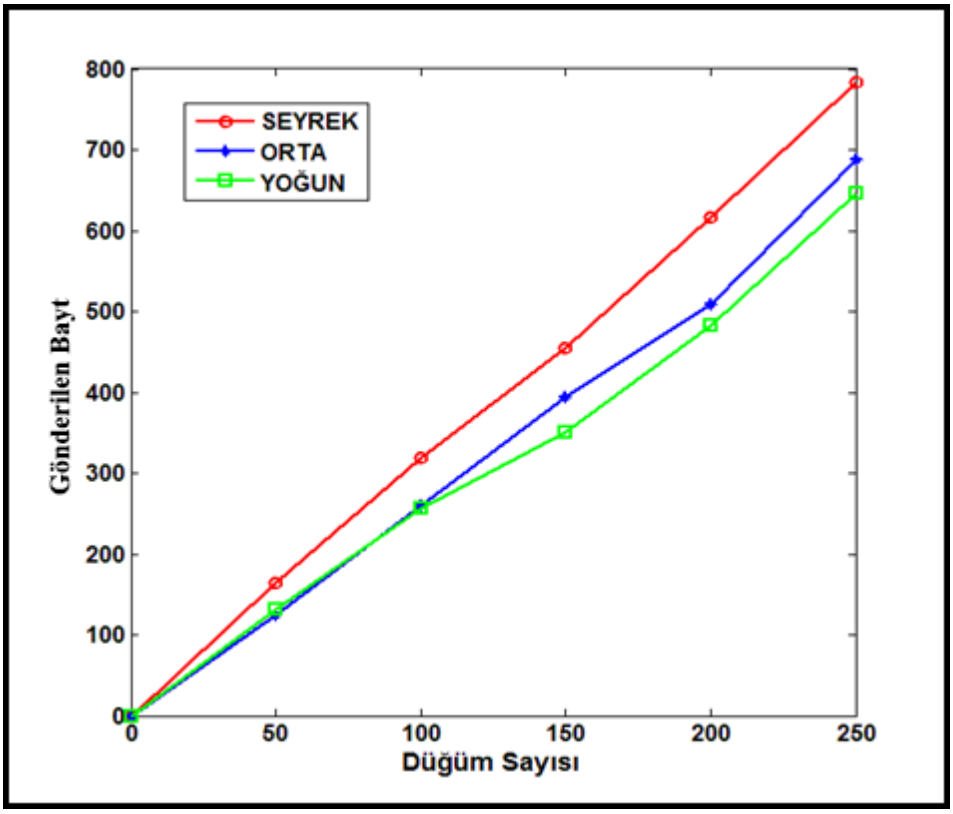
IKEDA, TURAU ve OZKAN TOSSIM simülatöründe rasgele oluşturulmuş olan birim disk çizgeler üzerinde test edildi. Her bir ölçüm için 50 farklı topoloji ve 50 ile 250 arasında 50'şer aralıklarla değişen düğüm sayıları kullanıldı. Grafiklerdeki değerler alınan sonuçların ortalamasıdır. Bütün algoritmaların TDA ortamında çalıştırıldığı varsayıldı. Topolojiler bağlı, yönsüz ve düğümleri tekil *id* değerine sahip olan çizgelerden oluşmaktadır. Algoritmalar turlar içinde çalışmaktadır. Düğümlerin yarısının başlangıç s_i değeri SİYAH, diğer yarısının başlangıç s_i değeri ise BEYAZ'dır. Başlangıçta bütün düğümler "Merhaba" mesajı ile birbirlerine başlangıçtaki *id* ve s_i değerlerini gönderir. Bütün komşularından mesaj alan düğüm, algoritmanın kurallarından herhangi birinin ön koşullarının sağlanıp sağlanmadığını kontrol eder. Eğer bir kuralın ön koşulu sağlanmış ise s_i değerini yayın yaparak komşularına gönderir. Sistem dağıtık olduğu için birden fazla düğüm aynı anda s_i değerini değiştirebilir. Sistem öz kararlı hale gelene kadar işlemler turlar halinde devam eder. Sistem öz kararlı olduğunda MBK oluşur.

Şekil 4.9'daki grafik değişen düğüm sayılarına bağlı seyrek, orta ve yoğun topolojilerde OZKAN çalıştırıldığında yapılan hareket sayılarını göstermektedir. Bu grafikte görüldüğü gibi OZKAN'ın çalıştırıldığı seyrek ağlardaki düğümler daha fazla hareket yaparak öz kararlı hale gelmiştir. Ağ yoğunluğu arttıkça hareket sayısı düşmektedir. Değişen düğüm sayısına baktığımızda ise düğüm sayısı arttıkça hareket sayısı da genel olarak artmaktadır. Bu sonuçlar algoritmanın ölçeklenebilirliğini göstermektedir.

Şekil 4.10'daki grafik değişen düğüm sayılarına bağlı seyrek, orta ve yoğun topolojilerde OZKAN çalıştırıldığında düğümlerin gönderdiği toplam bayt sayısını göstermektedir. Ağ yoğunluğu arttıkça hareket sayısı azaldığı için gönderilen mesaj sayısı da azalmaktadır. Çünkü hareket eden düğüm yeni s_i değerini komşularına yayın yapmaktadır. Mesaj sayısının azalması gönderilen toplam bayt sayısını azaltır. Bu durumda seyrek ağlardaki düğümler MBK'yı oluşturmak için daha fazla hareket yaparak öz kararlı hale gelirken; yoğun ağlardaki düğümler daha az hareket yaparak öz kararlı hale gelir. Değişen düğüm sayısına baktığımızda ise düğüm sayısı arttıkça gönderilen bayt sayısı da artmaktadır.



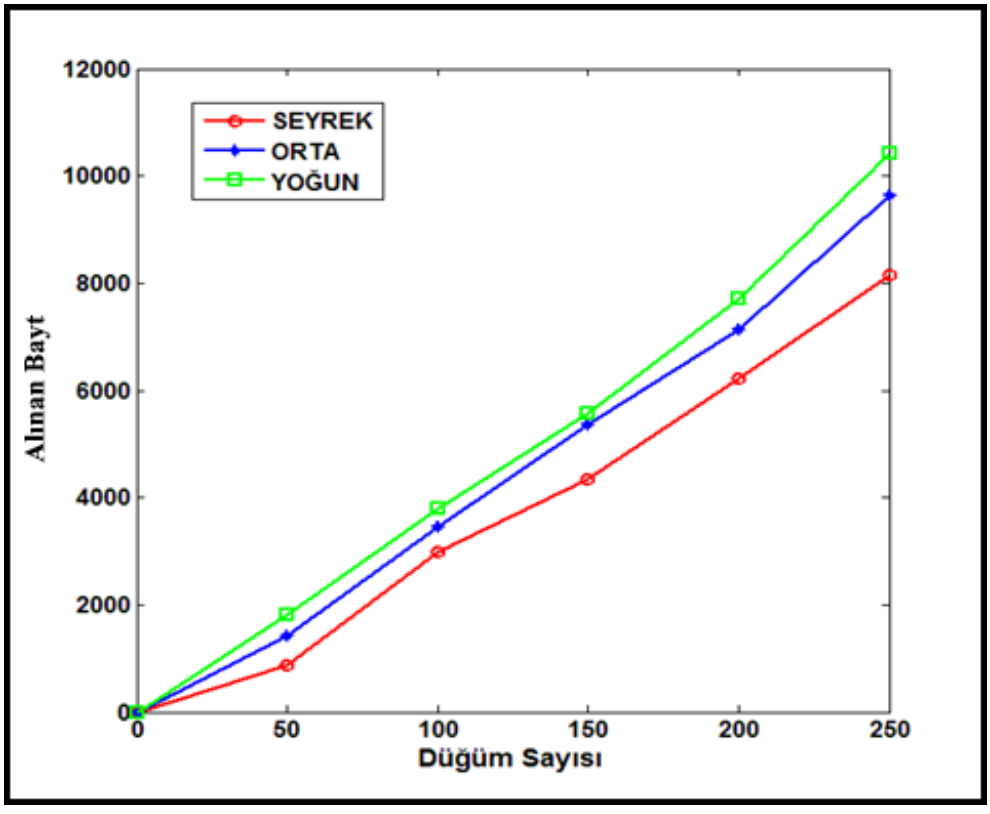
Şekil 4.9 Değişen düğüm sayısına bağlı farklı yoğunluktaki ağlarda hareket sayısı



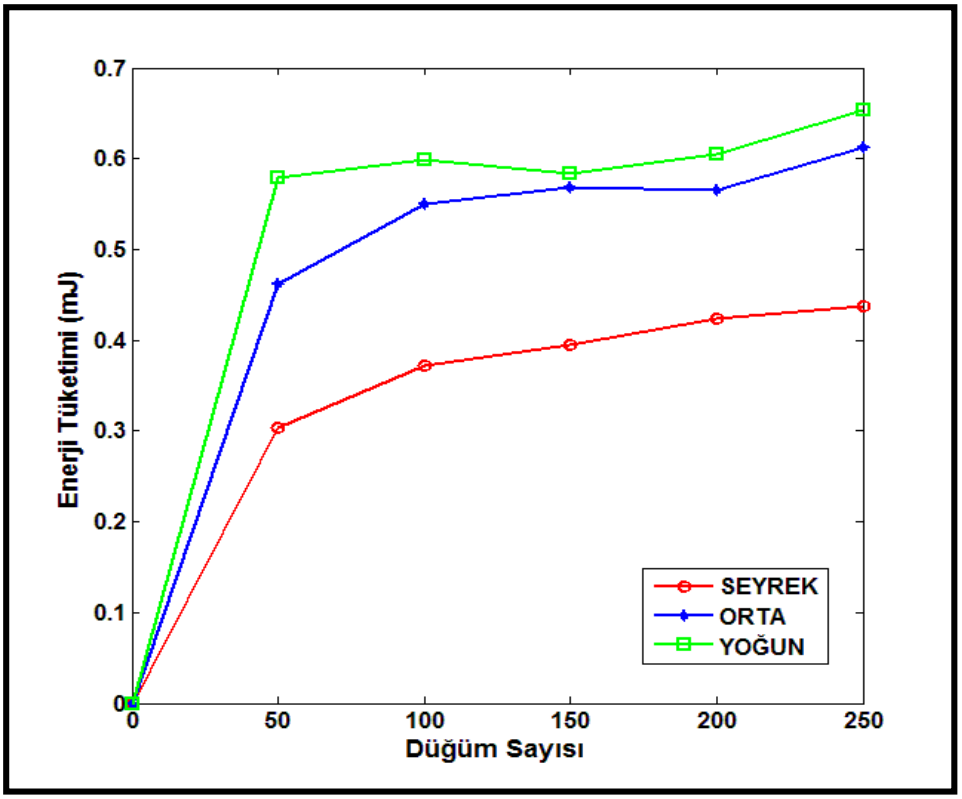
Şekil 4.10 Değişen düğüm sayısına bağlı farklı yoğunluktaki ağlarda gönderilen bayt sayısı

Şekil 4.11'deki grafik değişen düğüm sayılarına bağlı seyrek, orta ve yoğun topolojilerde OZKAN çalıştırıldığında düğümlerin aldığı toplam bayt sayısını göstermektedir. Ağ yoğunluğu arttıkça düğümlerin komşu sayısı artmaktadır. Durumu değişen düğüm, komşularına s_i değerini gönderdiği için komşuluk sayısı arttıkça alınan bayt sayısı da artar. Her ne kadar yoğun ağlarda gönderilen bayt sayısı seyrek ağlara göre daha az olsa da, yoğun ağlarda komşuluk sayısı fazla olduğu için alınan bayt sayısı çok daha fazladır. Yani ağ yoğunluğu arttıkça alınan bayt sayısı artar. Değişen düğüm sayısına baktığımızda ise düğüm sayısı arttıkça alınan bayt sayısı da artmaktadır.

Şekil 4.12'deki grafik değişen düğüm sayılarına bağlı seyrek, orta ve yoğun topolojilerde OZKAN çalıştırıldığında düğümlerin harcadığı ortalama enerji miktarını göstermektedir. Düğümler en fazla enerjiyi iletişim için harcar. Her ne kadar yoğun ağlarda gönderilen bayt sayısı seyrek ağlara göre daha az olsa da, yoğun ağlarda komşuluk sayısı fazla olduğu için alınan bayt sayısı çok daha fazladır. Bu nedenle ağ yoğunluğu arttıkça düğümlerin ortalama enerji tüketimi artmaktadır. Değişen düğüm sayısına baktığımızda ise düğüm sayısı arttıkça enerji tüketimi de genel olarak artmaktadır.



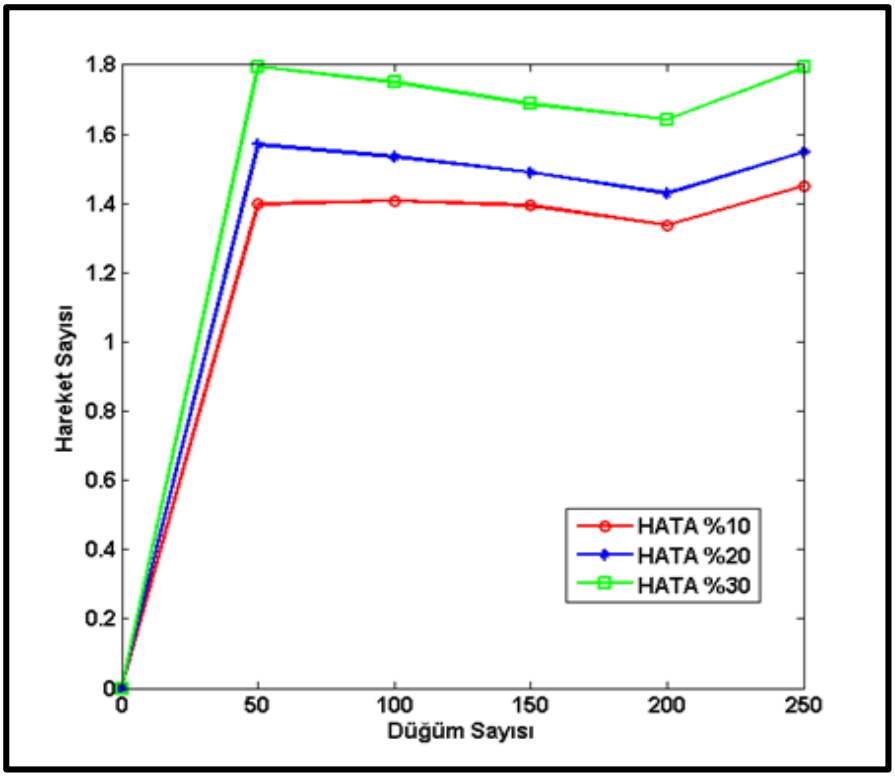
Şekil 4.11 Değişen düğüm sayısına bağlı farklı yoğunluktaki ağlarda alınan bayt sayısı



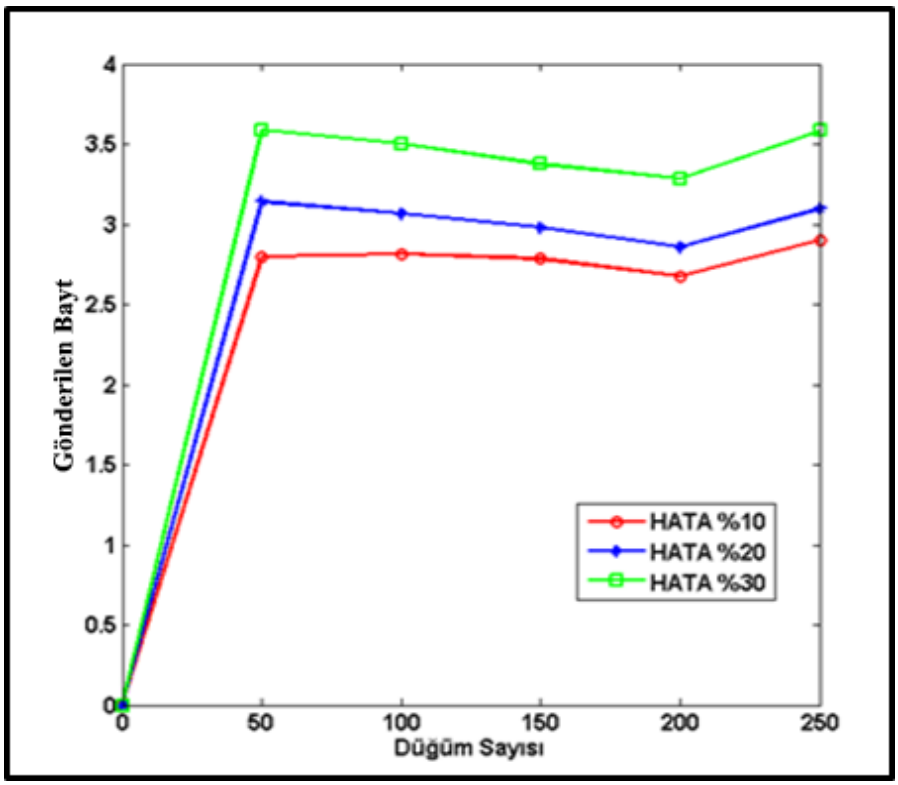
Şekil 4.12 Değişen düğüm sayısına bağlı farklı yoğunluktaki ağlarda tüketilen enerji miktarı

Şekil 4.13'teki grafik değişen düğüm sayılarına bağlı %10, %20 ve %30 hata olan topolojilerde OZKAN çalıştırıldığında yapılan hareket sayılarını göstermektedir. Topolojiler orta yoğunluktadır. Algoritma çalıştırıldıktan sonra farklı turlar içinde belirtilen yüzde oranlarında sistem öz kararlı hale gelmeden düğümler kapatılmıştır. Sistem belirli yüzdelerde çöken düğümlere rağmen öz kararlı hale gelmiştir ve sistemde MBK oluşmuştur. Ağdaki hata oranı arttıkça çöken düğüm sayısı da artmaktadır. Bu yüzden hata oranı düşük ağda çalışan düğüm sayısı daha fazladır ve ağdaki hata yüzdesi arttıkça düğüm başına düşen hareket sayısı artar. Değişen düğüm sayısına baktığımızda ise belirli yüzdelerde hata olan ağda düğüm sayısı arttıkça hareket sayısı da artmaktadır.

Şekil 4.14'deki grafik değişen düğüm sayılarına bağlı %10, %20 ve %30 hata olan topolojilerde OZKAN çalıştırıldığında gönderilen bayt sayılarını göstermektedir. Ağdaki hata yüzdesi arttıkça hareket sayısı arttığı için gönderilen bayt sayısı da artmaktadır. Değişen düğüm sayısına baktığımızda ise belirli yüzdelerde hata olan ağda düğüm sayısı arttıkça gönderilen bayt sayısı da artmaktadır.



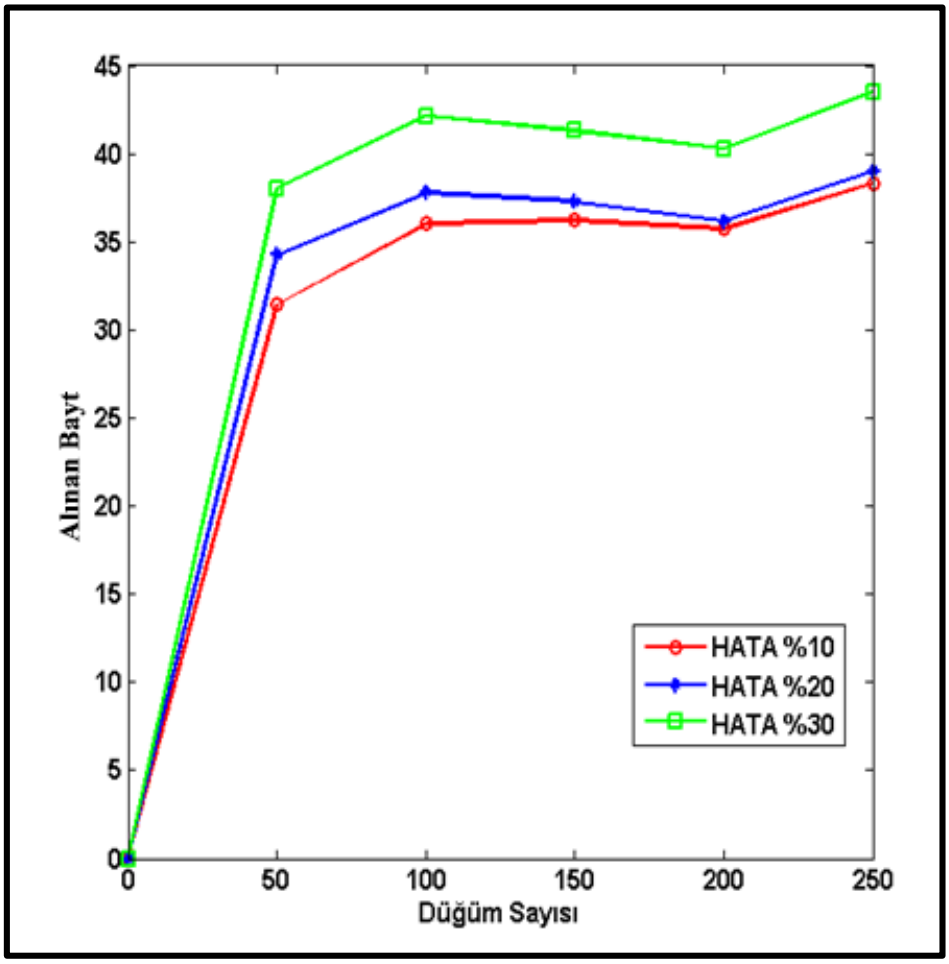
Şekil 4.13 Değişen düğüm sayısına bağlı farklı hata yüzdelerinde hareket sayısı



Şekil 4.14 Değişen düğüm sayısına bağlı farklı hata yüzdelerinde gönderilen bayt sayısı

Şekil 4.15'deki grafik değişen düğüm sayılarına bağlı %10, %20 ve %30 hata olan topolojilerde OZKAN çalıştırıldığında alınan bayt sayılarını göstermektedir. Ağdaki çöken düğüm sayısının yüzdesi arttıkça düğüm başına düşen alınan bayt sayısı da artar. Değişen düğüm sayısına baktığımızda ise belirli yüzdelerde hata olan ağda düğüm sayısı arttıkça alınan bayt sayısı artmaktadır.

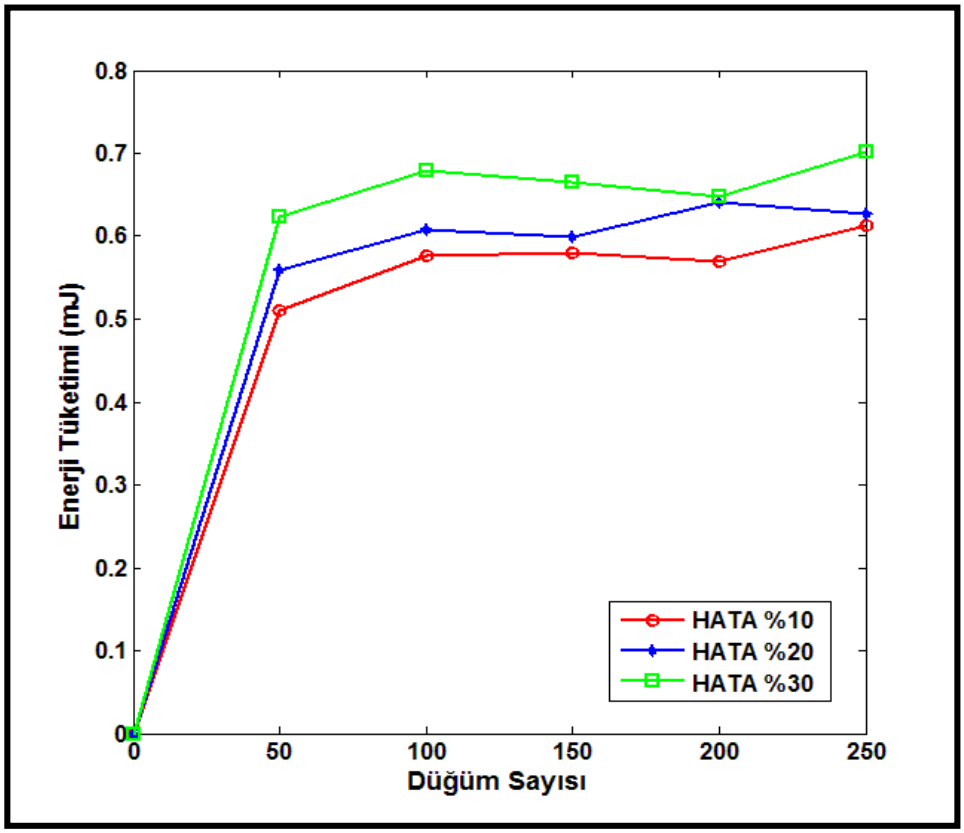
Şekil 4.16'daki grafik değişen düğüm sayılarına bağlı %10, %20 ve %30 hata olan topolojilerde OZKAN çalıştırıldığında düğümlerin enerji tüketimini göstermektedir. Ağdaki çöken düğüm sayısı arttıkça ağdaki düğüm başına düşen hareket sayısı, gönderilen ve alınan bayt sayısı arttığı için düğüm başına düşen enerji miktarı da hata yüzdesi arttıkça artar. Değişen düğüm sayısına baktığımızda ise belirli yüzdelerde hata olan ağda düğüm sayısı arttıkça düğümlerin enerji tüketimi de genel olarak artmaktadır.



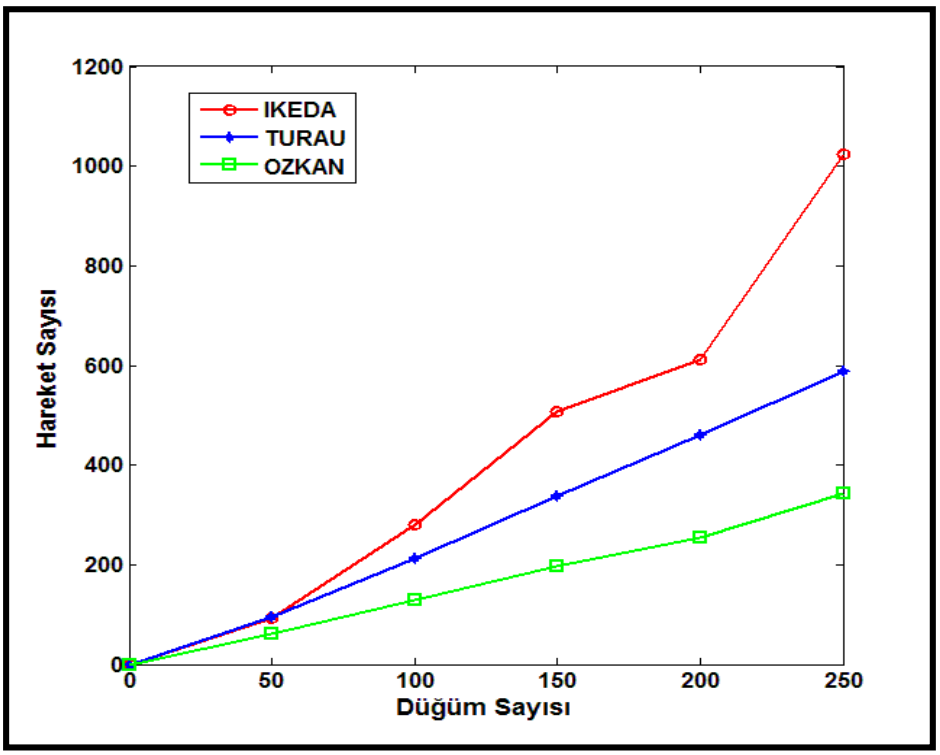
Şekil 4.15 Değişen düğüm sayısına bağlı farklı hata yüzdelerinde alınan bayt sayısı

Şekil 4.17'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayılarına bağlı hareket sayısını göstermektedir. Topolojiler orta yoğunluktadır ve sistemde hata yoktur. Sistemde en fazla hareket yaparak sistemin öz kararlı olmasını sağlayan algoritma IKEDA'dır. TURAU, IKEDA'dan daha az hareket sayısına sahiptir. En az hareket sayısı ile sistemin öz kararlı olmasını sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların hareket sayısı artmaktadır.

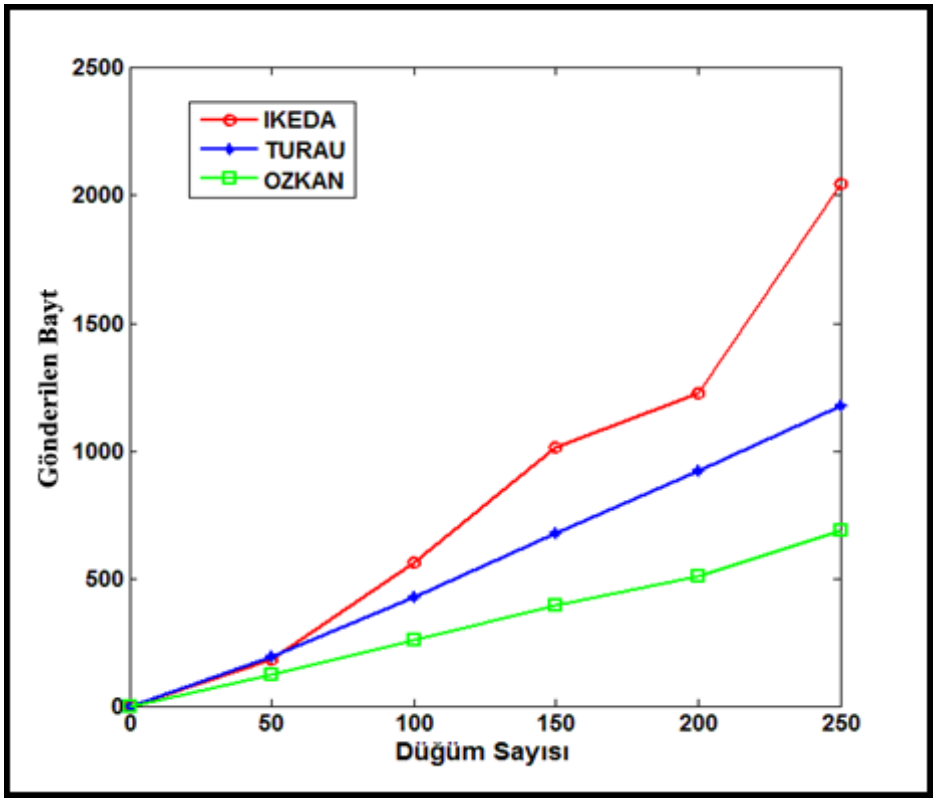
Şekil 4.18'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayılarına bağlı gönderilen toplam bayt sayısını göstermektedir. IKEDA, en fazla hareket sayısına sahip olduğu için sistemin öz kararlı olabilmesi için ağda daha fazla mesaj gönderilmesi gerekir. Bu nedenle IKEDA, en fazla gönderilen bayt sayısına sahip iken, TURAU ondan daha az gönderilen bayt sayısına sahiptir. Sistemde en az bayt gönderilerek sistemin öz kararlı hale gelmesini sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların gönderilen bayt sayısı artmaktadır.



Şekil 4.16 Değişen düğüm sayısına bağlı farklı hata yüzdelerinde düğümlerin enerji tüketimi



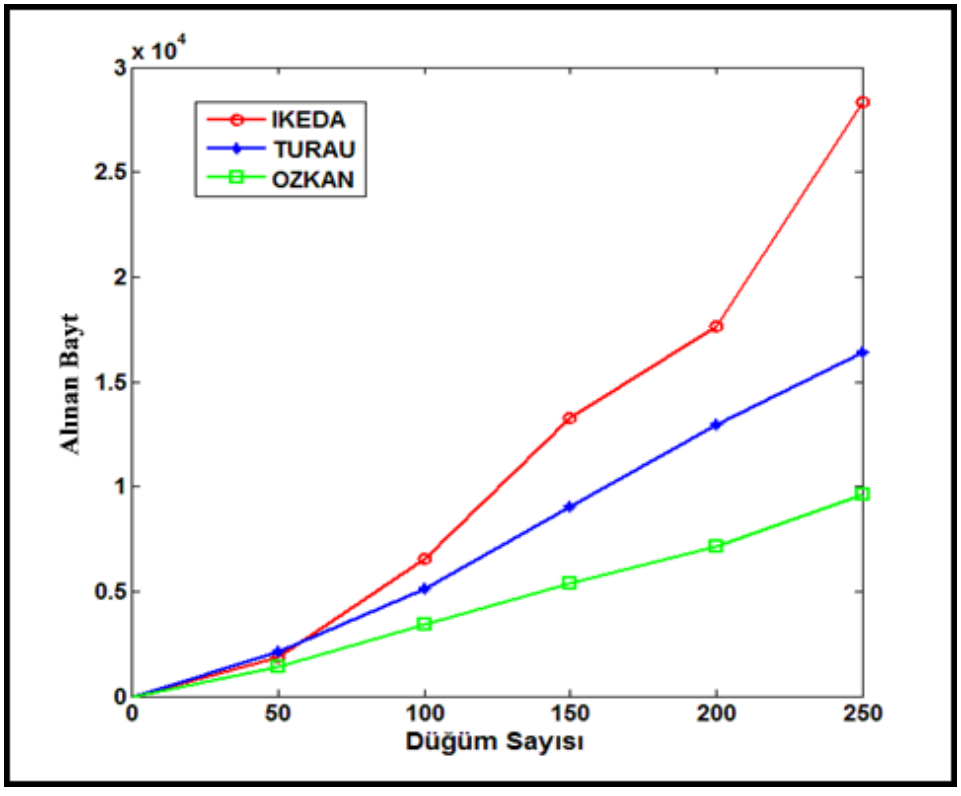
Şekil 4.17 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı hareket sayısı



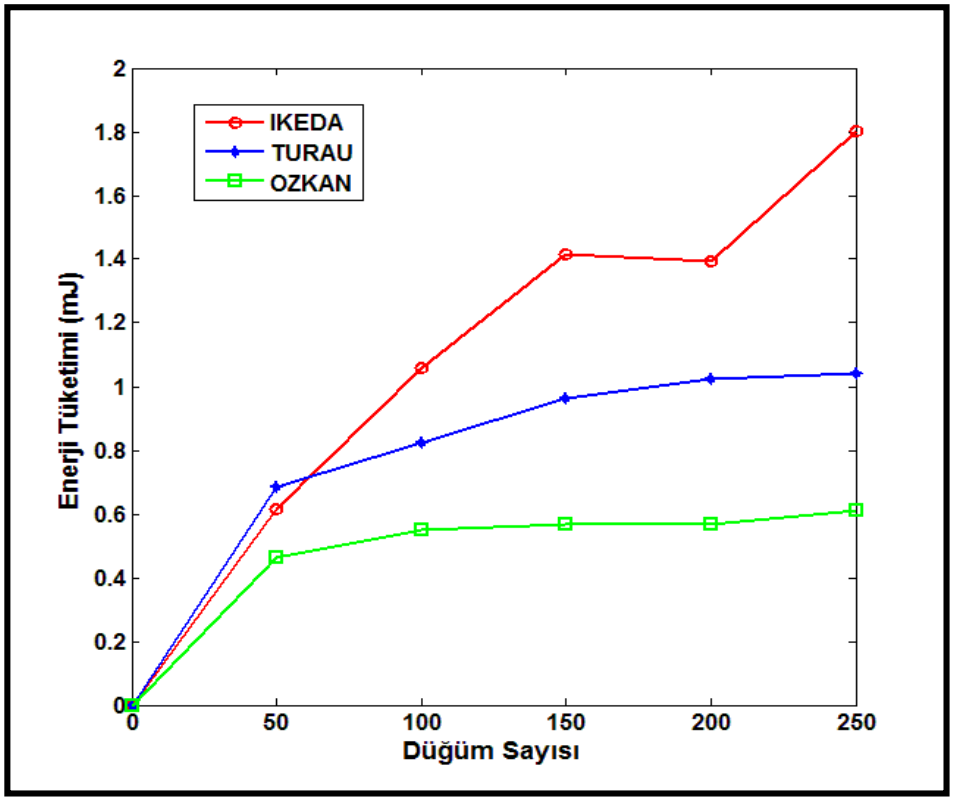
Şekil 4.18 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı gönderilen bayt sayısı

Şekil 4.19'daki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı alınan bayt sayısını göstermektedir. IKEDA, en fazla hareket sayısına sahip olduğu için ağda gönderilen mesaj sayısı fazla olur. Gönderilen mesajlar düğümlerin komşuları tarafından alındığı için alınan mesaj sayısı da fazladır. Bu nedenle IKEDA, en fazla alınan bayt sayısına sahip iken, TURAU ondan daha az alınan bayt sayısına sahiptir. Sistemde en az bayt alınarak sistemin öz kararlı hale gelmesini sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların çalıştırıldığı ağda alınan bayt sayısı da artmaktadır.

Şekil 4.20'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı enerji tüketimini göstermektedir. Düğümler en fazla enerjiyi mesaj gönderirken ve mesaj alırken harcadıkları için en fazla mesaj gönderilen ve alınan algoritma IKEDA olduğu için en fazla enerji tüketimini IKEDA yapmıştır. Daha sonra TURAU gelmektedir. En az hareket yapan, en az mesaj gönderen ve en az mesaj alan algoritma OZKAN olduğu için OZKAN'ın enerji tüketimi de en az olmuştur. Düğüm sayısı arttıkça bütün algoritmaların enerji tüketimi de artmaktadır.



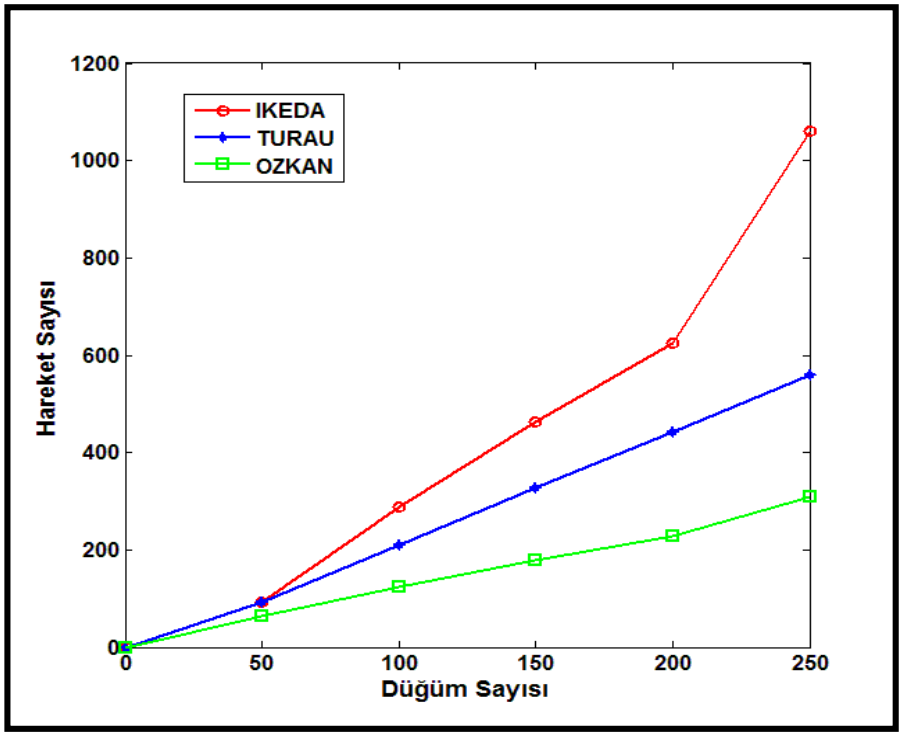
Şekil 4.19 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı alınan bayt sayısı



Şekil 4.20 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı enerji tüketimi

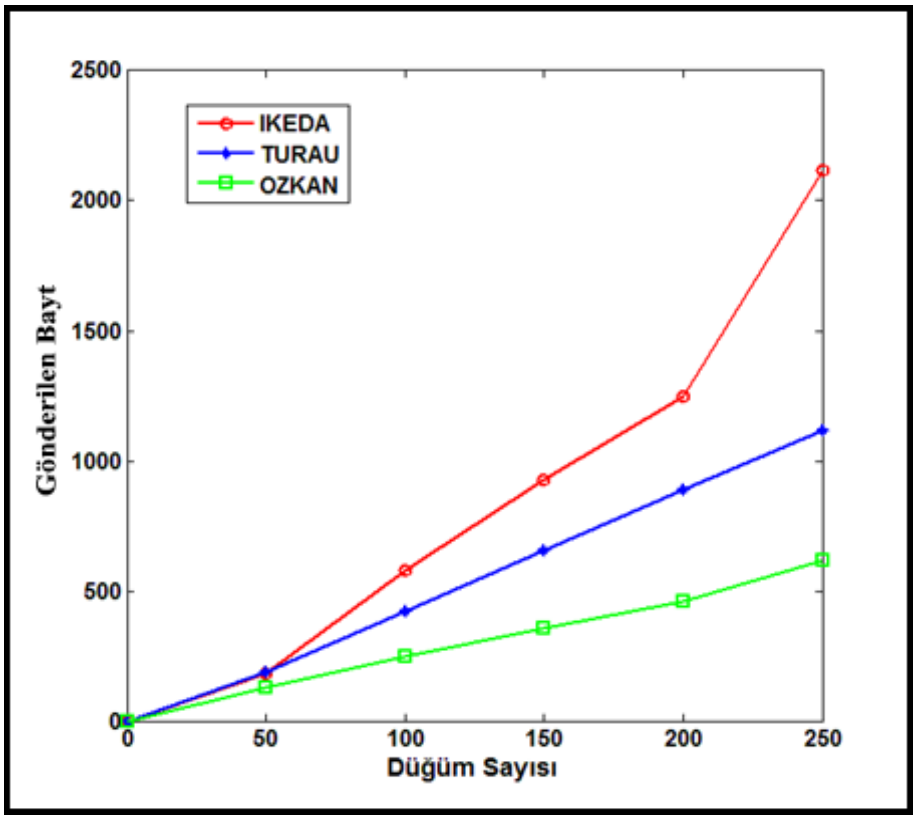
Şekil 4.21'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı hareket sayısını göstermektedir. Topolojiler orta yoğunluktadır fakat; sistemde %20 hata vardır. Ağdaki düğümlerin %20'si farklı turlar içinde sistem öz kararlı hale gelmeden çökmüştür. Sistemde en fazla hareket yaparak sistemin öz kararlı olmasını sağlayan algoritma IKEDA'dır. TURAU, IKEDA'dan daha az hareket sayısına sahiptir. En az hareket sayısı ile ağda %20 hata olmasına rağmen sistemin öz kararlı olmasını sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların hareket sayısı artmaktadır.

Şekil 4.22'deki IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına gönderilen bayt sayısını göstermektedir. Topolojiler orta yoğunluktadır fakat; sistemde %20 hata vardır. Ağdaki düğümlerin %20'si farklı turlar içinde sistem öz kararlı hale gelmeden çökmüştür. IKEDA, en fazla gönderilen bayt sayısına sahip iken, TURAU ondan daha az gönderilen bayt sayısına sahiptir. Sistemde en az bayt gönderilerek sistemin öz kararlı hale gelmesini sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların çalıştırıldığı ağdaki gönderilen bayt sayısı da artmaktadır.



Şekil 4.21 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı hareket sayısı

(Ağdaki hata oranı %20)



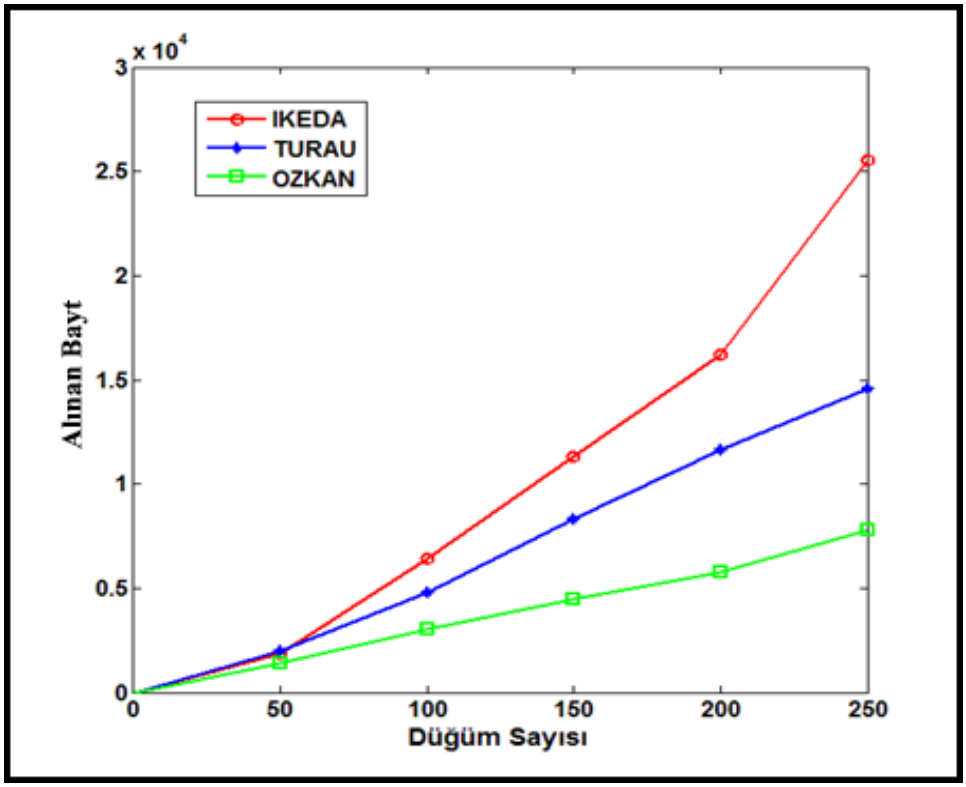
Şekil 4.22 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı gönerilen bayt sayısı

(Ağdaki hata oranı %20)

Şekil 4.23'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı alınan bayt sayısını göstermektedir. Topolojiler orta yoğunluktadır fakat; sistemde %20 hata vardır. Ağdaki düğümlerin %20'si farklı turlar içinde sistem öz kararlı hale gelmeden çökmüştür. IKEDA, en fazla hareket sayısına sahip olduğu için ağda gönderilen mesaj sayısı da fazladır. Gönderilen mesajlar düğümlerin komşuları tarafından alındığı için alınan mesaj sayısı da fazladır. Bu nedenle IKEDA, en fazla alınan bayt sayısına sahip iken, TURAU'da alınan bayt sayısı daha azdır. Sistemde en az bayt alınarak sistemin öz kararlı hale gelmesini sağlayan algoritma ise OZKAN'dır. Ağdaki düğüm sayısı arttıkça bütün algoritmaların çalıştırıldığı ağdaki alınan bayt sayısı da artmaktadır.

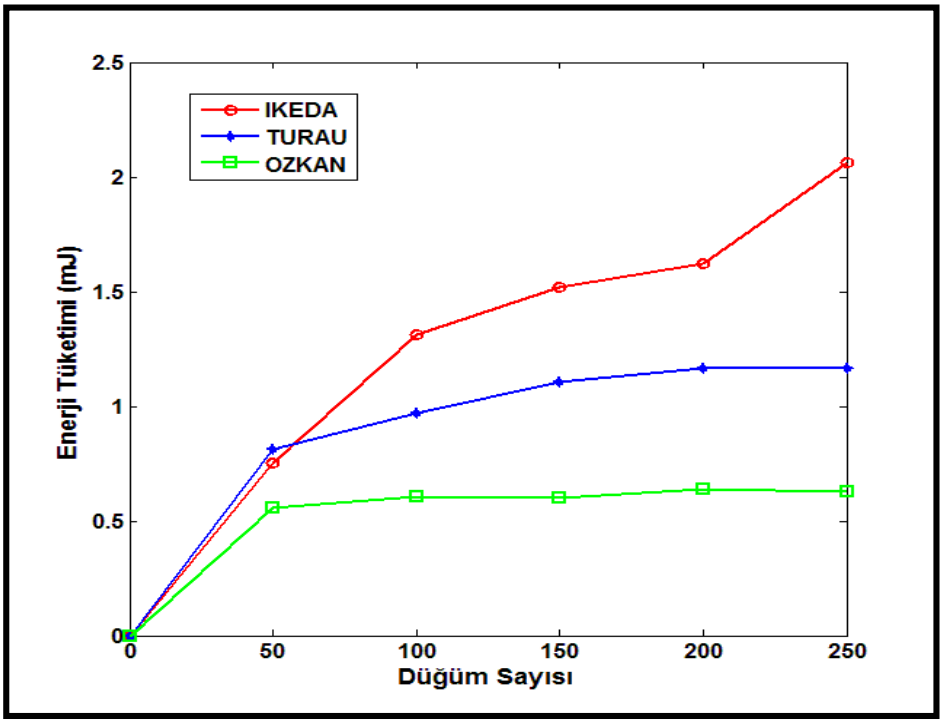
Şekil 4.24'deki grafik IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı enerji tüketimini göstermektedir. Topolojiler orta yoğunluktadır fakat; sistemde %20 hata vardır. Ağdaki düğümlerin %20'si farklı turlar içinde sistem öz kararlı hale gelmeden çökmüştür. Düğümler en fazla enerjiyi mesaj gönderirken ve alırken harcadıkları için en fazla mesaj gönderilen ve alınan algoritma IKEDA olduğu için en fazla enerji tüketimini IKEDA yapmıştır. Daha sonra TURAU gelmektedir. En az hareket yapan, en az mesaj gönderen ve en az mesaj alan algoritma OZKAN olduğu için OZKAN'ın enerji tüketimi de en az olmuştur. Düğüm sayısı arttıkça bütün algoritmaların enerji tüketimi de artmaktadır.

Genel olarak simülasyon sonuçları incelendiğinde OZKAN algoritmasının gönderilen bayt sayısı, alınan bayt sayısı, enerji tüketimi, hareket sayısı değerleri değişen düğüm sayılarına, ağ bağlantı şekillerine ve hata oranlarına göre ölçeklenebilir ve etkin olarak ölçülmüştür. TURAU algoritması IKEDA'dan daha iyi sonuçlar vermiştir, en iyi sonuçları OZKAN üretmiştir.



Şekil 4.23 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı alınan bayt sayısı

(Ağdaki hata oranı %20)



Şekil 4.24 IKEDA, TURAU ve OZKAN'ın değişen düğüm sayısına bağlı enerji tüketimi

(Ağdaki hata oranı %20)

5. SONUÇLAR

Bu tezde dağıtık öz kararlı MBK algoritmaları üzerine çalışılmıştır. Literatürdeki algoritmalar teorik olarak analiz edilmiştir. IKEDA'nın 2 durumlu MBK için en kötü durumda en az hareket kullandığı ispatlanmıştır. En az 3 durum kullanan bir algoritmanın en az 2n-1 hareket yapması gerektiği ispatlanmıştır. Bu tez kapsamında OZKAN algoritması önerilmiştir. Önerilen algoritmanın doğruluk ispatı yapılmıştır. OZKAN algoritması ile sistem en fazla 2n-1 hareket yaparak öz kararlı hale gelmektedir. Bu değerin en iyi değer olduğu bu tez kapsamında ispatlanmıştır. Önerilen algoritma gerçek uygulamalar ve benzetimlerle literatürdeki algoritmalarla karşılaştırıldı.

Bu tez kapsamında MBK algoritmaları Iris düğümleri üzerinde uygulanmış ve TOSSIM benzetim ortamında gerçeklenmiştir. Bu çalışmalarda değişen düğüm sayısı, çeşitli ağ bağlantı şekilleri ve çeşitli hata yüzdelerine bağlı olarak çalışmalar yapılmıştır. Teorik çalışmanın yanında uygulama ve benzetim sonuçları sistemin öz kararlı hale gelmesi ve MBK'yı oluşturması için en az hareket sayısına sahip olan algoritmanın OZKAN olduğunu göstermiştir. Bunun yanı sıra en az gönderilen ve alınan bayt sayısında da OZKAN en etkin algoritma olmuştur. Ağdaki düğümler en fazla enerjiyi iletişim için harcadıklarından dolayı sistemin öz kararlı hale gelmesi ve MBK'nın oluşması için düğümlerin enerji tüketimi en az OZKAN'da olmuştur. Hata olmayan sistemin yanı sıra belirli yüzde oranlarında hata olan sistemlerde de algoritmalar karşılaştırıldı. Test sonuçları OZKAN'ın hata toleransının literatürdeki dağıtık öz kararlı MBK algoritmalarından daha etkin olduğunu göstermiştir. Tüketilen enerjinin en az olması ve hata toleransının yüksek olması ile OZKAN ağın yaşam süresini en fazla artıran algoritma olduğunu göstermiştir. Sonuç olarak OZKAN literatürdeki algoritmalar içinde en etkin çalışan dağıtık öz kararlı MBK algoritmasıdır. Gelecek çalışmalar olarak senkron ortamda MBK oluşturma, sinyalin kapsama gücüne bağlı MBK oluşturma ve bağlı hakim küme üzerinde MBK oluşturma konuları üzerinde çalışmayı planlamaktayız.

KAYNAKLAR DİZİNİ

Abbasi, A.A. and Younis, M., 2007, A survey on clustering algorithms for wireless sensor networks, *Computer Communications*, 30:2826-2841 pp.

Beaquier, J., Datta, A.K., Gradinariu, M. and Magniette, F., 2002, Self-stabilizing local mutual exclusion and daemon refinement, *Chicago Journal of Theoretical Computer Science*

Dijkstra, E.W., 1974, Self-stabilizing systems in spite of distributed control, *Communications of the ACM*, 17(11):643-644 pp.

Dolev, S., 2000, Self-stabilization, MIS Press, 207 p.

Erciyes, K., 2013, Distributed Graph Algorithms for Computer Networks, *Springer-Verlag*, London, 324 p.

Goddard, W., Hedetniemi S.T., Jacobs D.P. and Srimani P.K., 2003, Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks, *In Proceedings of the 5th International Parallel and Distributed Processing Symposium (*IPDPS)

Gradinariu, M. and Tixeuil, S., 2000, Self-stabilizing vertex coloring of arbitrary graphs, *In Proceedings of the 2nd Workshop on Self-Stabilizing Systems (WSS99)*, 48-53 pp.

Guellati, N. and Kheddouci, H., 2010, A survey on self-stabilizing algorithms for independence, domination, coloring, and mathing in graphs, *Journal of Parallel and Distributed Computing*, 70:406-415 pp.

Hedetniemi, S.M., Hedetniemi, S.T., Jacobs, D.P. and Srimani, P.K., 2003, Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, *Computer Mathematics and Applications*, 46(5-6):805-811 pp.

http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf, Erişim Tarihi: 11.18.2014

http://www.youtube.com/watch?v=VytqV07mbss, Erişim Tarihi : 19.12.2014

Ikeda, M., Kamei, S. and Kakugawa, H., 2002, A space-optimal self-stabilizing algorithm for the maximal independent set problem, *In Proceedings of the 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 70-74 pp.

KAYNAKLAR DİZİNİ(devam)

Karl, H. And Willig, A., 2005, Protocols and Architectures for Wireless Sensor Networks, *John Wiley & Sons, England*, 526 p

Lagemann, L., Nolte, J., Weyer, C. And Turau, V., 2009, Mission Statement: Applying Self-Stabilization to Wireless Sensor Networks, In *Proceedings of the 8th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'09), Hamburg, 47-49 pp.*

Li, Y., Thai, T. and Wu,W., 2008, Wireless Sensor Networks and Applications, *Springer Science & Business Media,* New York, 464 p.

Schneider, M., 1993, Self-stabilization, *ACM Computing Surveys,* 25(1):45-67

Shukla, S.K., Rosenkrantz, D.J. and Ravi, S.S., 1995, Observations on self-stabilizing graph algorithms for anonymous networks, *In Proceedings of the 2nd Workshop on Self-Stabilizing Systems*

Sohraby, K., Minoli, D. and Znati, T., 2007, Wireless Sensor Networks: technology, protocols, and applications, *John Wiley & Sons*, New Jersey, 326 p.

Turau, V. and Weyer, C., 2006, Randomized self-stabilizing algorithms for wireless sensor networks, *In Proceedings of the International Workshop on Self Stabilizing Systems,* 74-89 pp.

Turau, V., 2007, Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler, *Information Processing Letters,* 103(3):88-93 pp.

Wu, Y., Du, H., Jia, X., Li, Y. and Huang, S.C.-H., 2005, Minumum connected dominating sets and maximal independent sets in unit disk graphs.

Wu, Y., Wang, F., Thai, T. and Li, Y., 2007, Constructing k-Connected m-Dominating Sets in Wireless Sensor Networks, *Millitary Communications Conference*, USA, 1-7 pp.

ÖZGEÇMİŞ

Türkiye Cumhuriyeti vatandaşı olan Özkan ARAPOĞLU, 1985 yılında Çorum'da dünyaya gelmiştir. Lise eğitimini Çorum Anadolu Lisesi'nde 2004 yılında tamamladıktan sonra, lisans eğitimine Balıkesir Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi bölümünde 2004 yılında başlamıştır. 2008 yılında lisans eğitimini tamamladıktan sonra Milli Eğitim Bakanlığı'nda Bilişim Teknolojileri öğretmeni olarak görevine başlamış ve halen öğretmenliğe devam etmektedir.

Araştırma alanları web tasarım ve yazılım, dağıtık algoritmalar ve sistemler, telsiz duyarga ağları ve yapay zekadır.

EKLER

EK1 TOSSIM Uygulama Kodları

EK 1

TOSSIM Uygulama Kodları

Starter Node

misP.nc

```
#include <Timer.h>
        #include "mis.h"
        #define MSGTYPE 13   // use 13 for IKEDA, 23 for TURAU, 33 for OZKAN
        module misP {
          uses {
             interface Boot;
             interface SplitControl;
             interface AMSend;
             interface Packet;
             interface AMPacket;
             interface Receive;
             interface Leds;
             interface PacketAcknowledgements;
             interface Timer<TMilli> as Timer0;
          }
        }
implementation{
           message_t myMsg;
           uint16_t sentNumber=0;

              /****************** Prototypes ****************/
           void sendBroadcast();

            /****************** Boot Events ****************/
           event void Boot.booted() {
             call SplitControl.start();
           }
             /****************** SplitControl Events ****************/
           event void SplitControl.startDone(error_t error) {
                        call Timer0.startPeriodic(10);
          }
           event void SplitControl.stopDone(error_t error) {
           }
```

```
  /****************** Receive Events ****************/
  event message_t *Receive.receive(message_t *msg, void *payload, uint8_t len) {
        return msg;}

  /****************** AMSend Events ****************/
    event void AMSend.sendDone(message_t *msg, error_t error) {}

  /****************** Tasks ****************/
    void sendBroadcast() {
          newMsg_t* btrpkt =(newMsg_t*)(call Packet.getPayload(&myMsg, sizeof(newMsg_t)));
          if (btrpkt == NULL) {
                return;
          }
          btrpkt->nodeId= TOS_NODE_ID;
          btrpkt->state= MSGTYPE;
          btrpkt->receive=0;
          if(call AMSend.send(AM_BROADCAST_ADDR, &myMsg, sizeof(newMsg_t))==SUCCESS) {}
     }
     event void Timer0.fired(){
          sentNumber++;
          sendBroadcast();
          call Leds.led0Toggle();
          if(sentNumber==2)
          call Timer0.stop();
     }
  }
```

misC.nc

```
#include <Timer.h>
#include "mis.h"
configuration misC {
}
implementation {
  components misP,
     MainC,
     ActiveMessageC,
     new TimerMilliC() as Timer0,
     new AMSenderC(AmId),
     new AMReceiverC(AmId),
     LedsC;
```

```
        misP.Boot -> MainC;
        misP.SplitControl -> ActiveMessageC;
        misP.Leds -> LedsC;
        misP.AMPacket -> AMSenderC;
        misP.Packet -> AMSenderC;
        misP.AMSend -> AMSenderC;
        misP.Receive -> AMReceiverC;
        misP.PacketAcknowledgements -> ActiveMessageC;
        misP.Timer0 -> Timer0;
}
```

mis.h

```
#ifndef mis_H
#define mis_H
enum {
        AmId =6,
};
typedef nx_struct newMsg{
        nx_uint8_t nodeId;
        nx_uint8_t state;
} newMsg_t;
#endif
```

Running Nodes

misP.nc

```
#include <Timer.h>
#include "mis.h"

#define NODESIZE 18
#define IN 1
#define OUT 2
#define WAIT 3

module misP {
  uses {
        interface Boot;
        interface SplitControl;
        interface AMSend;
        interface Packet;
        interface AMPacket;
```

```
    interface Receive;
    interface Leds;
    interface PacketAcknowledgements;
        interface Timer<TMilli> as Timer0;
        interface Timer<TMilli> as Timer1;
        interface Timer<TMilli> as Timer2;
        interface PacketField<uint8_t> as PacketRSSI;
  }
}
implementation{
  message_t myMsg;
  uint8_t myState, state, nodeId, neighs[NODESIZE], states[NODESIZE], i=0, msgType=0,
myType=0, changed=0, rcv=0;

  bool chkIN,chkMinIN,chkWAIT,chkMinWAIT,chkMinOUT,chkNeighbor;

  /****************** Prototypes ****************/
  void sendBroadcast();
  void ikedaState();
  void turauState();
  void ozkanState();
  void startState();

  /****************** Boot Events ****************/
  event void Boot.booted() {
    call SplitControl.start();
  }

  /****************** SplitControl Events ****************/
  event void SplitControl.startDone(error_t error) {}

  event void SplitControl.stopDone(error_t error) {
        call SplitControl.start();
  }

  /****************** Receive Events ****************/
  event message_t *Receive.receive(message_t *msg, void *payload, uint8_t len) {
        newMsg_t* btrpkt = (newMsg_t*)payload;
        uint16_t rssiValue=0,j=0;
        bool myNeighbor=FALSE;
```

```
rssiValue=call PacketRSSI.get(msg);
nodeId=btrpkt->nodeId;
state=btrpkt->state;
if(btrpkt->nodeId==0){

        call Leds.led2Toggle();
        call Leds.led0Off();
        call Leds.led1Off();
        call Leds.led2Off();

        if(call Timer1.isRunning())
        {
                call Timer1.stop();
        }else if(call Timer2.isRunning())
        {
                call Timer2.stop();
        }

        for(j=0;j<NODESIZE;j++)
        {
                states[j]=0;
                neighs[j]=0;
        }
        i=0;
        rcv=0;
        msgType=state;
        startState();
        call Timer0.startOneShot(TOS_NODE_ID*250);
        call Timer1.startPeriodic((NODESIZE+1)*250);

}else if(rssiValue<=12){

        return msg;
}else if(state<4){

        myNeighbor=FALSE;

                for(j=0;states[j];j++){
                        if(neighs[j]==nodeId)
                        {
```

```
                                        myNeighbor=TRUE;
                                        break;
                                }
                        }

                        if(myNeighbor)
                        {
                                states[j]=state;

                        }else{
                                neighs[i]=nodeId;
                                states[i]=state;
                                i++;
                        }
          }
        return msg;
 }
  /****************** AMSend Events ****************/
  event void AMSend.sendDone(message_t *msg, error_t error) {
        }

 /****************** Timer Events ****************/
  event void Timer0.fired() {
           changed=1;
           sendBroadcast();
           call Timer2.startPeriodic((NODESIZE+1)*250);
  }
   event void Timer1.fired() {
         int k=0;
         changed=0;
         if(k=1) rcv=i;
         if(myType==1)
           {          ikedaState();
           }else if(myType==2){
                   turauState();
           }else{
                   ozkanState();
          }
   k++;
}
```

```
  event void Timer2.fired() {
          if(changed){ sendBroadcast();
        }
}

 /****************** Tasks *****************/
    void sendBroadcast() {
          newMsg_t* btrpkt =(newMsg_t*)(call Packet.getPayload(&myMsg, sizeof(newMsg_t)));
           if (btrpkt == NULL) {
                        return;
           }
           btrpkt->nodeId= TOS_NODE_ID;
           btrpkt->state= myState;
           btrpkt->receive=rcv;
           if (call AMSend.send(AM_BROADCAST_ADDR, &myMsg,
sizeof(newMsg_t))==SUCCESS) {
                    }
  }
  void startState(){
                    if(msgType>=10 && msgType<20)
                    {
                        myType=1;
                    }else if(msgType>=20 && msgType<30){
                        myType=2;
                    }else {
                        myType=3;
                    }
                    if((msgType-3) % 10==0 && TOS_NODE_ID>(NODESIZE/2))
                    {
                        myState=OUT;
                        call Leds.led1On();
                    }else if((msgType-2) % 10==0) {

                        myState=OUT;
                        call Leds.led1On();
                    }else{
                        myState=IN;
                        call Leds.led0On();
                    }
}
```

```
//IKEDA Algorithm
 void ikedaState()
        {
                int j;
                chkIN=FALSE;
                for(j=0;states[j];j++)
                {
                        if(states[j]==IN)
                        {
                                if(myState==IN &&  neighs[j]<TOS_NODE_ID)
                                {
                                        myState=OUT; //Rule 1
                                        changed=1;
                                        call Leds.led0Off();
                                        call Leds.led1On();
                                }
                                chkIN=TRUE;
                        }
                }
                if(myState==OUT && !chkIN)
                {
                        myState=IN; //Rule 2
                        changed=1;
                        call Leds.led1Off();
                        call Leds.led0On();
                }
        }

//Turau   Algorithm
void turauState()
        {
                int j;
                chkIN=FALSE;
                chkMinIN=FALSE;
                chkMinWAIT=FALSE;

                for(j=0;states[j];j++)
                {
                        if(states[j]==IN)
                        {
```

```
                if(myState==IN && neighs[j]<TOS_NODE_ID)
                {
                        chkMinIN=TRUE;
                }
                chkIN=TRUE;
        }
        if(states[j]==WAIT)
        {
                if(myState==WAIT &&  neighs[j]<TOS_NODE_ID)
                {
                        chkMinWAIT=TRUE;
                }
        }
}

if(myState==OUT && !chkIN)
{
        myState=WAIT; //Rule1
        changed=1;
        call Leds.led0Off();
        call Leds.led1Off();
        call Leds.led2On();

}else if(myState==WAIT && chkIN)
{
        myState=OUT; //Rule2
        changed=1;
        call Leds.led0Off();
        call Leds.led2Off();
        call Leds.led1On();

}else if(myState==WAIT && !chkIN && !chkMinWAIT)
{
        myState=IN; //Rule3
        changed=1;
        call Leds.led1Off();
        call Leds.led2Off();
        call Leds.led0On();

}else if(myState==IN && chkMinIN)
```

```
		{
			myState=OUT; //Rule 4
			changed=1;
			call Leds.led0Off();
			call Leds.led1On();
		}
}
//OZKAN Algorithm
void ozkanState()
{
		int j;
		chkIN=FALSE;
		chkMinIN=FALSE;
		chkMinWAIT=FALSE;
		chkMinOUT=FALSE;
		chkWAIT=FALSE;

		for(j=0;states[j];j++)
		{
			if(states[j]==IN)
			{
				if(myState==IN && neighs[j]<TOS_NODE_ID)
				{
					chkMinIN=TRUE;
				}
				chkIN=TRUE;}

			if(states[j]==WAIT)
			{
				if(myState==WAIT && neighs[j]<TOS_NODE_ID)
				{
					chkMinWAIT=TRUE;
				}

				chkWAIT=TRUE;
			}

			if(states[j]==OUT)
			{
				if(myState==OUT &&  neighs[j]<TOS_NODE_ID)
```

```
                {
                        chkMinOUT=TRUE;
                }
        }
}

if(myState==OUT && !chkIN && !chkWAIT && !chkMinOUT)
{
        myState=IN; //Rule 1
        changed=1;
        call Leds.led2Off();
        call Leds.led1Off();
        call Leds.led0On();

}else if(myState==OUT && !chkIN && !chkWAIT && chkMinOUT)
{
        myState=WAIT; //Rule2
        changed=1;
        call Leds.led0Off();
        call Leds.led1Off();
        call Leds.led2On();

}else if(myState==WAIT && chkIN)
{
        myState=OUT; //Rule3
        changed=1;
        call Leds.led0Off();
        call Leds.led2Off();
        call Leds.led1On();

}else if(myState==WAIT && !chkIN && !chkMinWAIT)
{
        myState=IN; //Rule4
        changed=1;
        call Leds.led1Off();
        call Leds.led2Off();
        call Leds.led0On();

}else if(myState==IN && chkMinIN)
{
```

```
                    myState=WAIT;//Rule5
                    changed=1;
                    call Leds.led1Off();
                    call Leds.led0Off();
                    call Leds.led2On();
                }
        }
}
```

misC.nc

```
#include <Timer.h>
#include "mis.h"
configuration misC {
}

implementation {
  components misP,
     MainC,
     ActiveMessageC,
     new AMSenderC(AmId),
     new AMReceiverC(AmId),
          new TimerMilliC() as Timer0,
          new TimerMilliC() as Timer1,
          new TimerMilliC() as Timer2,
          RF230ActiveMessageC,
     LedsC;

  misP.Boot -> MainC;
  misP.SplitControl -> ActiveMessageC;
  misP.Leds -> LedsC;
  misP.AMPacket -> AMSenderC;
  misP.Packet -> AMSenderC;
  misP.AMSend -> AMSenderC;
  misP.Receive -> AMReceiverC;
  misP.PacketAcknowledgements -> ActiveMessageC;
  misP.PacketRSSI -> RF230ActiveMessageC.PacketRSSI;
  misP.Timer0 -> Timer0;
  misP.Timer1 -> Timer1;
  misP.Timer2 -> Timer2;

}
```

mis.h

```
#ifndef mis_H
#define mis_H
enum {
	AmId =6,
};
typedef nx_struct newMsg{
	nx_uint8_t nodeId;
	nx_uint8_t state;
} newMsg_t;
#endif
```