

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Санкт-Петербургский государственный университет
телекоммуникаций им. проф. М.А. Бонч-Бруевича»

На правах рукописи



Аль-Кадами Нассер Ахмед Салех

ИССЛЕДОВАНИЕ АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ В БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЯХ

Специальность 05.12.13 – Системы, сети и устройства телекоммуникаций

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор **А.Е. Кучерявый**

Санкт-Петербург – 2016

Содержание	
ВВЕДЕНИЕ.....	5
1. АНАЛИЗ СУЩЕСТВУЮЩИХ РАБОТ В ПРЕДМЕТНОЙ ОБЛАСТИ ДИССЕРТАЦИИ	15
1.1. Архитектура беспроводных сенсорных сетей	16
1.2. Приложения беспроводных сенсорных сетей.....	18
1.3. Характеристики архитектуры и классификация беспроводных сенсорных сетей.....	20
1.4. Маршрутизация в беспроводных сенсорных сетях.....	22
1.5. Проблемы планирования и маршрутизации в беспроводных сенсорных сетях	24
1.6. Гомогенные и гетерогенные БСС.....	28
1.6.1. Типы гетерогенных ресурсов.....	28
1.6.2. Влияние неоднородности сенсорных узлов на БСС	28
1.7. Кластеризация БСС.....	30
1.7.1 Задачи диссертационной работы.....	32
Выводы	34
2. ОЦЕНКА И СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ МАРШРУТИЗАЦИИ ДЛЯ ГОМОГЕННЫХ И ГЕТЕРОГЕННЫХ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ.....	36
2.1 Обзор исследований в области БСС.	36
2.2. Классификация алгоритмов маршрутизации в БСС.	37
2.2.1 Алгоритмы маршрутизации на основе сетевой структуры.	38
2.3. Сравнительный анализ алгоритмов <i>DT</i> , <i>LEACH</i> , <i>SEP</i> , <i>DEEC</i> , <i>TEEN</i>	47
2.3.1. Первый сценарий.	49
2.3.2 Второй сценарий.....	53

2.3.3. Третий сценарий.	56
Выводы	59
3. АДАПТИВНЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ ДЛЯ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ С МОБИЛЬНЫМИ УЗЛАМИ ...	61
3.1. Мобильные Беспроводные Сенсорные Сети (<i>MWSN</i>).	61
3.1.1. Различные случаи мобильности.....	64
3.1.2. Модели мобильности.....	65
3.2. Адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами (<i>MACA</i>)	65
3.2.1. Модель для исследования алгоритма <i>MACA</i>	66
3.2.2. Комбинированный критерий прогнозирования.....	67
3.2.3. Алгоритм <i>MACA</i>	68
3.3. Результаты моделирования.	71
3.3.1. Длительность жизненного цикла сети.	72
3.3.2. Период стабильности.....	74
3.3.3. Число успешно переданных пакетов.	75
Выводы.	76
4. ОТКАЗОУСТОЙЧИВЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ ДЛЯ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ.....	77
4.1. Обзор исследований в предметной области.....	77
4.2. Алгоритм <i>TEEN</i>	79
4.3. Алгоритм <i>FT-TEEN</i>	81
4.3.1. Модель для исследования алгоритма <i>FT-TEEN</i>	82
4.3.2. Фаза передачи данных.....	85
4.3.3. Обнаружение и восстановление ошибок.	85
4.4. Результаты моделирования.....	87
Выводы	92
5. ПОКРЫТИЕ, СВЯЗНОСТЬ И ПЛОТНОСТЬ В ДВУМЕРНЫХ И ТРЕХМЕРНЫХ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЯХ.....	94
5.1. Обзор исследований в предметной области.....	94

5.2. Стратегии расположения БСУ в двумерном и трехмерном пространствах.....	96
5.3. Соотношение между радиусом покрытия сенсорного узла, долей покрытия и плотностью в $2D$ и $3D$ БСС.....	97
5.3.1. Двумерная БСС	98
5.3.2. Трехмерная БСС.....	99
5.4. Связность сети.....	104
5.5. Соотношения между радиусом покрытия и дальностью связи.	105
Выводы	107
ЗАКЛЮЧЕНИЕ	109
СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	112
СЛОВАРЬ ТЕРМИНОВ	113
Список литературы	115
СПИСОК ИЛЛЮСТРАТИВНОГО МАТЕРИАЛА	125
ПРИЛОЖЕНИЕ А. АЛГОРИТМ МАСА.....	127
ПРИЛОЖЕНИЕ Б. <i>FT-TEEN</i>	159
ПРИЛОЖЕНИЕ В. $2D$ и $3D$	163

ВВЕДЕНИЕ

Актуальность темы исследования. Беспроводные сенсорные сети (БСС) рассматривают как одно из самых важных современных направлений развития технологий в двадцать первом веке. В прошедшее десятилетие идея беспроводных сенсорных сетей получила большое признание и в научном мире, и в промышленности во всем мире.

Базовая идея БСС - это отказ от непосредственного участия человека в сборе информации, например, в связи с невозможностью присутствия людей в конкретном месте или при реализации технологического процесса, особенно если необходим сбор информации в течение долгого времени. При использовании БСС пользователи могут исследовать явления и процессы, и получать информацию, которая может быть как простой (измерения температуры), так и сложной (использование в военных областях).

На сегодняшний день беспроводные сенсорные сети *WSNs (Wireless Sensor Networks)* являются основным способом сбора данных для широкого спектра приложений, таких как военные, сельское хозяйство, окружающая среда, транспорт, здоровье и т.д.

Каждый сенсорный узел собирает данные от окружающей среды и передает их к шлюзам или базовым станциям (БС) либо непосредственно, либо через другие сенсорные узлы. Сенсорные узлы, как правило, обладают ограниченными возможностями по электропитанию и его восстановлению. Поэтому, выбор способа организации передачи информации между сенсорным узлом и шлюзами или базовыми станциями является одной из основных научных проблем при создании БСС.

Беспроводные сенсорные узлы (БСУ) потребляют энергию для передачи, обработки сообщений, а также для вычисления. Срок службы сенсорного узла сильно зависит от срока службы элементов питания, в виде которых достаточно часто выступают обычные батареи. Энергия, расходуемая во время передачи сообщений, может составить до 75% доступного энергетического

ресурса. При этом должны быть решены задачи как по отправке сообщений, так и по их маршрутизации.

Сенсорная сеть, в последнее время ее часто называют также сенсорным полем, может включать в себя тысячи узлов. В соответствии со спецификациями протокола *Zig Bee* сенсорная сеть может содержать до 64 000 (шестьдесят четырёх тысяч) узлов. Как уже отмечалось выше, БСС имеют ограниченные ресурсы: это ограниченная мощность системы питания, малая память, низкая скорость передачи данных и т.д. Эти ограничения непосредственно влияют на разработку протоколов и алгоритмов, используемых в БСС. Таким образом, разработанные алгоритмы для беспроводных сенсорных сетей должны эффективно работать на очень ограниченных ресурсах аппаратного обеспечения. При этом большую часть времени сенсорные узлы находятся в спящем состоянии, что требует использования для функционирования БСС принципов самоорганизующихся сетей.

Кластеризация оказалась при этом одним из важнейших методов создания БСС. Функционирование кластеризованной БСС во многом зависит от алгоритма выбора головного узла, основные требования к которому заключаются в требованиях по обеспечению максимальной длительности жизненного цикла сети и максимального покрытия. Следует отметить, что в кластерных БСС алгоритм выбора головного узла фактически совпадает с протоколом маршрутизации сообщений. В общем виде требования к алгоритму выбора головного узла могут быть сформулированы следующим образом:

- Алгоритм должен быть масштабируемым и эффективно функционировать в кластерах и сетях любого размера;
- Алгоритм должен минимизировать вычислительную сложность для узлов, таким образом, продлевая жизнь сети;
- Алгоритм должен быть как можно более простым и независимым от аппаратных возможностей узлов;
- Алгоритм должен ограничить количество необходимых передач, таким образом, увеличивая срок службы сети.

К настоящему времени разработано достаточно много алгоритмов выбора головного узла, в основном для БСС со стационарными сенсорными узлами. Весомый вклад в разработку принципов построения БСС, а также алгоритмов выбора головного узла внесли А. Е. Кучерявый, А. Салим, В. А. Мочалов, Е. А. Кучерявый, Д. А. Молчанов, П. А. Абакумов, W. Heinzelman, O.Yonis, M. Lindsey, D. Kim.

В то же время, целый ряд проблем, связанных с разработкой эффективных алгоритмов кластеризации для мобильных сетей, отказоустойчивых алгоритмов, еще требует своего решения. Поскольку кластеризация используется для БСС с достаточно высокой плотностью, важной представляется также разработка методики размещения сенсорных узлов, обеспечивающей требуемое покрытие пространства.

В связи с изложенным исследование, направленное на разработку новых алгоритмов выбора головного узла и методики размещения сенсорных узлов для эффективной кластеризации БСС, представляется актуальным.

Цель работы и задачи исследования. Цель диссертационной работы состоит в разработке алгоритмов выбора головного узла в БСС для сетей с мобильными узлами, отказами и разработке методики размещения сенсорных узлов, обеспечивающей требуемое покрытие пространства.

Для достижения поставленной цели в диссертационной работе последовательно решаются следующие задачи:

1. Анализ современного состояния в области исследований БСС, определение наиболее важных характеристик и структуры беспроводных сенсорных сетей;
2. Анализ существующих алгоритмов маршрутизации, самоорганизации и выбора головного узла кластера в беспроводных сенсорных сетях;

3. Оценка и сравнительный анализ алгоритмов прямой передачи DT и кластеризации *LEACH*, *SEP*, *TEEN*, *DEEC* в гомогенной, двухуровневой и многоуровневой гетерогенных сенсорных сетях;

4. Разработка нового адаптивного алгоритма кластеризации для беспроводных сенсорных сетей с мобильными узлами на основе комбинированного критерия прогнозирования и значения пригодности сенсорного узла;

5. Разработка нового отказоустойчивого алгоритма кластеризации для беспроводных сенсорных сетей (*FT-TEEN*);

6. Исследование характеристик покрытия, связности и плотности в двумерных (*2D*) и трехмерных (*3D*) беспроводных сенсорных сетях с целью разработки методики размещения сенсорных узлов, способной обеспечить, по крайней мере, 90% покрытие для *2D* и *3D* БСС;

7. Оценка длительности жизненного цикла сети, периода стабильности и пропускной способности БСС на основе отношения между радиусом покрытия и радиусом дальности связи.

Объект диссертации. Беспроводные сенсорные сети.

Предмет диссертации. Алгоритмы кластеризации и покрытие в беспроводных сенсорных сетях.

Научная новизна диссертационной работы заключается в следующих новых научных результатах:

1. Разработан адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами под названием *МАСА*, отличающийся комплексным применением известных ранее комбинированного критерия прогнозирования и значения пригодности сенсорного узла для выполнения роли головного, который за счет комплексного использования указанных выше величин обеспечивает большее значение длительности жизненно-

го цикла и увеличение длительности периода стабильности по сравнению с известными алгоритмами.

2. Разработан отказоустойчивый алгоритм кластеризации для беспроводных сенсорных сетей под названием *FT-TEEN*; отличающийся от известного алгоритма *TEEN* наличием резервных головных узлов кластера, обнаружением и восстановлением отказов в БСС, что позволяет увеличить число пакетов, успешно полученных как в головных узлах кластеров, так и на базовой станции;

3. Разработана методика размещения сенсорных узлов для двумерных (2D) и трехмерных (3D) БСС, отличающаяся от известных тем, что обеспечивается, по крайней мере, 90% покрытие пространства в зависимости от соотношения плотности размещения и радиуса действия сенсорного узла R_S .

Теоретическая и практическая значимость исследования. Теоретическая значимость диссертационной работы состоит в разработке и исследовании нового алгоритма выбора головного узла кластера в БСС с мобильными головными узлами, обеспечивающего большее значение длительности жизненного цикла, уменьшение потерь пакетов и увеличение длительности периода стабильности по сравнению с известными алгоритмами, отказоустойчивого алгоритма, использующего резервные головные узлы для увеличения числа пакетов, успешно полученных как в головных узлах кластеров, так и на базовой станции, а также определении методики размещения сенсорных узлов для двумерных и трехмерных БСС, отличающейся от известных тем, что обеспечивается, по крайней мере, 90% покрытие пространства в зависимости от соотношения плотности размещения и радиуса действия сенсорного узла R_S . Практическая ценность работы состоит в возможности использования полученных результатов для планирования и проектирования БСС.

Результаты диссертационной работы используются в СПбГУТ им. проф. М.А. Бонч-Бруевича при чтении лекций, проведении практических занятий и лабораторных работ по курсам “Современные проблемы науки в области телекоммуникаций” и “Интернет Вещей и самоорганизующиеся сети”.

Методы исследования. Для решения поставленных в работе задач использованы современные методы вычислительной геометрии, прогнозирования, имитационного моделирования. В качестве инструментов моделирования использовались программные пакеты *C# Visual.NET* и *MATLAB*, в целях визуализации полученных результатов применялось программное обеспечение *Microsoft Excel*.

Основные положения, выносимые на защиту:

- Адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами *MACA* обеспечивает большее значение длительности жизненного цикла и увеличение длительности периода стабильности по сравнению с известными алгоритмами (*DCA*, *LEACH-mobile*);
- Отказоустойчивый алгоритм кластеризации для беспроводных сенсорных сетей *FT-TEEN* обеспечивает увеличение числа пакетов, успешно полученных как в головных узлах кластеров, так и на базовой станции по сравнению с базовым алгоритмом *TEEN*;
- Методика размещения сенсорных узлов, обеспечивающая, по крайней мере, 90% покрытие для двумерных (2D) и трехмерных (3D) БСС в зависимости от соотношения плотности размещения и радиуса действия сенсорного узла R_S .

Степень достоверности и апробация результатов. Достоверность полученных в диссертационной работе результатов подтверждается корректным применением математического аппарата, результатами имитационного моделирования и широким обсуждением материалов диссертации на Международных и Всероссийских конференциях.

Материалы, отражающие основные результаты диссертационной работы, докладывались и обсуждались на 69-й и 70-й конференциях СПбНТОРЭС им. А.С. Попова (Санкт-Петербург, 2014, 2015); на 17-й Международной конференции по современным технологиям связи (*The 17th IEEE International Conference on Advanced Communication Technology – ICACT 2015, Korea, Phoenix Park, 1-3 June 2015*); на 15-ой международной конференции по проводным/беспроводным сетям связи следующего поколения «*15th International Conference on Next Generation Wired/Wireless Networking NEW2AN*» (St.-Petersburg, Russia, 2015); на IV и V международной научно-технической и научно-методической конференции «Актуальные проблемы инфотелекоммуникаций в науке и образовании» СПбГУТ (Санкт-Петербург, 2015, 2016), а также на заседаниях кафедры сетей связи и передачи данных СПбГУТ.

Публикации. Материалы, отражающие основные результаты диссертационной работы, опубликованы в сборниках научно-технических конференций, в том числе международных, а также в журналах отрасли. Всего опубликовано 9 печатных работ, из них 3 работы в зарубежных научно-технических сборниках (*Scopus*), 3 работы опубликованы в ведущих рецензируемых журналах, входящих в перечень ВАК Министерства образования и науки Российской Федерации, 1 статья в журнале, включенном в РИНЦ и тезисы докладов в количестве 2 в материалах научных конференций.

Структура и объем диссертации. Диссертация включает введение, пять глав, заключение, список сокращений и условных обозначений, словарь терминов, список литературы, включающий 108 наименований на русском и английском языке, список иллюстративного материала и 3 приложения. Основные результаты изложены на 164 страницах, содержат 57 рисунков, 9 таблиц и объем приложений составляет 39 страниц.

Личный вклад. Все результаты диссертационной работы получены автором лично.

Краткое содержание работы.

Во введении обоснована актуальность диссертационной работы, рассмотрено состояние исследуемой проблемы, формируются цель и задачи диссертации, перечислены основные научные результаты диссертации, определена научная новизна и практическая ценность результатов, описана область их применения, представлены основные положения, выносимые на защиту, приведены сведения об апробации работы, публикациях по теме работы, описана структура диссертации и ее объем.

В первой главе диссертационной работы проанализированы основные характеристики современных БСС и существующие работы в предметной области. Приводится информация о концепции Интернета Вещей (*IoT*), архитектура и приложения беспроводных сенсорных сетей, рассматриваются классификация беспроводных сенсорных сетей, проблемы проектирования и маршрутизации в беспроводных сенсорных сетях, анализируются особенности гомогенных и гетерогенных БСС, методы кластеризации.

Во второй главе проводится оценка и сравнительный анализ широко используемых при построении БСС алгоритмов маршрутизации и самоорганизации для беспроводных сенсорных сетей со стационарными узлами на плоскости. Сравнение алгоритмов осуществляется по результатам моделирования на *MATLAB*. В качестве показателей для сравнения используются длительность стабильного периода функционирования сенсорной сети и остаточная энергия. Анализируются алгоритм прямой передачи *DT* и наиболее известные алгоритмы кластеризации *LEACH* (*Low Energy Adaptive Cluster Hierarchy*), *SEP* (*Stable Election Protocol*), *DEEC* (*Distributed Energy Efficient Clustering*) и *TEEN* (*Threshold-sensitive Energy Efficient Protocol*). Сравнение алгоритмов производится для гомогенной, двухуровневой и многоуровневой гетерогенных сенсорных сетей.

В третьей главе разработан новый адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами под названием *MACA* (*Mobility Adaptive Clustering Algorithm*).

Множество исследовательских работ посвящено проблемам БСС со стационарными узлами. Тем не менее, передовые технологии для Интернета Вещей включают в себя применение более сложных приложений, которые требуют мобильности их узлов.

Кластерная архитектура нашла применение и в мобильных сенсорных сетях (MWSNs), поэтому поиск наилучших вариантов организации кластера и выбор головного узла для MWSN является сегодня актуальной задачей.

Разработанный алгоритм основан на сочетании лучших свойств двух известных существующих алгоритмов – *DCA (Distributed Clustering Algorithm)* и *MBC (Mobility-Based Clustering)*.

Четвёртая глава посвящена разработке отказоустойчивого алгоритма кластеризации для беспроводных сенсорных сетей.

В беспроводных сенсорных сетях (БСС) с кластерной архитектурой головные узлы кластеров играют ключевую роль в процессе функционирования сети и маршрутизации данных. Естественно, любые сбои в функционировании этих важных узлов влияют на качество услуг и надежность сети и могут привести даже к отказу сети в целом. Отказоустойчивость представляет собой одну из ключевых проблем в области проектирования и эксплуатации беспроводных сенсорных сетей БСС. Отказ некоторых узлов в БСС практически неизбежен по следующим причинам: отказ аппаратных средств, энергетическое истощение, сбой ПО, ошибки связи, вредоносная атака и т.д.

Предложенный алгоритм представляет собой модифицированный вариант алгоритма *TEEN*, вводит резервный механизм для кластеризации, обнаружение и восстановление ошибок и состоит из трех фаз.

В пятой главе исследуются покрытие, связность и плотность в двумерных и трехмерных беспроводных сенсорных сетях.

Несмотря на то, что большинство существующих работ в области беспроводных сенсорных сетей (БСС) в настоящее время посвящены двумерному пространству (*2D*), на самом деле такие сети работают в трехмерном пространстве (*3D*), особенно с учетом появления новых приложений, таких как

летающие сенсорные сети. Переход от двумерного к трехмерному пространству порождает множество новых проблем в связи с иной топологией сети. Требуются новые подходы к оценке таких характеристик БСС как покрытие, связность и плотность. Исходя из сказанного, в настоящей главе исследуются характеристики плотности и связности для БСС с целью определения такой методики размещения сенсорных узлов, чтобы возможно было обеспечить, по крайней мере, 90% покрытие для двумерных (2D) и трехмерных(3D) БСС. При этом оцениваются также длительность жизненного цикла сети, период стабильности и пропускная способность сети на основе отношения между радиусом покрытия (R_s) и радиусом дальности связи (R_c).

В заключении приводится описание основных результатов диссертационной работы.

В приложениях приводятся дополнительные материалы и акт внедрения.

1. АНАЛИЗ СУЩЕСТВУЮЩИХ РАБОТ В ПРЕДМЕТНОЙ ОБЛАСТИ ДИССЕРТАЦИИ

В настоящее время развитие сетей связи осуществляется на основе концепции Интернета Вещей [1, 6, 14]. Технологической базой для реализации концепции Интернета Вещей являются беспроводные сенсорные сети [11, 10]. Интернет вещей (*IoT*), иногда называют интернет объектов: Машина к машине, машина к инфраструктуре, машина к окружающей среде и т.д. *IoT* рассматривается как миллиарды умных, связанных “вещей” (своего рода “универсальная глобальная нейронная сеть” в облаке), которые охватят каждый аспект жизнедеятельности людей и общества.

Концепция ИВ включает множество различных приложений, например, следующие:

- Коммуникация от машины к машине;
- Коммуникация машины к инфраструктуре;
- Телемедицина: дистанционный мониторинг пациентов, доставка медикаментов;
- Непрерывный мониторинг техники;
- Автоматическая организация дорожного движения;
- Удаленная безопасность и контроль;
- Экологический мониторинг и контроль;
- Автоматизация производства.

Беспроводные сенсорные сети (БСС) представляют собой самоорганизующиеся сети и состоят из множества распределенных в пространстве беспроводных сенсорных узлов, предназначенных для мониторинга характеристик окружающей среды или объектов, расположенных в ней [12].

Ресурсы сенсорных узлов в беспроводных сенсорных сетях ограничены с точки зрения возможности обработки информации, пропускной способности, объема памяти, вычислительных возможностей, что существенно отличает беспроводные сенсорные сети от других сетей [17]. Рассмотрим далее архитектуру таких сетей.

1.1. Архитектура беспроводных сенсорных сетей

Беспроводные сенсорные сети (БСС) состоят из двух основных типов узлов:

1. Сенсоры (сенсорные узлы), которые развертывают в местах мониторинга (сенсорное поле) для сбора данных изучаемого явления или процесса и их передачи на базовые станции.

2. Базовая станция (БС), также известная как шлюз (*Gateway*) или приемник (*Sink*), является интерфейсом, соединяющим сенсорную сеть с внешним миром. БС может являться центральной точкой в сенсорном поле или находиться в отдалении от него. БС несет ответственность за получение данных от сенсоров, их обработку и доставку конечному пользователю разнообразными способами, включая телекоммуникационную сеть онлайн или через спутник и другие сети. Структура беспроводной сенсорной сети приведена на рисунке 1.1.

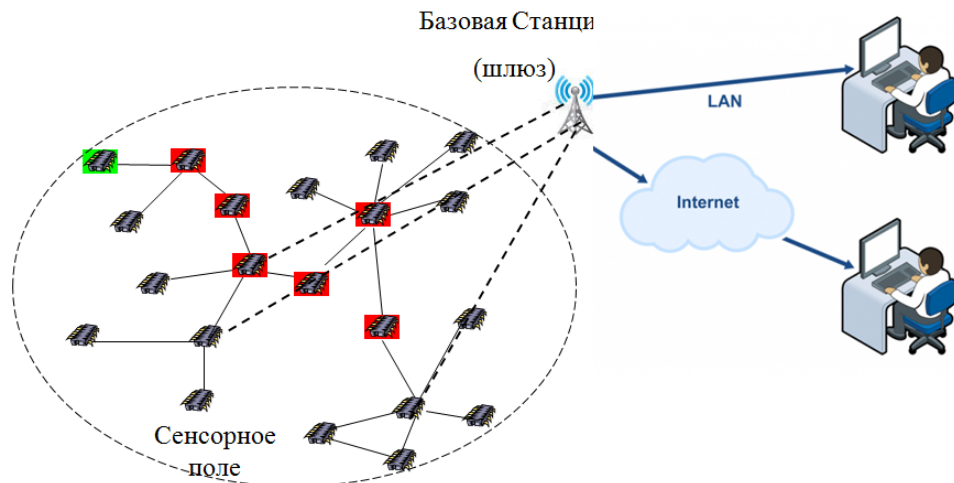


Рис. 1.1 – Структура БСС

Выбор конкретных решений для организации беспроводной сенсорной сети в первую очередь зависит от функциональных возможностей, размера, затрат, энергетических характеристик и, в настоящее время, при начале до-

статочного широкого внедрения сенсорных сетей – от обеспечения требуемого качества обслуживания.

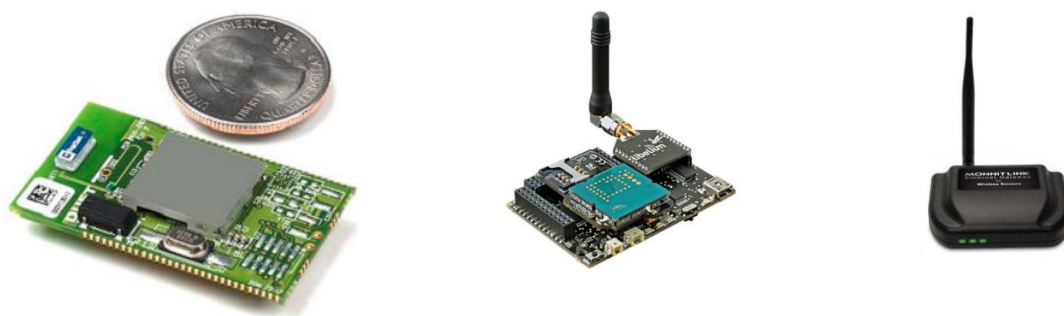


Рис. 1.2 – Сенсорные узлы

Примеры конструкций современных сенсорных узлов приведены на рисунке 1.2, а на рисунке 1.3 рассматривается типовая архитектура сенсорного беспроводного узла, которая состоит из четырех основных компонентов:

- 1) Сенсорной подсистемы, включающей один или несколько сенсоров (с соответствующими аналого-цифровыми преобразователями) для получения и накопления данных;
- 2) Подсистема обработки, включающая микроконтроллеры и память для местной обработки данных;
- 3) Радиопередатчик для беспроводной передачи данных;
- 4) Устройство электропитания

В зависимости от определенного конкретного применения сенсорные узлы могут также включать в себя дополнительные компоненты, такие как система позиционирования, чтобы определить их положение, мобилизатор, чтобы изменить их местоположение или конфигурацию (например, ориентация антенны), и так далее. Однако, поскольку последние компоненты дополнительные, мы не будем принимать их во внимание в последующем анализе.

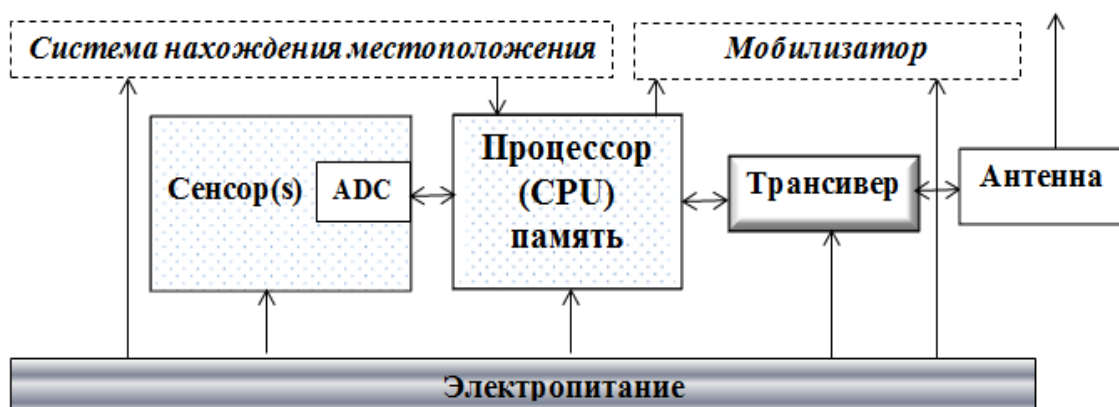


Рис. 1.3 – Типовая архитектура сенсорного узла

Сенсорные узлы образуют сенсорное поле, как правило, они находятся в спящем положении. Когда происходит событие, или по расписанию в определенное время, эти узлы просыпаются, самоорганизуются в сеть и передают информацию в центр сбора данных (см. Рисунок 1.4).

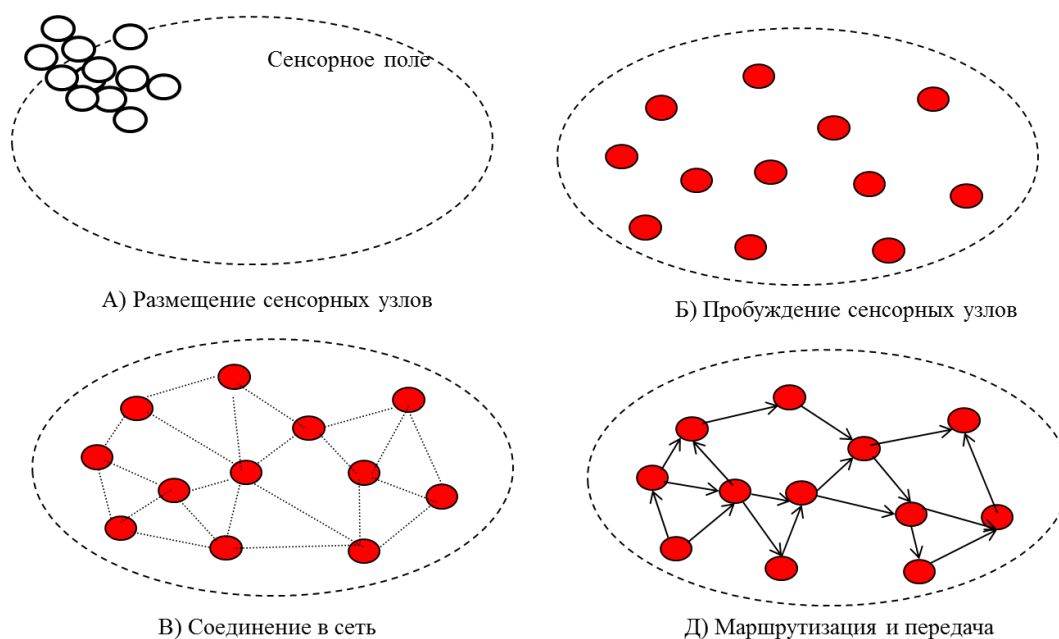


Рис. 1.4 – Самоорганизация и передача данных в беспроводных сенсорных сетях

1.2. Приложения беспроводных сенсорных сетей

БСС может состоять из сенсоров различных типов, включая сейсмический, магнитный, тепловой, визуальный, инфракрасный, акустический и ра-

дар, которые в состоянии контролировать большое разнообразие внешних условий, включая следующие: температура, влажность, давление, скорость, направление движения, освещенность, уровень шума, присутствие или отсутствие определенных видов объектов и т.д.

Этот спектр приложений БСС может быть, главным образом, разделен на следующие категории: военная, экологическая, здоровье, дом. Рассмотрим далее для примера ряд приложений беспроводных сенсорных сетей:

- **Наблюдения за внешней средой.** С помощью БСС возможен мониторинг изменения климата, пожароопасности в лесах, контроля освещённости, влажности, температуры в сельском хозяйстве и т.д.

Военный мониторинг. Сенсорные сети могут быть использованы в военных наблюдениях. Приложения беспроводных сенсорных сетей показаны на рисунке 1.5.



Рис. 1.5 – Приложения беспроводных сенсорных сетей

- **Мониторинг зданий.** Достаточно важные задачи по мониторингу жилого фонда, систем водоснабжения и т.д. составляют значительную часть приложений сенсорных сетей.
- **Контроль здоровья.** Сенсорные узлы могут размещаться не только в зданиях, в лесах и т.д., но и на и в теле человека, и иных биомасс. Одно из новых важнейших приложений сенсорных сетей получило название *MBAN (Medicine Body Area Network)* и изучается в рабочей группе *IEEE 802.15.6*.

Список приложений можно было бы продолжить, но уже рассмотренных вполне достаточно для того, чтобы определить всепроникающий характер сенсорных сетей.

1.3. Характеристики архитектуры и классификация беспроводных сенсорных сетей

Беспроводные сенсорные сети могут быть классифицированы на основе различных концепций в зависимости от типа узлов в сети или в зависимости от окружающей среды, в которой они развернуты, а также в зависимости от метода развертывания или в зависимости от расположения узлов в сети и т.д. Выделим некоторые принципиальные для дальнейшей разработки алгоритмов выбора головного узла характеристики БСС.

- **Проактивные и Реактивные БСС.** На основе способа функционирования и целевого применения сенсорных сетей они могут быть классифицированы на проактивные и реактивные БСС. В проактивной БСС сенсорные узлы в сети периодически проверяют среду и передают данные, представляющие интерес для БСС. В реактивных БСС узлы оперативно реагируют на внезапные и резкие изменения в области сенсорного поля.
- **Гомогенные и Гетерогенные БСС.** Сенсорные сети могут быть разделены на два основных типа Гомогенные (однородные) и Гетерогенные БСС (неоднородные) в зависимости от составляющих их узлов. В однородных беспроводных сенсорных сетях все сенсорные узлы одинаковы с точки

зрения энергии батареи и аппаратной сложности. Гетерогенные же сенсорные сети могут содержать два, три или больше типов узлов, соответственно с различными энергетическими и функциональными возможностями.

- **Одноранговые и Иерархические сети.** БСС могут быть классифицированы в зависимости от структуры сети на Одноранговые и Иерархические: В одноранговой сети все узлы выполняют одинаковые задачи и передача данных на шлюз или базовую станцию осуществляется непосредственно. В иерархических сетях узлы подразделяются на головные и сенсорные узлы. Сенсорные узлы собирают данные, а головные занимаются их обработкой, анализом и передачей на шлюз или БС (см. Рисунок 1.6).

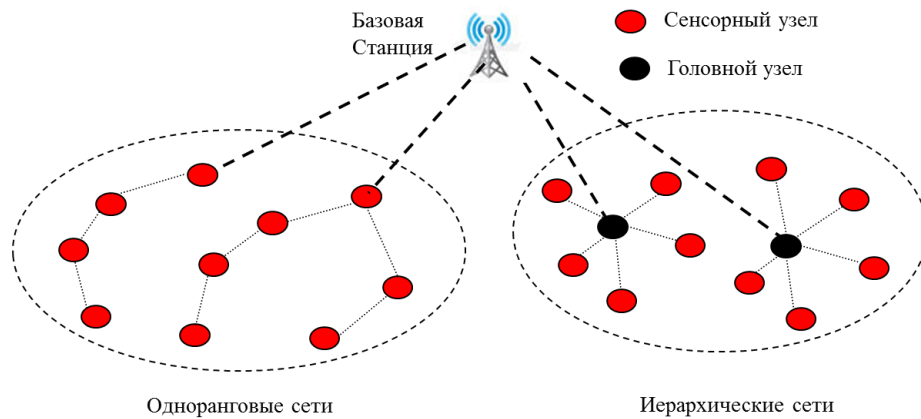


Рис. 1.6 – Классификация БСС в зависимости от структуры сети

- **Случайное и детерминированное размещение сенсорных узлов.** БСС могут быть классифицированы в зависимости от метода размещения сенсорных узлов. При случайном размещении сенсорные узлы могут быть случайным образом разбросаны по некоторой области. Детерминированное размещение предполагает размещение узлов в соответствии с предварительно определенным планом построения сети. Естественно, что алгоритмы распределения данных между сенсорными узлами в первом и втором случае могут быть существенно различны.

- **Статичность и Мобильность.** Потребность в мониторинге людей, животных и вещей в целом вызывает необходимость применения мо-

бильных узлов в БСС. Сенсорные узлы, используемые в беспроводных сенсорных сетях, могут быть стационарным или мобильным. При этом мобильные сенсорные узлы могут перемещаться с места на место, из-за чего связь между двумя узлами в сенсорной сети с мобильными узлами может быть очень сложной.

- **Двумерные и трехмерные беспроводные сенсорные сети.** БСС могут быть также классифицированы на двумерные и трехмерные беспроводные сенсорные сети. Несмотря на то, что большинство существующих работ в области беспроводных сенсорных сетей в настоящее время посвящены двумерному пространству, на самом деле такие сети работают в трехмерном пространстве, особенно с учетом появления новых приложений, таких как летающие сенсорные сети. Таким образом, требуется оценить возможность применения известных протоколов для $2D$ в области БСС, работающих в $3D$.

1.4. Маршрутизация в беспроводных сенсорных сетях

Беспроводные сенсорные сети, как правило, функционируют на базе протокола *IEEE 802.15.4* (Gutierrez et al. 2001), который был разработан специально для коммуникаций на небольшом расстоянии с малой мощностью сенсорных сетей, и поддерживается большинством академических и коммерческих сенсорных узлов. Когда диапазон передачи данных всех сенсорных узлов достаточно большой, и сенсорные узлы могут передавать свои данные непосредственно к базовой станции, они могут сформировать звездную топологию, как показано слева на рисунке 1.7.

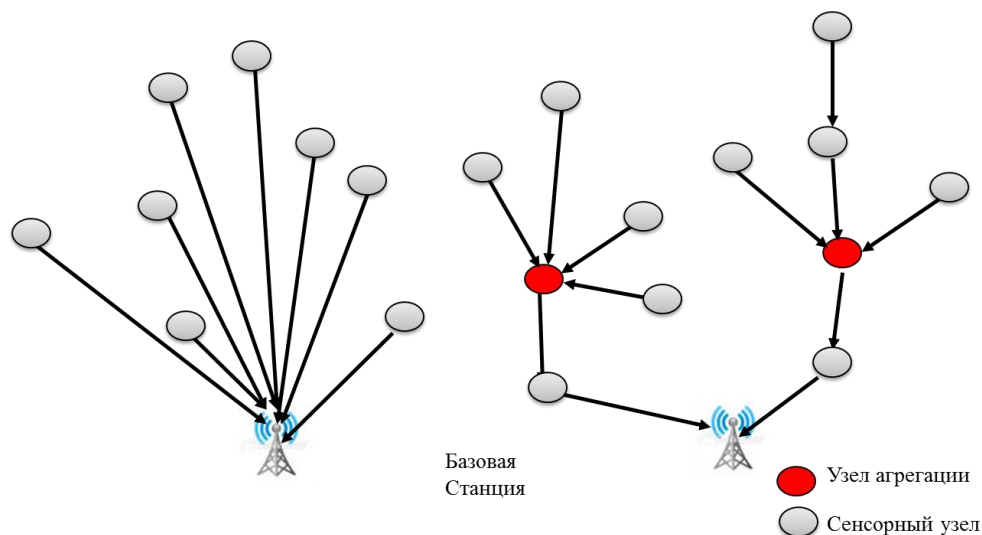


Рис. 1.7 – Одношаговое и многошаговое взаимодействие в БС

В этой топологии каждый сенсорный узел непосредственно связывается с базовой станцией, используя единственный шаг. Поэтому, такое взаимодействие называется одношаговым. Тем не менее, сенсорные сети часто покрывают большие географические районы, и мощность радиопередачи должна быть сведена к минимуму в целях экономии энергии; следовательно, многошаговое взаимодействие является более общим случаем для сенсорных сетей (показано справа на рисунке 1.7). В этой топологии сети сенсорные узлы должны не только передать свои собственные данные, но также служат в качестве передатчиков для других сенсорных узлов, то есть, они должны обеспечить передачу данных и от других сенсорных узлов к базовой станции. Эта проблема маршрутизации, то есть, задача нахождения пути с несколькими шагами ретрансляции от сенсорного узла на базовую станцию, является одной из самых важных проблем и получила огромное внимание научного сообщества.

Маршрутизация в беспроводных сенсорных сетях является весьма сложной задачей из-за присущих БС особенностей, которые отличают эти сети от других беспроводных сетей таких, как мобильные *Ad Hoc* сети или сотовые сети:

- Традиционная адресация на основе *IP*-протоколов не может быть применена к БСС из-за относительно большого количества сенсорных узлов. В БСС иногда получить данные важнее, чем знать идентификаторы узлов, с которых они отправлены.

- Ресурсы сенсорных узлов в беспроводных сенсорных сетях ограничены с точки зрения возможностей обработки информации, пропускной способности, объема памяти, вычислительных возможностей, таким образом, требуется рациональное управление ресурсами.

- Сенсорные узлы как очень простые элементы зачастую ненадежны.

- Топология сенсорных сетей изменяется очень часто. В некоторых приложениях сенсорные узлы могут мобильными и изменять свое местоположение.

- Некоторые узлы в сети осуществляют одни и те же цели, то есть трафик данных может быть сгенерирован так, что различные сенсорные узлы будут передавать одну и ту же информацию.

С учетом вышеизложенных характеристик было создано достаточно много протоколов маршрутизации для применения в беспроводных сенсорных сетях (БСС). Протоколы, предназначенные для *Ad-hoc* сетей, не могут эффективно использоваться в БСС, поскольку они не используют специфику БСС и соответствующие требования.

1.5. Проблемы планирования и маршрутизации в беспроводных сенсорных сетях

Одна из главных целей маршрутизации в БСС состоит в том, чтобы выполнить передачу данных, продлить при этом срок службы сети и предотвратить деградацию соединений. Создание протоколов маршрутизации в БСС зависит от многих ограничивающих факторов. Эти факторы должны быть учтены, прежде чем эффективная связь может быть установлена в БСС.

Одним из наиболее важных вопросов является адаптация к изменениям в связности, масштабируемости и отказоустойчивости.

Поэтому при разработке алгоритмов маршрутизации и планировании беспроводных сенсорных сетей должны учитываться следующие факторы:

- **Потребление энергии (Энергетическая эффективность).** Идеальный алгоритм должен обеспечить своевременную передачу информации с заданной точностью и с минимальными энергозатратами. В многошаговой БСС каждый узел играет двойную роль: сенсор и маршрутизатор данных. Сбой некоторых сенсорных узлов из-за перерыва в питании может вызвать значительные топологические изменения и может потребовать повторной маршрутизации пакетов и реорганизации сети. Энергетическая эффективность – одна из важнейших задач в создании беспроводных сенсорных сетей.

- **Модель передачи данных.** Сбор и передача данных в БСС зависят от приложения и актуальности представления данных (сообщений). Модели передачи данных могут быть классифицированы как непрерывная, управляемая событиями и гибридная. Непрерывная модель передачи данных подходит для приложений, которые требуют постоянного мониторинга. Таким образом, сенсоры и передатчики будут работать непрерывно. В событийно-управляемой и гибридной моделях сенсорные узлы реагируют на появление какого-либо события или запроса и генерируют сообщения на головной узел кластера или на БС. Алгоритм выбора головного узла и протокол маршрутизации во многом зависят от модели передачи данных, непосредственно связанной с потреблением энергии и стабильностью маршрута.

- **Гетерогенные узлы.** В зависимости от применения сенсорный узел может играть различную роль в сети или иметь различные параметры. Существование разнородных сенсоров поднимает много технических проблем, связанных с маршрутизацией данных.

- **Толерантность к отказам или надежность.** Некоторые сенсорные узлы могут выходить из строя или быть заблокированы вследствие недостаточного уровня электропитания, возникновения критических условий во

внешней среде, физического повреждения или помех. Отказ отдельных сенсорных узлов не должен влиять на общее функционирование сенсорной сети. БСС должна поддерживать требуемый уровень качества обслуживания и обеспечивать необходимую надежность. Далее в диссертации будет предложен новый отказоустойчивый алгоритм кластеризации для беспроводных сенсорных сетей.

– **Масштабируемость.** В зависимости от решаемой задачи число сенсорных узлов в беспроводных сенсорных сетях, размещенных в сенсорном поле, может изменяться от нескольких сотен до тысяч или более. Плотность сенсорных узлов можно определить по формуле:

$$n\mu(R) = (N\pi R^2)/A$$

где: N – число узлов, R – радиус связности сенсорного узла.

– **Гибкость.** Алгоритмы в сенсорных сетях должны быть способны адаптироваться к различным приложениям БСС. Условия работы и возможности самого сенсорного узла могут сильно изменяться. Всё это должно быть учтено при разработке алгоритмов для беспроводных сенсорных сетей.

– **Средства передачи.** Характеристики окружающей среды также определяют метод радиосвязи для БСС. Например, для подводных беспроводных сенсорных сетей *UWSN (Underwater Wireless Sensor Networks)* средой передачи является вода. Традиционные проблемы, связанные с беспроводным каналом (например, замирание, высокий коэффициент ошибок и помехи), также могут влиять на работу сенсорной сети.

– **Связность.** Высокая плотность узлов в сенсорных сетях способствует поддержанию необходимого значения связности. Тем не менее, связность может уменьшаться вследствие вмешательства, помех, шумов или препятствий.

– **Покрывтие.** В беспроводных сенсорных сетях каждый сенсорный узел покрывает ограниченную физическую область окружающей среды. Доля покрытия пространства также является важным параметром БСС, особенно для систем мониторинга.

– **Мобильность.** Большинство приложений предполагает, что сенсорные узлы стационарны. Тем не менее, мобильность сенсорных узлов может быть необходима в различных приложениях, например, летающих сенсорных сетях. Маршрутизация данных от или до движущихся узлов является более сложной, так как стабильность маршрута становится важной проблемой в дополнение к энергии, пропускной способности и т.д.

– **Двумерное и трехмерное пространство.** Сенсорные сети могут часто развертываться в трехмерном пространстве, например, в многоэтажных зданиях, при мониторинге лесов, подводном мониторинге, летающих сенсорных сетях и т.д. Переход от двумерного пространства к трехмерному порождает множество новых задач в связи с иной топологией сети.

– **Агрегация данных.** Сенсорные узлы могут генерировать значительные избыточные данные, включая аналогичные сообщения о событиях от нескольких узлов. Естественно, такие данные можно агрегировать для того, чтобы число сообщений уменьшилось. Объединение данных является типичной операцией во многих приложениях БСС, особенно, в иерархических протоколах маршрутизации для самоорганизующейся БСС. Такой метод широко используется для увеличения жизненного цикла сети.

– **Самоорганизация.** Сенсорные сети должны иметь возможность самоорганизации. Поэтому, вычислительные возможности, возможности обеспечения связи, возможности управления должны обеспечивать и возможность автономного существования БСС в течение какого-либо времени.

– **Точность и латентность.** Обеспечение точной информацией в реальном времени – одна из главных задач приложений БСС. Данные должны передаваться через беспроводную сенсорную сеть своевременно и точно. Идеальный алгоритм должен обеспечивать качественную передачу данных с минимальными энергозатратами.

1.6. Гомогенные и гетерогенные БСС

Сенсорные сети могут быть разделены на два основных типа: гомогенные (однородные) и гетерогенные (неоднородные) сенсорные сети [71].

- **Гомогенные.** В однородных беспроводных сенсорных сетях все сенсорные узлы одинаковы с точки зрения энергии батареи и функциональных возможностей.

- **Гетерогенные.** Сенсорные сети могут содержать два, три или больше типов узлов, соответственно с различными энергетическими и функциональными возможностями.

1.6.1. Типы гетерогенных ресурсов

Есть три распространенных вида неоднородности ресурсов в сенсорных узлах: вычислительная неоднородность, неоднородность сетевых возможностей и неоднородность энергии.

Вычислительная неоднородность означает, что гетерогенный узел имеет более мощный микропроцессор и больше памяти, чем типовой сенсорный узел. С помощью более мощных вычислительных ресурсов узлы могут обеспечить комплексную обработку данных и их долгосрочное хранение.

Неоднородность сетевых возможностей означает, что гетерогенный узел имеет более высокую пропускную способность и более мощный радио передатчик, чем типовой узел. Такие гетерогенные узлы могут обеспечить более надежную передачу данных, чем типовые.

Неоднородность энергии означает, что для гетерогенного сенсорного узла существует возможность либо постоянного электропитания от сети, либо возможность замены батареи или аккумулятора в течение срока службы сенсорного узла.

1.6.2. Влияние неоднородности сенсорных узлов на БСС

Размещение нескольких разнородных узлов в сенсорной сети может принести следующие преимущества:

Уменьшение задержки передачи данных: Вычислительная неоднородность может уменьшить задержку обработки информации для ближайших узлов, а неоднородность сетевых возможностей – уменьшить длительность ожидания в очереди для передачи информации.

Увеличение длительности жизненного цикла сети: Средний расход энергии для передачи пакета от обычных узлов в БС в гетерогенных сенсорных сетях будет гораздо меньше энергии, потребляемой в однородных сенсорных сетях.

Повышение надежности передачи данных: Использование в гетерогенной БСС сенсорных узлов с высокой пропускной способностью и более мощными радиопередатчиками может повысить надежность передачи данных.

На рисунке 1.8 показана гетерогенная модель для беспроводной сенсорной сети.

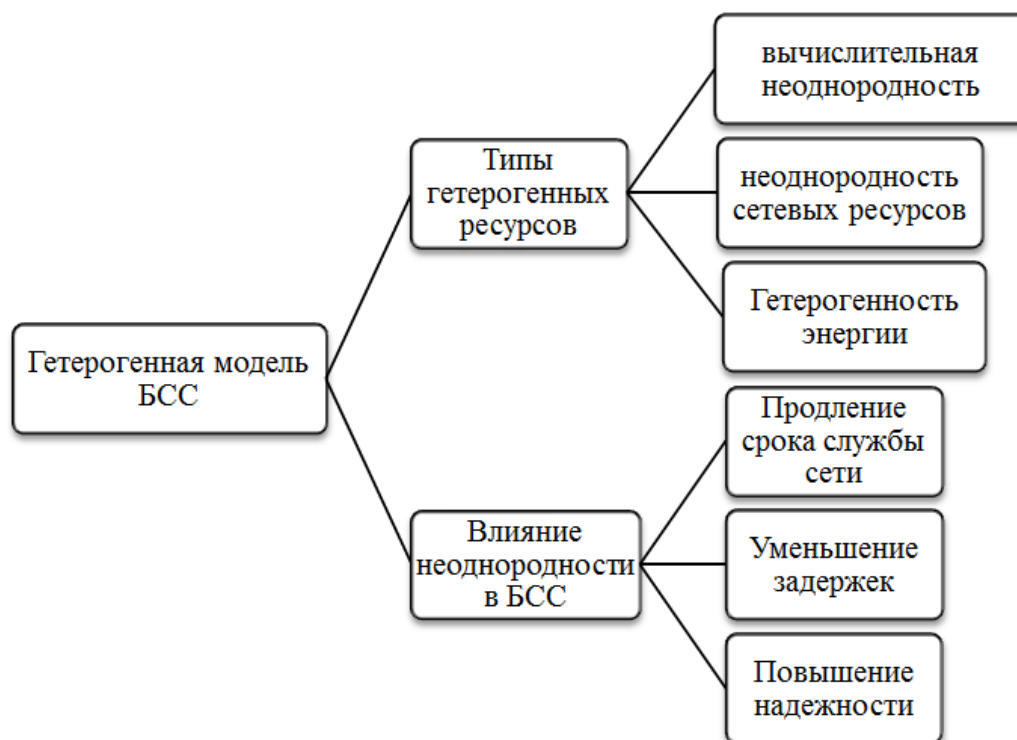


Рис. 1.8 – Гетерогенная модель БСС

1.7. Кластеризация БСС

Кластеризация является неотъемлемой частью иерархических протоколов маршрутизации. В процессе исследований по кластеризации БСС было разработано несколько механизмов кластеризации, таких как *LEACH*, *PEGASIS*, *TEEN* и *APTEEN*. Как правило, эти механизмы кластеризации основаны на выборе головного узла кластера. Поэтому, наиболее распространенными для кластерных БСС являются задачи по разработке алгоритмов выбора головных узлов кластера. Пример кластерной БСС показан на рисунке 1.9.

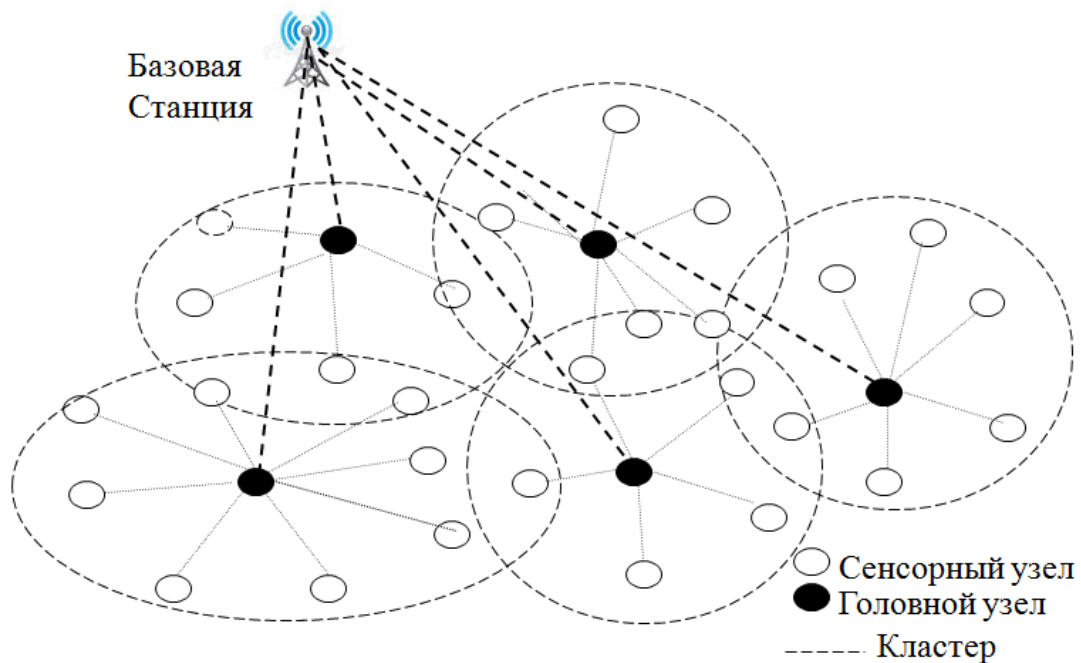


Рис. 1.9 – Кластерная БСС

Протоколы маршрутизации на основе кластеризации состоят из двух фаз, которые в свою очередь делятся на два этапа: выбор головного узла кластера *CH* (*Cluster Head*), формирование кластера, агрегирование данных и передача данных (см. Рисунок.1.10).

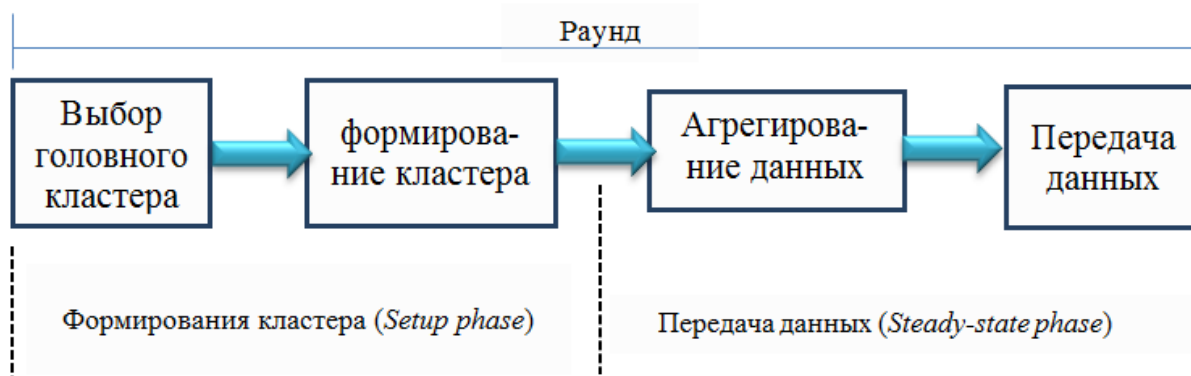


Рис. 1.10 – Блок-схема протоколов маршрутизации на основе кластеризации (процесс кластеризации)

Сенсорные узлы в соответствии с выполняемыми ими функциями в алгоритмах кластеризации могут быть сгруппированы в следующие четыре категории:

- **Головной узел кластера (CH):** Координация группы узлов, расположенных в границах кластера, агрегация данных от членов кластера и передача собранных или агрегированных данных на следующий узел являются основными функциями CH.
- **Базовая станция (BS):** Учитывая высокие возможности обработки информации и неограниченный источник энергии, БС может быть координатором сети и / или приемным узлом, где все агрегированные данные обрабатываются в соответствии с приложением БСС и требованиями конечного пользователя.
- **Релейный узел RN (Relay node):** Транзитные узлы в многошаговых сетях, выполняющие функции передачи собранных или агрегированных данных другими узлами, к месту назначения.
- **Типовой узел GN (General node):** Большинство узлов в сети, которые только обеспечивают сбор данных, основанных на типе применения.

Рисунок 1.11 иллюстрирует два варианта кластеризации. В первом из них (рисунок 1.11a) все головные узлы кластера передают информацию непосредственно на базовую станцию. Для второго варианта кластеризации образуется многошаговая сеть, предполагающая, что от части головных сен-

сорных узлов передача данных на базовую станцию осуществляется либо через другие головные узлы кластеров, либо через специально выделенные транзитные узлы (на рис.1.1б не показаны).

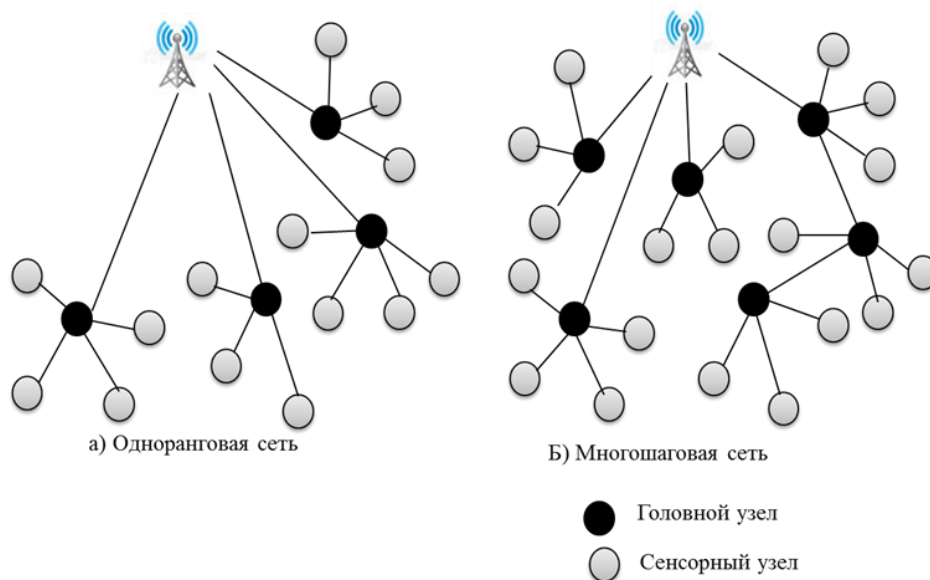


Рис. 1.11 – Кластеризация с одноранговой сетью головных узлов (а) и кластеризация с многошаговой сетью головных узлов (б).

1.7.1. Задачи диссертационной работы.

Проведенный выше анализ позволил сконцентрировать внимание на задачах, связанных с кластеризацией БСС. Кластеризация оказалась одним из важнейших методов создания БСС. Функционирование кластеризованной БСС во многом зависит от алгоритма выбора головного узла, основные требования к которому заключаются в требованиях по обеспечению максимальной длительности жизненного цикла сети и максимального покрытия. Следует отметить, что в кластерных БСС алгоритм выбора головного узла фактически совпадает с протоколом маршрутизации сообщений. В общем виде требования к алгоритму выбора головного узла могут быть сформулированы следующим образом:

1. Алгоритм должен быть масштабируемым и эффективно функционировать в кластерах и сетях любого размера.

2. Алгоритм должен минимизировать вычислительную сложность для узлов, таким образом, продлевая жизнь сети.

3. Алгоритм должен быть как можно более простым и независимым от аппаратных возможностей узлов.

4. Алгоритм должен ограничить количество необходимых передач, таким образом, увеличивая срок службы сети.

К настоящему времени разработано достаточно много алгоритмов выбора головного узла, в основном для БСС со стационарными сенсорными узлами. В то же время, целый ряд проблем, связанных с разработкой эффективных алгоритмов кластеризации для мобильных сетей, отказоустойчивых алгоритмов, еще требует своего решения. Поскольку кластеризация используется для БСС с достаточно высокой плотностью, важной представляется также разработка методики размещения сенсорных узлов, обеспечивающей требуемое покрытие пространства. Именно на решение этих проблем и направлена диссертационная работа.

Таким образом, цель диссертационной работы состоит в разработке новых алгоритмов выбора головного узла в БСС для сетей с мобильными узлами, отказами и разработке методики размещения сенсорных узлов, обеспечивающей требуемое покрытие пространства.

Для достижения поставленной цели в диссертационной работе последовательно решаются следующие задачи:

- анализ современного состояния в области исследований БСС, определение наиболее важных характеристик и структуры беспроводных сенсорных сетей;
- анализ существующих алгоритмов маршрутизации, самоорганизации и выбора головного узла кластера в беспроводных сенсорных сетях;
- оценка и сравнительный анализ алгоритмов прямой передачи *DT* и кластеризации *LEACH*, *SEP*, *TEEN*, *DEEC* в гомогенной, двухуровневой и многоуровневой гетерогенных сенсорных сетях;

- разработка нового адаптивного алгоритма кластеризации для беспроводных сенсорных сетей с мобильными узлами на основе комбинированного критерия прогнозирования и значения пригодности сенсорного узла;
- разработка нового отказоустойчивого алгоритма кластеризации для беспроводных сенсорных сетей (*FT-TEEN*);
- исследование характеристик покрытия, связности и плотности в двумерных (*2D*) и трехмерных (*3D*) беспроводных сенсорных сетях с целью разработки методики размещения сенсорных узлов, способной обеспечить, по крайней мере, 90% покрытие для *2D* и *3D* БСС;
- оценка длительности жизненного цикла сети, периода стабильности и пропускной способности БСС на основе отношения между радиусом покрытия (R_S) и радиусом дальности связи (R_C).

Выводы

1. Саморегулирующийся характер, особые условия по энергетической эффективности, толерантность к отказам и масштабное значение числа узлов в БСС требуют разработки специфических алгоритмов маршрутизации для беспроводных сенсорных сетей.
2. Архитектура сенсорной сети может быть представлена как одноуровневая или как иерархическая. При иерархической архитектуре построения БСС наибольшее распространение получил метод кластеризации сети.
3. К настоящему времени разработано достаточно много алгоритмов выбора головного узла, в основном для БСС со стационарными сенсорными узлами. В то же время, целый ряд проблем, связанных с разработкой эффективных алгоритмов выбора головного узла для мобильных сетей, отказоустойчивых алгоритмов еще требует своего решения. Поскольку кластеризация используется для БСС с достаточно высокой плотностью, важной

представляется также разработка методики размещения сенсорных узлов, обеспечивающей требуемое покрытие пространства.

4. По результатам проведенного аналитического исследования сформулированы цель и задачи диссертационной работы.

2. ОЦЕНКА И СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ МАРШРУТИЗАЦИИ ДЛЯ ГОМОГЕННЫХ И ГЕТЕРОГЕННЫХ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ

2.1. Обзор исследований беспроводных сенсорных сетей.

Беспроводные сенсорные сети представляют собой самоорганизующиеся сети, состоящие из большого количества недорогостоящих, энергоэффективных и multifunctionальных беспроводных узлов, включающих сенсоры, возможности радиосвязи и вычислений. Сенсорные узлы соединены на не-большом расстоянии через беспроводную среду и выполняют общую задачу: сбор данных о наблюдаемых явлениях, событиях, процессах и передачу данных в БС для обработки и анализа. Примерами собираемой информации могут быть данные об окружающей среде, военное наблюдение и управление производственным процессом.

Сенсорные узлы БСС довольно часто могут функционировать в условиях, когда отсутствует возможность гарантированного электропитания или даже замены батареи сенсорного узла. Поэтому важнейшей характеристикой БСС является остаточная энергия [53, 54]. Не случайно, в настоящее время энергетические вторжения в БСС рассматриваются как наиболее эффективные с точки зрения снижения длительности жизненного цикла сети [5, 28, 27].

В последнее время в исследовательских работах появилась еще одна группа алгоритмов маршрутизации для БСС – специализированные алгоритмы для n -мерного пространства [24, 1, 15], но в данной главе эти алгоритмы не рассматриваются. Также в главе не рассматриваются специализированные алгоритмы для мобильных сенсорных сетей [52, 51, 50], т.е. предполагается, что сенсорные узлы располагаются в фиксированной точке пространства.

В главе проводится оценка и сравнительный анализ широко используемых при построении БСС алгоритмов маршрутизации и самоорганизации для беспроводных сенсорных сетей со стационарными узлами на плоскости.

Сравнение алгоритмов осуществляется по результатам моделирования на *MATLAB*.

В качестве показателей для сравнения используются длительность стабильного периода функционирования сенсорной сети, остаточная энергия, длительности периодов времени до гибели первого и последнего сенсорных узлов. Анализируются алгоритм прямой передачи *DT* и наиболее известные алгоритмы кластеризации *LEACH* (*Low Energy Adaptive Cluster Hierarchy*), *SEP* (*Stable Election Protocol*), *DEEC* (*Distributed Energy Efficient Clustering*) и *TEEN* (*Threshold-sensitive Energy Efficient Protocol*). Сравнение алгоритмов производится для гомогенной, двухуровневой и многоуровневой гетерогенных сенсорных сетей.

2.2. Классификация алгоритмов маршрутизации в БСС.

Многие исследователи предлагали различные решения по алгоритмам маршрутизации для БСС с учётом требуемых особенностей, изложенных выше. Далее будем классифицировать алгоритмы маршрутизации, используемые в БСС, в зависимости от множества параметров и критериев.

Алгоритм маршрутизации считается адаптивным, если некоторые параметры системы можно контролировать для того, чтобы адаптироваться к текущим условиям сети и доступным энергетическим уровням. Кроме того, эти алгоритмы могут быть классифицированы на основе сетевой структуры, выбора маршрута и процесса управления и т.д., как было показано на рисунке 1.2. Алгоритмы маршрутизации могут быть классифицированы в соответствии с сетевой структурой как иерархические *LEACH* (*Heinzelman, 2002*), неиерархические *SPIN* (*Heinzelman, 1999*) и основанные на местоположении.

Алгоритмы маршрутизации на основе выбора маршрута могут быть разделены на 4 категории: многолучевые алгоритмы (*Multipath-based protocols*), алгоритмы маршрутизации на основе теории очередей (*Query-based routing protocols*) *COUGAR* (*Yao, 2002*), алгоритмы на основе местоположения (*negotiation-based protocols*) *GFG* (*Intanagonwiwat, 2000*), алгоритмы на основе

качества обслуживания (*QoS-based routing protocols*) SAR (Sohrabi, 2000). В дополнение к вышеизложенному, алгоритмы маршрутизации на основе процессов управления могут быть классифицированы на три категории: активные, реактивные и гибридные.



Рис. 2.1 – Категории протоколов маршрутизации в БСС

2.2.1. Алгоритмы маршрутизации на основе сетевой структуры.

Структура БСС может быть разделена на одношаговые сети (*One-hop networks*) в сенсорных полях небольшого размера и иерархические многошаговые сети (*Multi-hop networks*). Далее рассмотрим более подробно алгоритмы маршрутизации и самоорганизации в БСС, которые могут быть классифицированы в соответствии с сетевой структурой.

- **Алгоритмы на основе местоположения (*location based*)**

Информация о местонахождении узла может быть использована для маршрутизации. Сенсорные узлы, как правило, распределены случайным образом на сенсорном поле, и местоположение этих сенсоров определяется при помощи системы глобального позиционирования (*GPS*, ГЛОНАСС) или использования других методов, например, систем локального позиционирования.

ния на базе *WiFi*. Заметим, что знание информации о географическом положении может представлять собой дополнительные расходы, особенно когда расположение узлов изменяется с течением времени, что характерно для мобильных сенсорных сетей. Наиболее известными алгоритмами, которые принадлежат к данной категории, являются:

- Географическая и энергетическая маршрутизация *GEAR* (*Geographic and Energy Aware Routing*).
- Географическая маршрутизации по расстоянию *GEDIR* (*Geographic Distance Routing*).

- **Неиерархические алгоритмы. Примерами неиерархических алгоритмов маршрутизации могут быть следующие:**

- **Алгоритм прямой передачи (*Direct Transmission*).**

Алгоритм прямой передачи представляет собой неиерархический алгоритм маршрутизации для одношаговых сетей, где каждый сенсорный узел передает информацию непосредственно к шлюзу или базовой станции независимо от расстояния и все сенсорные узлы обладают одинаковой функциональностью. [38]. На рисунке 2.2. показан неиерархический алгоритм прямой передачи *DT*.

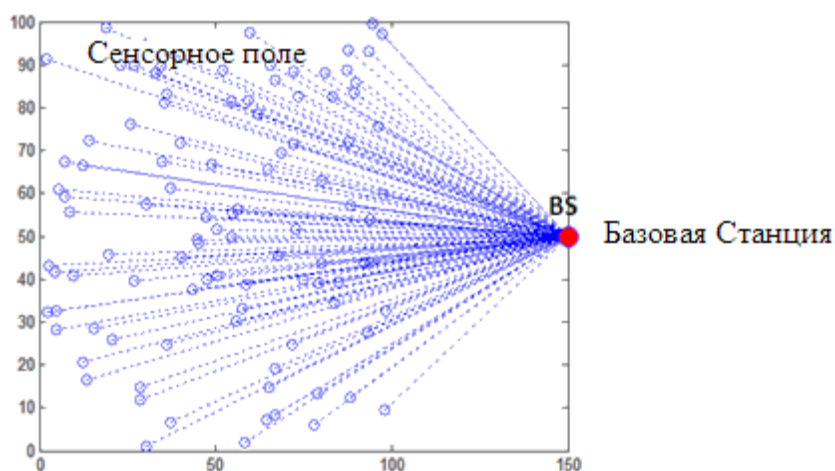


Рис. 2.2 – Алгоритм прямой передачи в БСС

– Алгоритм Наводнения

Алгоритм наводнения - это многошаговый неиерархический алгоритм маршрутизации, для которого каждый узел, получая пакет данных, передает его всем своим соседям. Этот процесс продолжается до тех пор, пока пакет не поступает на БС или шлюз, или не достигает максимально возможного количества переходов. Этот алгоритм очень легко реализовать, но он имеет определенные недостатки, которые включают дублирование сообщений, переданных в один и тот же сенсорный узел, и, как следствие, большое энергопотребление. Алгоритм приведен на рисунке 2.3.

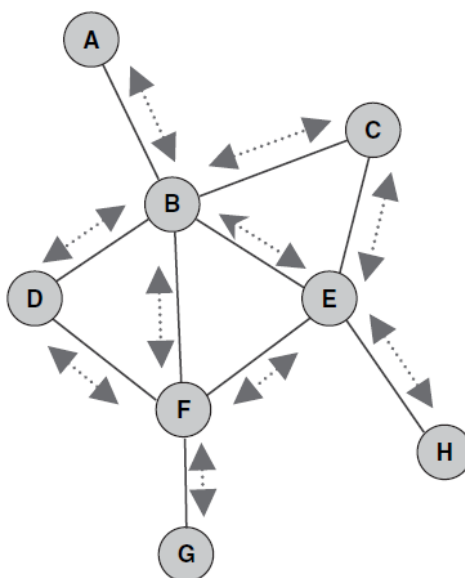


Рис. 2.3 – Алгоритм наводнения в БСС

– Алгоритм распространения слухов.

Алгоритм распространения слухов представляет собой улучшенный вариант алгоритма наводнения. В этом алгоритме вместо того, чтобы передавать каждый пакет всем соседям, пакет направляют единственному соседу, выбранному случайным образом из таблицы соседей. Алгоритм распространения слухов преодолевает проблему дублирования сообщений, но при этом увеличивается задержка при передаче сообщений, что представляет собой существенную проблему для неиерархических алгоритмов.

- **Иерархические протоколы маршрутизации для БСС.**

В иерархических алгоритмах маршрутизации узлы самоорганизуются в кластеры и головной узел выбирается для каждого кластера. Головные узлы осуществляют сбор данных с узлов – членов кластера, производят их обработку и передачу информации на шлюз или базовую станцию. Такое агрегирование данных в головных узлах значительно уменьшает энергопотребление в сети и увеличивает длительность жизненного цикла. Главным вопросом в разработке алгоритмов для иерархических беспроводных сенсорных сетей является выбор головного узла кластера. Существует два способа выбора головного узла кластера: выбор случайного головного узла или выбор предопределённого головного узла. Рассмотрим далее алгоритмы случайного выбора головного узла. Случайный выбор головных узлов приводит к созданию кластеров различных размеров. В иерархических алгоритмах маршрутизации и самоорганизации каждый из членов кластера имеет возможность стать головным узлом, головной узел задаёт расписание для членов кластера и т.д.

– **Алгоритм адаптивной кластеризации *LEACH***

Иерархический протокол адаптивной кластеризации с низким потреблением энергии *LEACH* (*Low-Energy Adaptive Clustering Hierarchy*) [40] является одним из первых иерархических протоколов и предполагает обеспечение баланса расхода энергии, Жизненный цикл сети состоит из этапа формирования кластера и этапа передачи собранной информации шлюзу.

Формирование кластера (*Set-up phase*).

В фазе формирования кластера каждый сенсорный узел генерирует случайное число от 0 до 1. Каждый сенсорный узел имеет порог, который соответствует предварительно определённому числу головных сенсорных узлов в сети. Если интегрированное случайное число меньше, чем порог, то сенсорный узел может стать головным в текущем раунде жизни БСС, в про-

тивном случае этот узел остаётся только членом кластера. Вычисление порога является ключевой задачей при реализации алгоритма *LEACH*.

$$Th(LEACH) = \begin{cases} \frac{p}{1 - p \times \left(r \times \text{mod} \frac{1}{p}\right)} & \text{if } n \in G \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

В (2.1) P – предопределенный процент головных узлов среди всех сенсорных узлов. Оптимальное значение P оценивается в 5% от общего числа сенсорных узлов.

Переменная r соответствует текущему интервалу функционирования сенсорной сети, G обозначает число сенсорных узлов, которые не были выбраны головными за последние $1/P$ интервалов. Кроме того, при выборе головного узла другие сенсорные узлы определяют, к какому кластеру они хотят присоединиться на основе мощности получаемого сигнала *RSS (Received Signal Strength)* от головного узла.

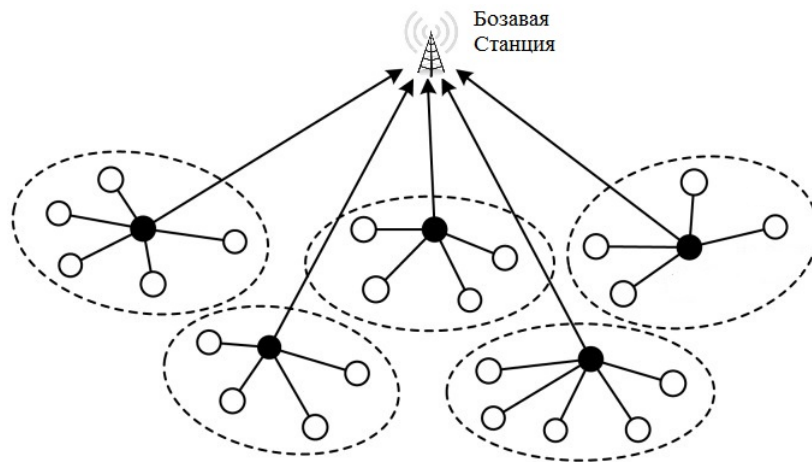


Рис. 2.4 – Топология LEACH

Когда все узлы организовались в кластеры, головной узел создает расписание передачи информации на основе метода *TDMA*, что гарантирует отсутствие коллизий при передаче сообщений.

Передача данных (*Steady-state phase*)

Головной узел широкополосным способом рассылает расписание передачи и запрашивает своих членов кластера о передаче данных. Узлы переда-

ют данные в отведенные для этого интервалы *TDMA*. После получения сообщений от всех узлов головной узел формирует свои сообщения и передает эти сообщения на шлюз или базовую станцию.

Для распределения энергетической нагрузки по сети головные узлы периодически переизбираются, и каждый из членов кластера имеет возможность стать головным узлом.



Рис. 2.5 – Блок-схема *LEACH* протокола

Алгоритм *LEACH* представляет собой распределенный подход к кластеризации сети и не требует глобальной информации о ней. Впоследствии алгоритм *LEACH* был неоднократно модифицирован и появились такие алго-

ритмы, как *TL-LEACH* [65], *E-LEACH* [35], *M-LEACH* [74], *LEACH-C* [42], *V-LEACH* [92], *LEACH-FL* [80], *W-LEACH* [16], *T-LEACH* [47] и т.д.

Преимущества алгоритма *LEACH* состоят в следующем [58]:

- 1) Любой сенсорный узел, который являлся головным узлом *CH* в текущем раунде, не может быть выбран в качестве *CH* снова, так что для каждого узла нагрузка в виде роли головного распределяется более или менее равномерно;
- 2) Используемый метод *TDMA* и расписание передачи позволяют избежать ненужных коллизий *CH*;
- 3) Члены кластера могут находиться в спящем режиме, чтобы избежать излишнего потребления энергии.

Тем не менее, существуют и несколько недостатков в алгоритме *LEACH*:

- 1) Он выполняет только прямую передачу данных внутри кластера и непосредственно из головного узла в БС. Это не всегда возможно в связи с большим размером сети. Кроме того, при большом расстоянии между головными узлами и БС потребляется много энергии;
- 2) Несмотря на ротацию головных узлов *CH* в каждом раунде, чтобы добиться балансировки нагрузки, *LEACH* не может обеспечить реальную балансировку в случае сенсорных узлов с различным количеством начальной энергии, поскольку головные узлы *CH* избираются в терминах вероятностей без энергетических соображений;
- 3) Так как выборы *CH* выполняются в терминах вероятностей, трудно равномерно распределить *CH* всей сети. Таким образом, существуют избранные *CHS*, сосредоточенные в одной части сети, и некоторые узлы, в окрестностях которых нет *CHs*;
- 4) Идея динамической кластеризации, т.е. выбор головных узлов в каждом раунде, приносит дополнительную нагрузку.

– *TEEN (Threshold-sensitive Energy Efficient Protocols)*

TEEN [67] алгоритм базируется на методе кластеризации *LEACH*. Каждый узел в кластере периодически становится головным узлом. Сеть с использованием алгоритма *TEEN* также имеет иерархическую структуру на основе кластерной организации. Головной узел может назначать своим узлам «жесткий» (*hard*) и «мягкий» (*soft*) пороги:

- Жесткий порог (*Hard Threshold*): Узел посылает информацию головному узлу только, если количество накопленных данных находится в заданных пределах;
- Мягкий порог (*Soft Threshold*): узел посылает информацию головному узлу только, когда количество накопленных данных изменилось как минимум на величину, равную или большую, чем мягкий порог.

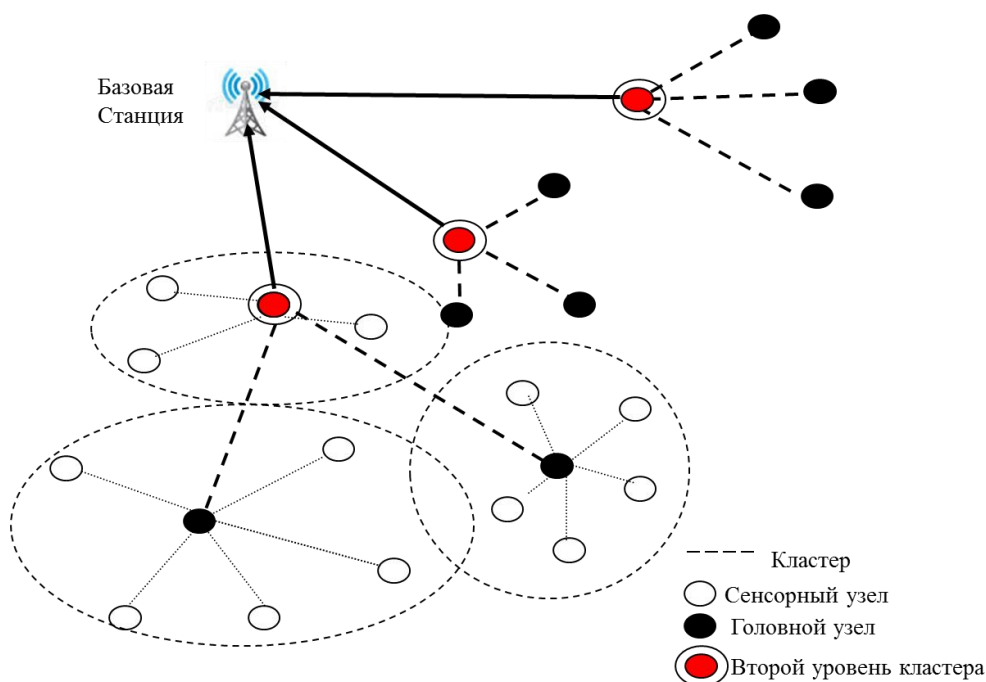


Рис. 2.6 – Основная топология алгоритма *TEEN*

Согласно сказанному выше, *TEEN* имеет следующие преимущества:

- 1) Жесткий порог (*HT*) сокращает количество сообщений, передавая информацию только тогда, когда собираемые данные находятся в диапазоне интереса. Кроме того, мягкий порог уменьшает количество сообщений ис-

ключением тех, у которых есть минимальное изменение в собираемых данных. Таким образом, алгоритм *TEEN* уменьшает потребление энергии и улучшает эффективность и полезность БСС в целом;

2) *TEEN* оперативно реагирует на большие изменения в собираемых данных, что подходит для реактивных сценариев и критически важных приложений.

Тем не менее, существуют несколько недостатков и в алгоритме *TEEN*:

1) Алгоритм *TEEN* не подходит для приложений БСС с периодическим сбором данных, так как пользователь не может получить никаких данных, если значения атрибутов не могут достичь порога;

2) Существуют потраченные впустую тайм-слоты и вероятность того, что БС может быть не в состоянии отличить погибшие узлы от живых, потому что передача может осуществляться только тогда, когда данные превосходят жесткий порог или текущее значение для мягкого порога отличается на величину, равную или большую, чем сам мягкий порог;

3) Если головные узлы *CH* находятся вне радиуса связи друг с другом, данные могут быть потеряны, потому что распространение информации осуществляется только посредством головных узлов *CH*.

– *SEP (Stable Election Protocol)*:

Алгоритм *SEP* [84] разработан для выбора головного узла в БСС исходя из предположения, что в сети возможно наличие двух типов узлов с точки зрения их энергии: обычные и продвинутые узлы. Последнее сразу же определяет, что в данном случае сеть является гетерогенной. Энергия продвинутых узлов больше, чем энергия обычных. При этом для обычных узлов начальная энергия равна E_0 , в то время как для продвинутых $E_0 (1 + \alpha)$.

Предположим, что мы имеем n узлов в сети, а процент продвинутых узлов равен m . Тогда:

$$n \cdot (1 - m) \cdot E_0 + n \cdot m \cdot E_0 \cdot (1 + \alpha) = n \cdot E_0 \cdot (1 + \alpha \cdot m) \quad (2.2)$$

– *DEEC (Distributed Energy Efficient Clustering)*

В [76] авторы предложили новый распределенный энергосберегающий алгоритм кластеризации для гетерогенных беспроводных многоуровневых сенсорных сетей, который назвали *DEEC*.

DEEC оценивает вероятность выбора головного узла в многоуровневых гетерогенных БСС в соответствии со следующей формулой:

$$p_i = p_{opt} \left[1 - \frac{E(r) - E_i(r)}{E(r)} \right] = p_{opt} \frac{E_i(r)}{E(r)} \quad (2.3)$$

При этом в алгоритме *DEEC* средняя энергия сети $E(r)$ для произвольного раунда r рассчитывается как:

$$E(r) = \frac{1}{N} E_{total} \left(1 - \frac{r}{R} \right) \quad (2.4)$$

R обозначает общее количество раундов жизни сети и оценивается следующим образом:

$$R = \frac{E_{total}}{E_{round}} \quad (2.5)$$

где E_{total} является суммарной энергией в сети, а E_{round} – это расход энергии в течение раунда.

2.3. Сравнительный анализ алгоритмов *DT*, *LEACH*, *SEP*, *DEEC*, *TEEN*

При сравнительном анализе алгоритмов маршрутизации и самоорганизации для гомогенных и гетерогенных БСС использовались следующие показатели:

- **Жизненный цикл.** Жизненный цикл БСС может быть определен с помощью следующих показателей:

- длительность интервала времени от начала функционирования БСС до момента гибели первого сенсорного узла,
- длительность интервала времени до момента времени, когда в живых остается заданный процент сенсорных узлов,

– длительность интервала времени до гибели последнего сенсорного узла.

- **Стабильность.** Длительность интервала времени от начала функционирования БСС до момента гибели первого сенсорного узла.

- **Нестабильность.** Длительность интервала времени от момента гибели первого сенсорного узла до момента времени гибели последнего.

- **Число живых узлов и число погибших узлов.**

Сравнительный анализ алгоритмов маршрутизации и самоорганизации маршрутизации *DT*, *LEACH*, *SEP*, *DEEC* и *TEEN* проводился с использованием *MATLAB* и типовых параметров сенсорных сетей, при которых, как правило, проводится моделирование (таблица 2.1 и таблица 2.2). В таблице 2.2 приведены значения констант для алгоритма *TEEN*. Радио модель соответствует модели Кся-Бертони [91].

Таблица 2.1. – Параметры моделирования

Обозначение	Значение	Размерность
N	100	Шт
P	5	%
Размеры поля (X, Y)	(100,100)	М
Расположение базовой станций (x, y)	(100,150)	М
E_s	0,5	Дж
E_{elec}	70	нДж
Постоянное усиление ε_{fs}	10	пДж/бит/м ²
Мультисетевая постоянная ε_{mp}	0,0013	пДж/бит/м ⁴
Размер пакета	4096	Бит
Энергия сбора данных одного бита E_{DA}	5	нДж
Потери на участках (экспоненциальные)	87	М

Таблица 2.2. – Начальные значения при моделировании для алгоритма *TEEN*.

Обозначение	Значение
Жесткий порог(<i>Hard Threshold</i>)	100
Мягкий порог (<i>Soft Threshold</i>)	2

При сравнительном анализе использовалось несколько сценариев построения БСС.

2.3.1. Первый сценарий.

В первом сценарии сенсорные узлы случайно распределены на плоскости размером 100м*100м. Число узлов в сети составляет 100 и базовая станция находится на границе сенсорного поля с координатами (150, 50) м (см. Рисунок 2.7).

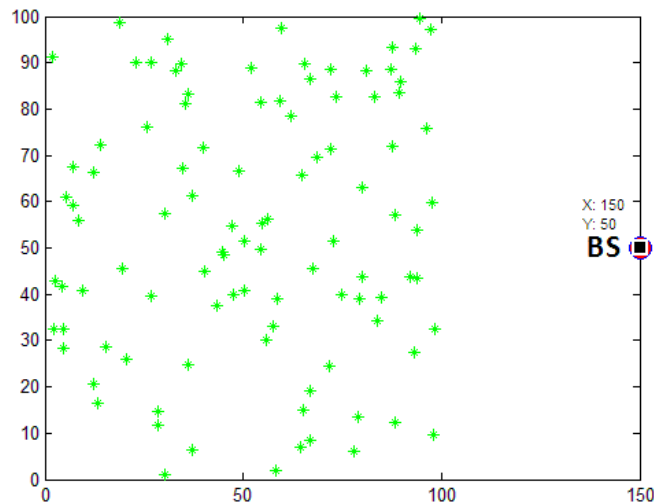


Рис. 2.7 – Случайное расположение узлов

В этом сценарии сравним алгоритм прямой передачи (*DT*) и алгоритмы иерархической маршрутизации *LEACH* и *TEEN* для гомогенных (однородных) сенсорных сетей. На рисунках 2.8 и 2.9 приведены характеристики для алгоритма прямой передачи в начале жизненного цикла и после 180 раундов.

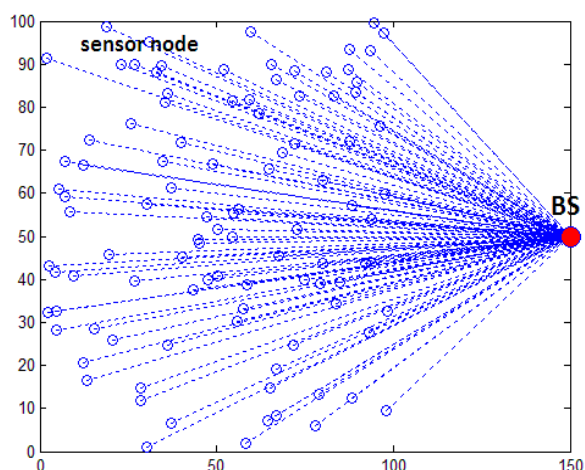


Рис. 2.8 – Прямая передача. Начальное состояни.

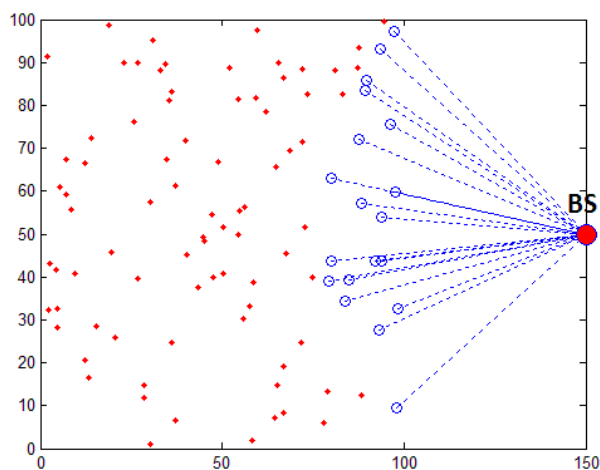


Рис. 2.9 – Состояние после 180 раундов

При использовании алгоритма прямой передачи *DT* сенсорный узел передает данные непосредственно на базовую станцию независимо от расстояния. Как видим, при этом узлы, которые находятся достаточно далеко от базовой станции, гибнут первыми. Поэтому, эффективное применение *DT* целесообразно для сенсорных полей небольшого размера.

На рисунке 2.10 показана агрегация сенсорных узлов в кластеры при использовании алгоритмов маршрутизации *LEACH* и *TEEN*, а рисунок 2.11 иллюстрирует передачу информации от головных узлов на базовую станцию.

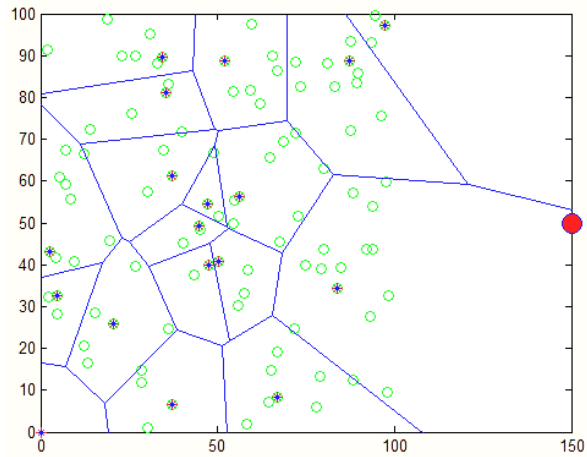


Рис. 2.10 – Кластеризация БСС

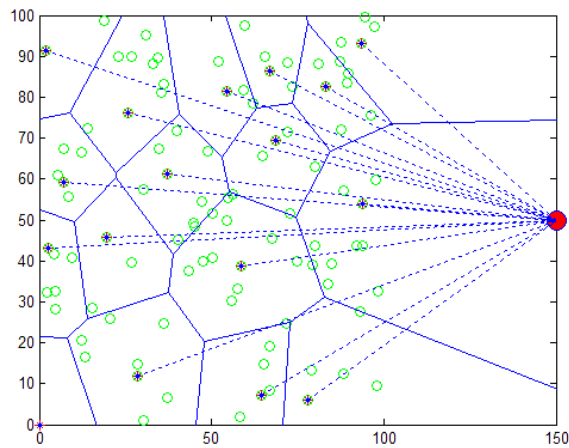


Рис. 2.11 – Случайное расположение узлов

По результатам моделирования установлено, что алгоритм *DT* имеет меньшие период стабильности сети и жизненный цикл. Первый узел для алгоритма *DT* погиб уже в 154 раунде, тогда как для алгоритма *LEACH* это произошло в 621 раунде, а для алгоритма *TEEN* в 1130 раунде.

На рисунке 2.12 показана длительность периода стабильности для всех указанных выше алгоритмов, а на рисунке 2.13 – зависимость остаточной энергии сети от числа раундов.

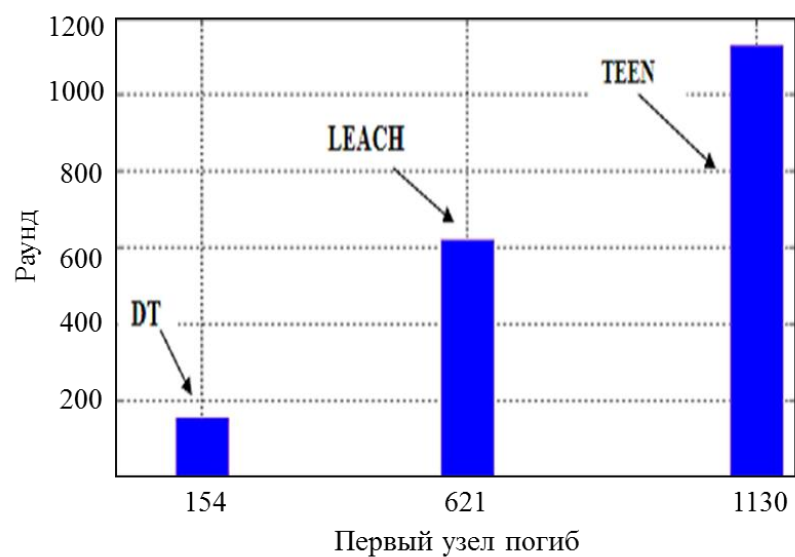


Рис. 2.12 – Длительность периода стабильности

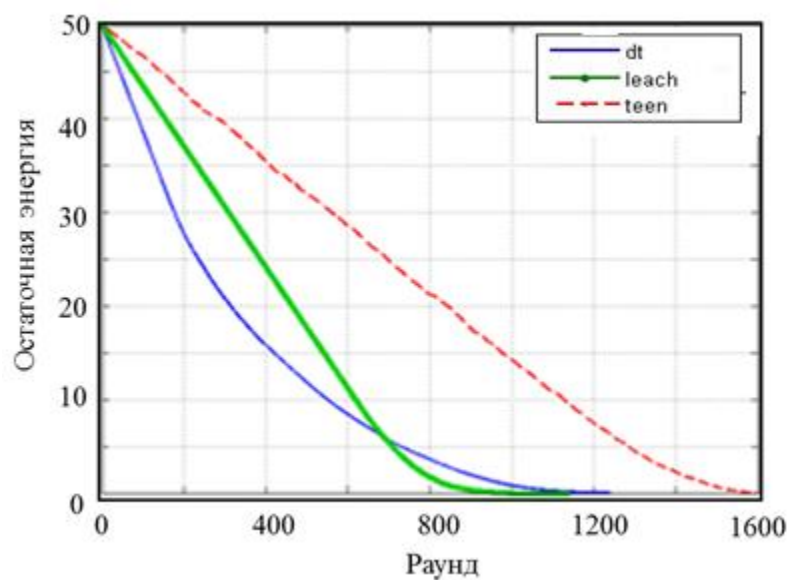


Рис. 2.13 – Остаточная энергия

Таким образом, алгоритмы кластеризации существенно эффективнее алгоритма прямой передачи. Для алгоритма *TEEN* длительность периода стабильности более, чем в 7 раз, а для алгоритма *LEACH* в 5 раз больше, чем для алгоритма прямой передачи *DT*. На рисунке 2.14 и рисунке 2.15 приведены зависимости числа живущих и погибших узлов соответственно.

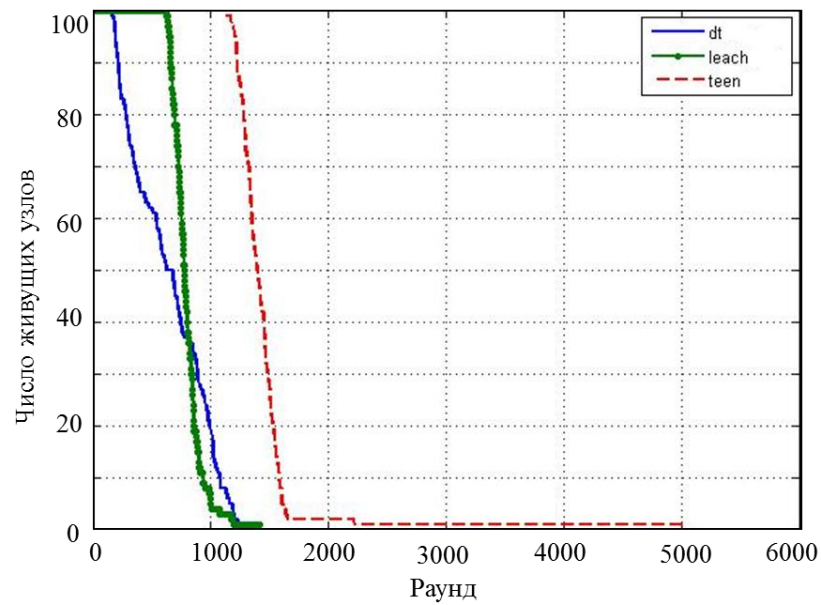


Рис. 2.14 – Число живущих узлов

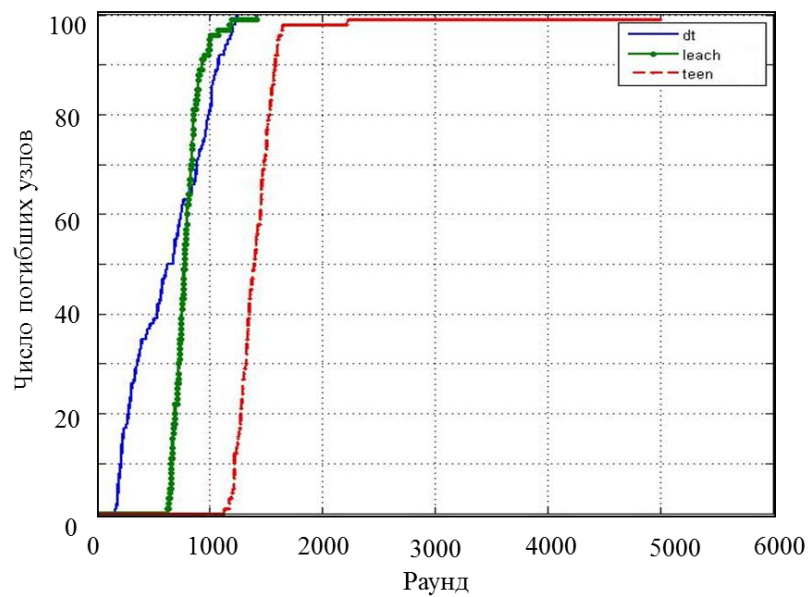


Рис. 2.15 – Число погибших узлов

2.3.2. Второй сценарий.

В этом сценарии рассмотрим гетерогенную сенсорную сеть, в которой есть два типа узлов – обычные и продвинутые. Предлагается новая модель, отличающаяся от известных тем, что начальная энергия для продвинутых узлов в два раза больше, чем для обычных. При этом их число составляет 10% от общего числа узлов на сенсорном поле. Сравним эффективность применения алгоритмов *DT*, *LEACH*, *TEEN* и *SEP* для модели гетерогенной сети. На

рисунке 2.16 показано распределение обычных узлов (кружки, *Normal node*) и продвинутых узлов (+, *Advanced node*) случайным образом на сенсорном поле.

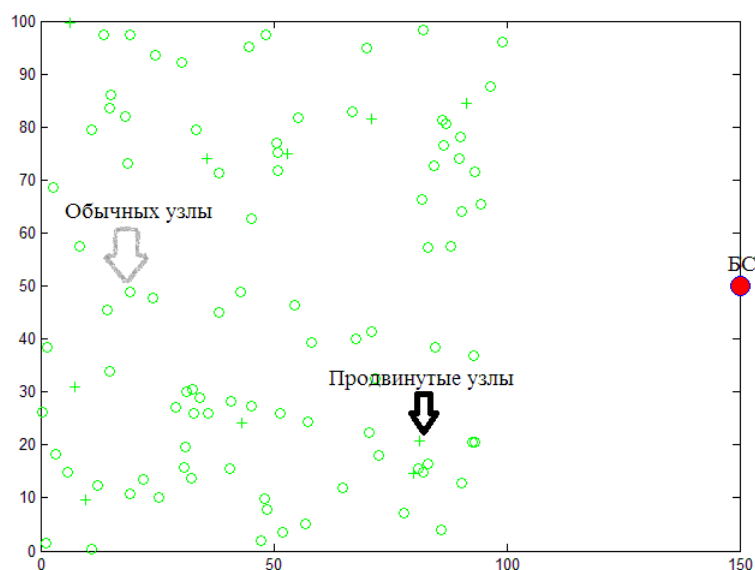


Рис. 2.16 – Случайное размещение узлов в гетерогенной сети

Результаты моделирования приведены на рисунках 2.17, 2.18, 2.19, 2.20. Анализ результатов показывает, что период стабильности для алгоритма *SEP*, который изначально учитывает наличие в сети гетерогенности, существенно больше по сравнению с алгоритмами *LEACH* и *DT*. Вместе с тем, последний узел при использовании алгоритма *LEACH* погибает позднее, чем для алгоритма *SEP*. Алгоритм *TEEN*, хотя и не учитывает энергетическое неравенство узлов, дает лучшие результаты по всем показателям, чем алгоритмы *SEP*, *LEACH* и *DT*.

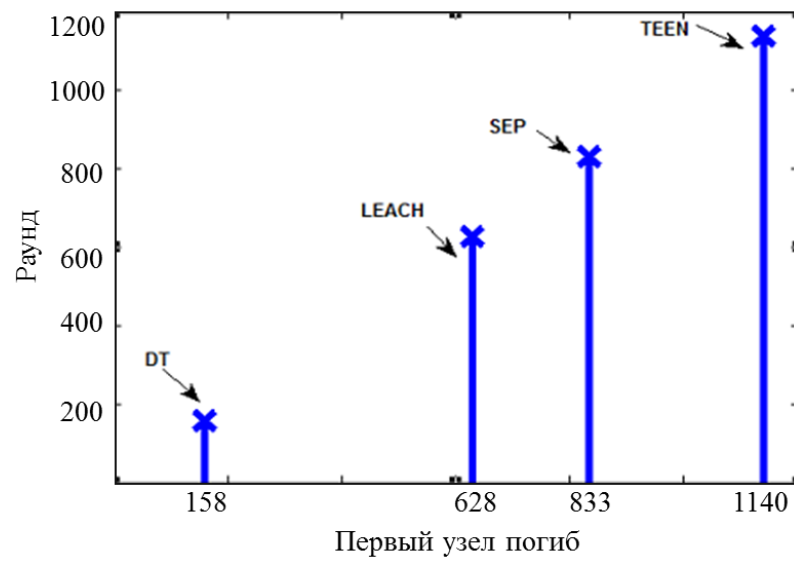


Рис. 2.17 – Длительность периода стабильности

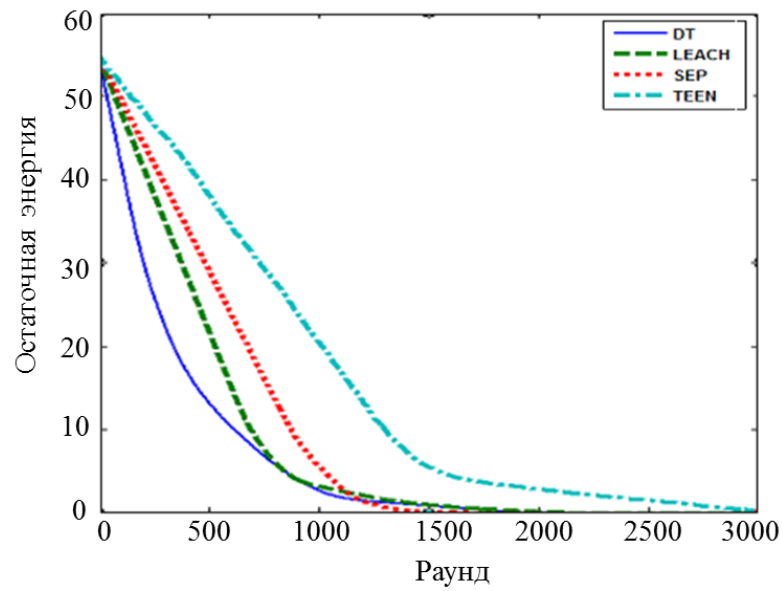


Рис. 2.18 – Остаточная энергия.

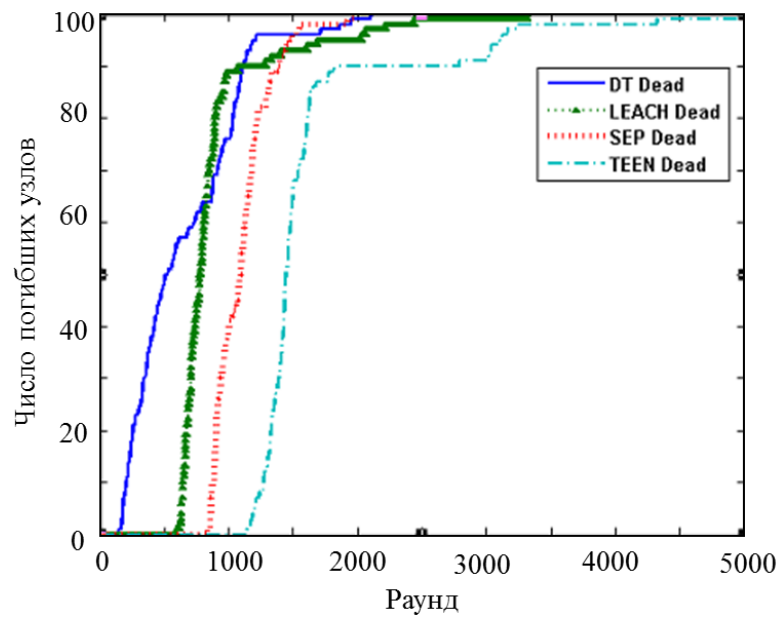


Рис. 2.19 – Число живущих узлов

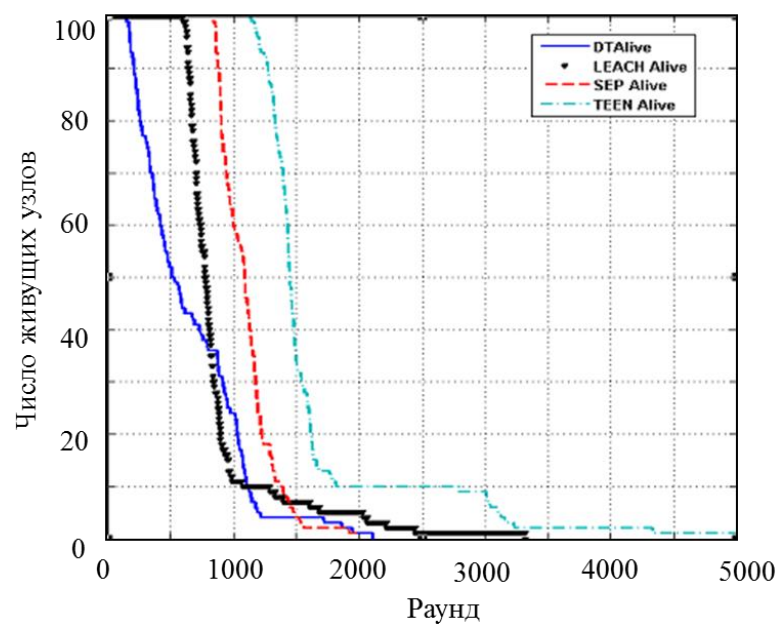


Рис. 2.20 – Число погибших узлов

2.3.3. Третий сценарий.

В этом сценарии рассмотрим многоуровневые по возможным энергетическим характеристикам сенсорных узлов гетерогенные сети. При этом гетерогенная сенсорная сеть содержит сенсорные узлы трех уровней: обычные, продвинутые и суперузлы. каждый узел имеет свою первичную энергию, значение первичной энергии изменяется в рамках от 0,1 до 1дж.

Результаты моделирования приведены на рисунках 2.21 и 2.22. Анализ результатов моделирования показывает, что и по длительности периода стабильности, и по остаточной энергии алгоритм *DEEC*, изначально специализированный для многоуровневых гетерогенных сетей, существенно эффективнее, чем алгоритмы *LEACH* и *DT*. Вместе с тем, алгоритм *TEEN* и для случая многоуровневых гетерогенных сенсорных сетей лучше, чем специализированный для этих целей алгоритм *DEEC*, и по длительности периода стабильности, и по остаточной энергии.

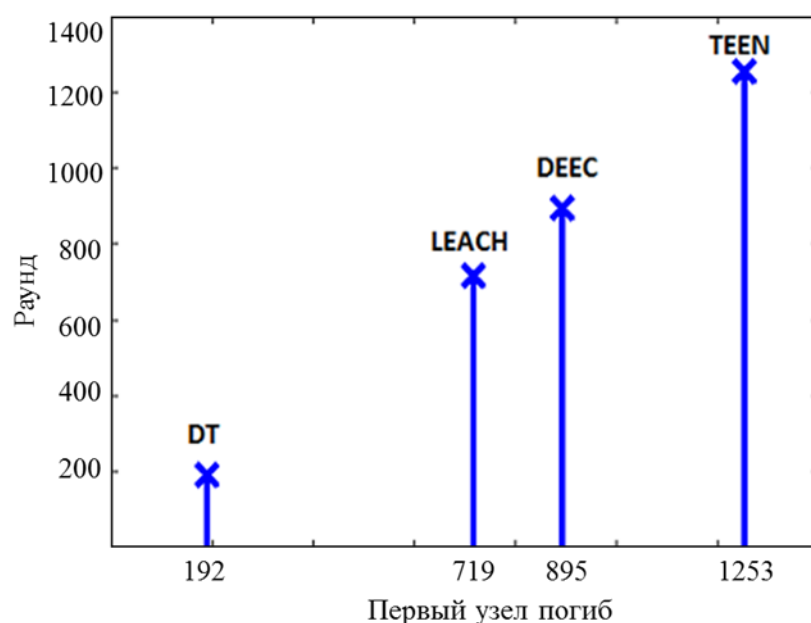


Рис. 2.21 – Длительность периода стабильности

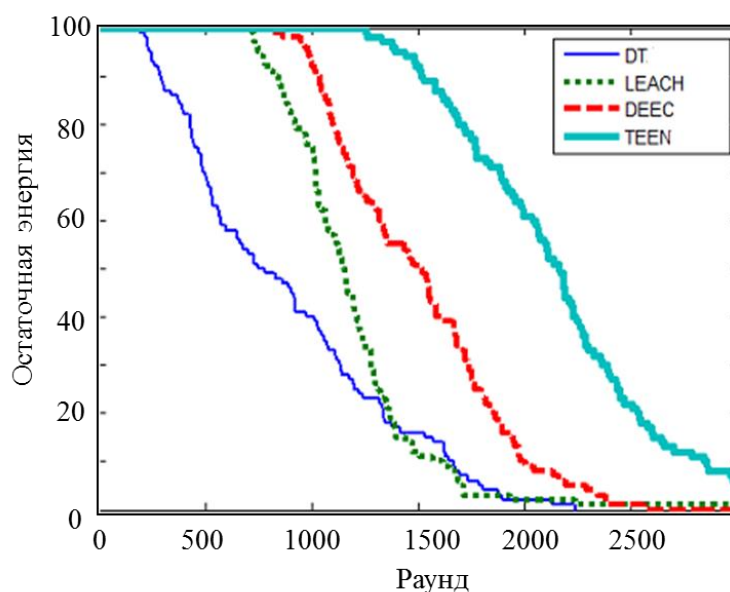


Рис. 2.22 – Остаточная энергия

Результаты моделирования для всех сценариев обобщены в таблице 2.3 и на рисунке 2.23. Анализ результатов моделирования показывает, что независимо от вида сети – гомогенная, двухуровневая гетерогенная или многоуровневая гетерогенная – алгоритм *TEEN* обеспечивает наибольшее значение длительности периода стабильности. Отсутствие кластеризации при использовании алгоритма прямой передачи приводит к тому, что длительность жизненного цикла БСС может уменьшиться более, чем в 7 раз по сравнению с применением алгоритмов кластеризации.

Таблица 2.3. – Длительность периода стабильности.

Алгоритмы маршрутизации и сценарии	DT	LEACH	TEEN	SEP	DEEC
Первый сценарий (одноуровневые БСС)	53	620	1129	----	822
Второй сценарий (двухуровневые БСС)	58	628	1139	833	856
Третий сценарий (многоуровневые БСС)	2	719	1253	----	895

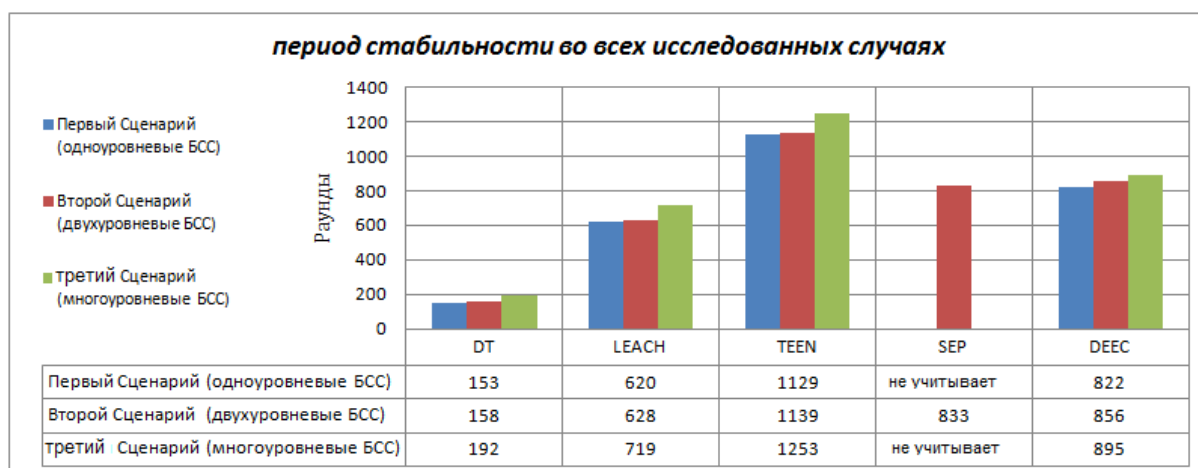


Рис. 2.23 – Длительность периода стабильности во всех исследованных случаях

Выводы

В главе рассмотрены основные алгоритмы маршрутизации и самоорганизации для гомогенных и гетерогенных беспроводных сенсорных сетей со стационарными сенсорными узлами, размещенными на плоскости. По результатам моделирования проведен сравнительный анализ эффективности применения алгоритмов *DT*, *LEACH*, *SEP*, *TEEN* и *DEEC* для различных сенсорных сетей (гомогенные, гетерогенные двухуровневые, гетерогенные многоуровневые). По результатам исследований можно сделать следующие выводы:

1. Использование неиерархических алгоритмов маршрутизации относительно эффективно для сенсорных полей небольшого размера. Для больших сенсорных сетей использование неиерархических алгоритмов нецелесообразно, поскольку требует больших энергозатрат и не обеспечивает более или менее равномерного жизненного цикла для сенсорных узлов.

2. Сравнивая одноуровневые и иерархические алгоритмы маршрутизации, можно отметить, что иерархические алгоритмы представляют больше возможностей по реализации различных приложений БСС

3. Иерархические алгоритмы маршрутизации увеличивают длительность жизненного цикла сети и стабильность, что повышает надежность функционирования БСС. Иерархическая маршрутизация позволяет гибко решать различные задачи с учётом возможностей сенсорных узлов. При этом, узлы с высокой энергоёмкостью (возможно с постоянным энергоснабжением) могут агрегировать данные и передавать их в сеть связи общего пользования, в то время как узлы с низкой энергоёмкостью могут использоваться исключительно для сбора данных.

4. Базовый алгоритм *LEACH* - крайне эффективный протокол для гомогенных БСС. Он помогает снизить энергозатраты в 7 раз по сравнению с прямым взаимодействием сенсорных узлов. В то же время, несмотря на ротацию головных узлов *CH* в каждом раунде, чтобы добиться балансировки нагрузки, *LEACH* не может обеспечить реальную балансировку в случае сенсорных узлов с различным количеством начальной энергии, поскольку головные узлы *CH* избираются в терминах вероятностей без энергетических соображений. Кроме того, так как выборы *CH* выполняется в терминах вероятностей, трудно равномерно распределить *CH* всей сети. Таким образом, существуют избранные *CH*, сосредоточенные в одной части сети, и некоторые узлы, в окрестностях которых нет *CH*.

5. Алгоритмы *SEP* и *DEEC* превосходят по длительности периода стабильности и остаточной энергии алгоритмы *DT* и *LEACH* в условиях применения гетерогенных БСС.

6. Алгоритм *TEEN*, являющийся развитием алгоритма *LEACH*, превосходит по длительности периода стабильности и остаточной энергии все рассмотренные алгоритмы и рекомендуется к использованию как в гомогенных, так и в гетерогенных БСС.

3. АДАПТИВНЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ ДЛЯ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ С МОБИЛЬНЫМИ УЗЛАМИ

3.1. Мобильные Беспроводные Сенсорные Сети (MWSN).

Традиционные Беспроводные Сенсорные Сети разработаны с использованием статических узлов (SNS) [17, 38, 82]. Эти сети используются в различных областях, таких как здравоохранение [90, 86], военная, промышленность, мониторинг окружающей среды, слежение за целью на основе мультимедийных датчиков [29] и других [75, 85, 20]. Множество исследовательских работ посвящено проблемам БСС со стационарными узлами. Тем не менее, передовые технологии в Интернет вещей [21, 64] включают в себя применение более сложных приложений, которые требуют мобильности их узлов [94]. Мобильность увеличивает число приложений БСС [66]. Она также может продлить срок службы узлов, так как для передачи данных между двумя узлами обычно не используются те же ретранслирующие узлы в маршруте. Кроме того, мобильность может улучшить связность сети, так как мобильные узлы (MN) могут помочь связать два изолированных узла от других стационарных узлов [36]. Мобильность позволяет также увеличить покрытие [61, 77]. Однако мобильность может вызвать и некоторые проблемы, например, выход мобильного узла за пределы радиуса связи в процессе передачи сообщений приводит к потере данных. Для мобильных БСС более сложными становятся вопросы управления ресурсами, топологией, протоколы маршрутизации, сложнее обеспечить необходимый уровень качества обслуживания и безопасности.

Сенсорные узлы БСС довольно часто могут функционировать в условиях, когда отсутствует возможность гарантированного электропитания или даже замены батареи БСУ. Поэтому важнейшей характеристикой БСС является остаточная энергия. Мобильность – крайне полезный шаг для БСС, так как мобильные сети менее подвержены рассеиванию энергии, что приводит к

увеличению срока службы батареи и узла в целом. Эта задача достигается путем подвижности сенсорных узлов, что увеличивает долю покрытия пространства мобильной БСС.

Как правило, в сенсорных сетях узлы самоорганизуются в кластеры и головной узел выбирается для каждого из них. Головные узлы осуществляют сбор данных с узлов – членов кластера, производят их обработку и передачу информации на шлюз или базовую станцию (БС). Такое агрегирование данных в головных узлах значительно уменьшает энергопотребление в сети и увеличивает длительность жизненного цикла [79, 23, 33].

В главе представлен новый алгоритм для выбора головного узла кластера в БСС с мобильными узлами. Алгоритм основан на сочетании лучших свойств двух известных существующих алгоритмов – *DCA (Distributed Clustering Algorithm)* [52] и *MBC (Mobility-Based Clustering)* [31]. Новый алгоритм является адаптивным алгоритмом кластеризации для БСС с мобильными узлами, что отражено и в его названии *MACA (Mobility Adaptive Clustering Algorithm)*. Алгоритм *MACA* предполагается использовать, в том числе, и для построения наземного сегмента летающих сенсорных сетей [7].

Вопросы разработки алгоритмов выбора головного узла в БСС уже более десяти лет занимают важное место в исследовательских работах по сетям связи. Большинство алгоритмов кластеризации разработано для БСС со стационарными узлами. Такие алгоритмы как *LEACH* [44], *LEACH-C* [43], *HEED* [93] и *TEEN* [68] эффективно используются для гомогенных и гетерогенных сенсорных сетей со стационарными узлами [3], что было рассмотрено в главе 2. Для всех этих алгоритмов жизненный цикл сенсорной сети делится на раунды. Каждый раунд состоит из этапа формирования кластера и этапа передачи собранной головным узлом информации шлюзу или базовой станции (БС).

Применение упомянутых алгоритмов для эффективной кластеризации в БСС с мобильными узлами не нашло широкого применения. Последнее связано с тем, что мобильность членов кластера, а что еще принципиальнее –

мобильность головного узла, приводят к нестабильности сформированного кластера в рамках одного раунда. Вместе с тем, мобильность БСУ может быть использована и для улучшения некоторых характеристик сети. Например, в [89] свойством мобильности предложено пользоваться для улучшения связности сети, а в [5, 28] – для защиты БСС от потоков ложных событий. Алгоритм *LEACH* для БСС с мобильными узлами *LEACH-M* [50] позволил улучшить характеристики сети, но не настолько как алгоритмы с предсказанием *DCA* [54, 53], *GMAC* [26].

В [51, 53] предложен распределенный алгоритм кластеризации (*DCA*), основанный на применении комбинированного критерия прогнозирования (*PCC*). В *DCA* каждый сенсорный узел вычисляет прогнозируемое значение *PCC* в соответствии с каким-либо предиктором (простым, экстраполяционным, комбинированным). Комбинированный критерий прогнозирования включает в качестве параметров связность, покрытие, мобильность и остаточную энергию. Значение *PCC* передается всем близлежащим узлам в радиусе кластера и головной узел выбирается по его максимальному значению.

В [26] предложен алгоритм для кластеризации групп мобильных устройств *GMAC* с использованием информации о топологии сети и предсказании направления движения БСУ. Это предсказание позволяет определить вместе или раздельно перемещаются отдельные сенсоры и группы БСУ. Число кластеров в *GMAC* фиксировано. В статье предложены оценки для определения оптимального размера группы и, соответственно, числа кластеров. Головной узел кластера выбирается на основе остаточной энергии и группы мобильности, к которой принадлежит БСУ.

В [83] предложен алгоритм кластеризации, позволяющий использовать временные интервалы *TDMA* (в случае их свободности) не только членам своего кластера, но и членам другого кластера, оказавшимся вследствие мобильности проходящими через поле данного кластера.

В алгоритме *MBC* головной узел выбирается традиционным способом – по значениям остаточной энергии и мобильности. Что касается члена класте-

ра, то он выбирает наиболее подходящий для себя головной узел на основе так называемого значения пригодности, которое может гарантировать стабильное взаимодействие с головным узлом кластера. Последнее позволяет уменьшить потери пакетов и одновременно снизить потребление энергии из-за менее частого изменения членства в конкретном кластере.

С учетом результатов, полученных в [54, 53], при разработке нового алгоритма МАСА используем выбор головного узла на базе комбинированного критерия прогнозирования с применением простого точечного предиктора и вступление в члены кластера по значениям пригодности из алгоритма МВС.

3.1.1. Различные случаи мобильности

Мобильные беспроводные сенсорные сети (MWSNs) являются частным классом БСС, в котором мобильность играет ключевую роль в выполнении задач того или иного приложения. В последние годы мобильность в сенсорных сетях стала важной областью исследований для БСС. Мобильность в БСС можно классифицировать следующим образом:

- **Мобильный узел для сбора информации.** Специальный мобильный узел или узлы участвуют в сборе информации с сенсорных узлов, что снижает энергопотребление в процессе передачи информации и увеличивает длительность жизненного цикла за счет уменьшения средней длины маршрута для передачи информации.
- **Мобильность базовая станция.** Мобильная базовая станция может перемещаться между фрагментами сети, что так же, как и в предыдущем случае, снижает энергопотребление и увеличивает длительность жизненного цикла сети.
- **Мобильность событий.** В БСС событие, которое должно наблюдаться, также может быть мобильным (например, в задачах поиска целей), при этом различные узлы БСС становятся “ответственными” за наблюдение такого события.

- **Мобильность членов кластера и головного узла кластера.**

Мобильность членов кластера и головного узла кластера, оказывая в целом отмеченное выше положительное влияние на параметры БСС, ставит и целый ряд новых проблем для кластеризации мобильных БСС, включая проблемы устойчивости кластера.

3.1.2. Модели мобильности

Модели мобильности могут быть разделены на следующие категории: (I) случайная мобильность, (II) предсказуемая мобильность и (III) управляемая мобильность. Различные категории моделей мобильности указывают на разнообразие алгоритмов сбора данных, используемых в каждом случае. Примером случайной мобильности являются сенсорные узлы, установленные на животных для сбора данных в дикой природе. В такой модели очень трудно контролировать и прогнозировать направление движения узлов.

В предсказуемой модели мобильности ожидаемые пути перемещения сенсорных узлов прогнозируются с помощью различных методов, при этом сенсорные узлы входят в спящий режим до предсказанного времени для передачи данных. Контролируемая мобильность предполагает схему сбора данных с поиском максимально короткого пути (*MAPS*) для увеличения продолжительности жизни узлов за счет оптимизации сети. Основываясь на предположении, что мобильный узел или базовая станция связываются с каждым сенсорным узлом, путь формируется подвижным планирующим элементом (*MES*).

3.2. Адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами (*MACA*)

Предлагается новый адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами под названием *MACA* (*Mobility Adaptive Clustering Algorithm*). В разработанном алгоритме кластеризации мобильный сенсорный узел выбирает себя головным узлом кластера на осно-

ве простого точечного предиктора *SPP* (*Single Point Predictor*) для комбинированного критерия прогнозирования (алгоритм *DCA*). Другие узлы представляют собой члены кластера, определяемые на основе некоторого значения, указывающего на их пригодность для подключения к конкретному головному узлу (алгоритм *MBC*). Сочетание двух известных алгоритмов обеспечивает существенное увеличение длительности жизненного цикла сети и ее стабильности. Разработанный алгоритм предназначен, в том числе, и для использования в наземном сегменте летающих сенсорных сетей.

3.2.1. Модель для исследования алгоритма *MACA*

При разработке алгоритма *MACA* была принята следующая модель сети:

- Сенсорная сеть является гомогенной, т.е. все БСУ обладают одинаковыми базовыми параметрами (начальная энергия, радиус действия);
- Топология сети изменяется, и БСУ могут перемещаться со скоростью от 0 до 2 м/с (типовые значения при исследованиях мобильных сенсорных сетей);
- БСУ осуществляют свою активность без централизованного управления;
- Базовая станция находится в стационарном состоянии и радиоканал симметричен.

Заметим также, что наибольшая эффективность алгоритма может быть достигнута, когда число покидающих кластер за время раунда беспроводных сенсорных узлов равно числу прибывающих в кластер. Существенное значение имеет выбор модели мобильности БСУ. Такие стандартные модели, как случайный выбор пути перемещения, обхода, направления движения и т.п., не гарантируют чисто случайного перемещения, поскольку поведение мобильных БСУ существенно зависит от конфигурации сети и используемого сценария ее функционирования [48].

Дальше будем использовать модель мобильности, близкую к мобильности человека в роли пешехода в реальных условиях. Отметим, что скоро-

сти перемещения узлов в сенсорных сетях с мобильными сенсорными сетями ограничиваются 2 м/с, что соответствует скорости быстро идущего пешехода. Распределение мобильных узлов по диапазону изменения скорости перемещения от 0 до 2 м/с производится случайным образом.

Для каждого из узлов при этом действует приведенный ниже алгоритм определения мобильности.

- 1) Случайным образом в соответствии с равномерным распределением выбирается скорость перемещения узла.
- 2) Случайным же образом выбирается направление перемещения.
- 3) В выбранном направлении с выбранной скоростью БСУ может перемещаться в течение заданного периода времени на определенное расстояние. После того, как достигнута заданная граница перемещения по времени или по расстоянию, выбирается новое направление перемещения.
- 4) Затем возможно некоторое время ожидания (до конца раунда).
- 5) Возвращение к первому шагу.

Для распространения электромагнитных волн также принимается стандартная модель для БСС. При этом для оценки потерь учитывается положение БСУ: находятся они в зоне прямой видимости (*LOS*) или нет (*NonLOS*) [88].

3.2.2. Комбинированный критерий прогнозирования

В соответствии с реализацией алгоритма *DCA* комбинированный критерий прогнозирования включает критерии связности, покрытия, мобильности и остаточной энергии. Комбинированный критерий прогнозирования $CC(s_i)$ для БСУ S_i в момент времени t определяется по формуле:

$$CC(s_i) = \alpha \times ConC(s_i) + \beta \times CovC(s_i) + \frac{\gamma}{1 + MC(s_i)} + \zeta \times REC(s_i), \quad (3.1)$$

В (3.1) $\alpha + \beta + \gamma + \zeta = 1$.

В [54, 53] рассмотрены три эвристических предиктора: простой точечный предиктор *SPP* (*Single Point Predictor*), линейный экстраполяционный предиктор *LEP* (*Linear Extrapolation Predictor*) и гибридный предиктор *HP* (*Hybrid Predictor*). Исследование показало, что простой предиктор *SPP* обеспечивает наибольший по длительности жизненный цикл сенсорной сети. Комбинированный критерий прогнозирования (*PCC*) должен быть предсказан в текущее время t в соответствии с историей этого критерия;

$$HCC = [(CC_1, t_1), (CC_2, t_2), \dots, (CC_n, t_n)] \quad (3.2)$$

где $t_1 < t_2 < \dots < t_n$. Простой точечный предиктор *SPP* (*Single Point Predictor*) всегда предсказывает следующее значение как предыдущую величину *HCC*:

$$PCC = SPP(t_c) = CC_n \quad (3.3)$$

3.2.3. Алгоритм МАСА

Как и большинство существующих алгоритмов, разработанный алгоритм МАСА содержит две фазы: фазу формирования кластеров, когда создается множество кластеров и расписание *TDMA*, и фазу стационарного состояния, когда головные узлы собирают информацию и передают их на шлюз или базовую станцию.

➤ Фаза формирования кластера. Выбор головного узла кластера.

С начала каждого раунда каждый БСУ вычисляет свои предикторы комбинированного критерия прогнозирования *PCC* в соответствии с простым точечным предиктором *SPP* (*Single Point Predictor*). После этого на втором шаге алгоритма вычисленное значение *PCC* передается в радиусе действия всем близлежащим БСУ.

Формирование кластера состоит из трех фаз:

1. Объявление головного кластера. После выбора головного узла кластера, БСУ передает широковещательное сообщение о том, что он стал

головным. В сообщении указывается его местоположение, доступная скорость передачи в пределах своего радиуса действия и т.п.

2. Запрос на присоединение к головному узлу. Точно так же, как и в алгоритме *МВС* [31], каждый БСУ в сети выбирает наиболее подходящий головной узел кластера для присоединения на основе значения пригодности. Его используют для указания эффективности присоединения к тому или иному кластеру конкретного БСУ. Значение пригодности может быть определено как:

$$W_{ij} = a \times \frac{E_{j-current}}{E_{max} \cdot N_{j-current}} + b \times \left(1 - \frac{d_{ij}}{R_{tran}}\right) + c \times \frac{\Delta t_{ij}}{t_{frame}} \quad (3.4)$$

Где W_{ij} – значение пригодности, присвоенное сенсорному узлу i и указывающее его пригодность для связи с головным узлом кластера j ; $E_{j-current}$ – текущая энергия головного узла кластера j ; $N_{j-current}$ – количество членов кластера j после присоединения сенсорного узла i к головному узлу кластера j ; d_{ij} – расстояние между сенсорным узлом i и головным узлом кластера j ; Δt_{ij} – расчетное время взаимосвязи между сенсорными узлами кластера i, j ; t_{frame} – длительность кадра данных.

В формуле (3.4) $a + b + c = 1$ постоянные коэффициенты a, b и c могут принимать значения из интервала $(0, 1)$ и представляют весовое значение каждого из факторов (энергии, расстояния, времени).

3. Подтверждение головного узла. После получения запроса члена кластера на присоединение головной узел кластера посылает подтверждение БСУ о его членстве в кластере и назначает ему слот времени, в течение которого этот БСУ может передавать информацию головному узлу кластера. Расписание транслируется на все БСУ в кластере.

➤ Фаза стационарного состояния.

Каждый БСУ просыпается за один временной интервал до запланированного временного интервала для передачи информации в соответствии с

расписанием *TDMA* и возвращается в спящий режим после своего временного интервала. Головной узел кластера должен сохранять свою активность постоянно, чтобы отправить сообщение на запрос данных, получить данные от БСУ, а также отправлять и получать другие необходимые для поддержания сети сообщения.

Предложенный алгоритм *МАСА* работает согласно следующему механизму:

Шаг 1: Головной узел кластера выбирается по максимуму *PCC*;

Шаг 2: Головной узел передает широкополосное оповещение о том, что он стал головным;

Шаг 3: Узлы вычисляют значения стабильности и выбирают наиболее подходящий головной узел кластера для присоединения;

Шаг 4: Головной узел формирует *TDMA* расписание;

Шаг 5: Узлы передают данные;

Шаг 6: Головной узел агрегирует данные и передает на базовую станцию;

Шаг 7: Новый раунд начинается с 1 пункта.

Поскольку рассматриваемая сенсорная сеть состоит из мобильных узлов, то в фазе стационарного состояния сети важным процессом является возможный переход БСУ из старого кластера в новый. Процесс этого перехода показан на рисунке 3.1. При этом, когда головной узел кластера не принимает данные от члена кластера, он удаляет его из расписания передачи. В то же время, если БСУ (член кластера) не получает запрос на получение данных от своего головного узла, этот узел будет пытаться передать сообщение на другой головной узел кластера. Для этого в каждом головном узле кластера предусмотрено определенное число свободных интервалов времени для передачи информации.

Кроме того, головной узел кластера и БСУ имеют информацию о примерном времени передачи информации, и можно заранее проверить будет ли БСУ оставаться в кластере, когда в его временном интервале придет запрос

на передачу информации. Если есть уверенность в том, что БСУ не останется в кластере, узел должен передать запрос на присоединение к новому кластеру и стараться избежать больших потерь пакетов, прежде чем он потеряет связь с текущим головным узлом.

На рисунке 3.1 узел 11 присоединяется к кластеру 1, в то время как узел 4 покидает кластер. Головной узел кластера 1 удаляет узел 4 из расписания *TDMA* и добавляет узел 11 в расписание *TDMA*, которое корректируется на основе расчетного времени подключения Δt между членом кластера и головным узлом кластера в порядке возрастания [22].

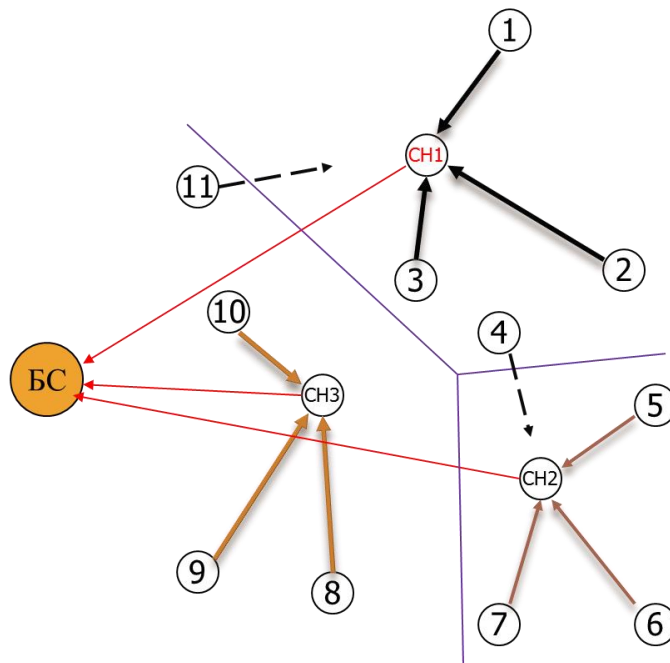


Рис. 3.1 – БСУ оставляет старый кластер и присоединяется к новому

3.3. Результаты моделирования.

Моделирование было проведено на языке *C#.NET* с использованием следующих численных характеристик (типовых) сенсорной сети. БСУ случайным образом распределены на плоскости 400м*400м, и число сенсоров изменяется от 100 до 400 с шагом 50. Параметры сети и их значения, используемые при моделировании, сведены в таблицу 3.1. Алгоритм *MACA* сравнивается с алгоритмами *LEACH-M* и *DCA* по длительности жизненного цикла

сенсорной сети, длительности периода стабильности и по числу успешно переданных пакетов. На рисунке 3.2 отображены экранные формы, показывающие процесс кластеризации.

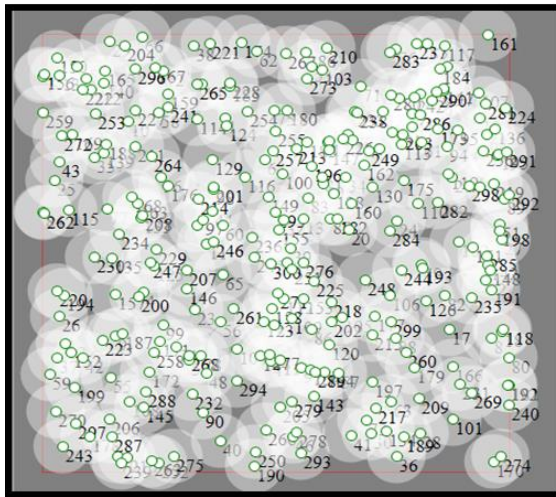
Таблица 3.1. Параметры моделирования

Параметр	Обозначение	Значение
Начальная энергия на узел	E	2 Дж
T_x/R_x	E_{elec}	50 нДж/бит
Постоянное усиление	e_{fs}	10 пДж/бит/м
Мультисетевая постоянная	e_{fs}	0,0013 пДж/бит/м
Потери на участках (экспоненциальные)	$\sqrt{\frac{\epsilon_{fs}}{\epsilon_{mp}}}$	87
Радиус сенсора	r	25м
Широкополосная зона вещания	R	$2r$
Радиус кластера	$1 - hop$	$2r$
Скорость передачи данных	9600	бит/с
Радиус связности	$2r$	50м
Размеры поля (X, Y)	(400,400)	М

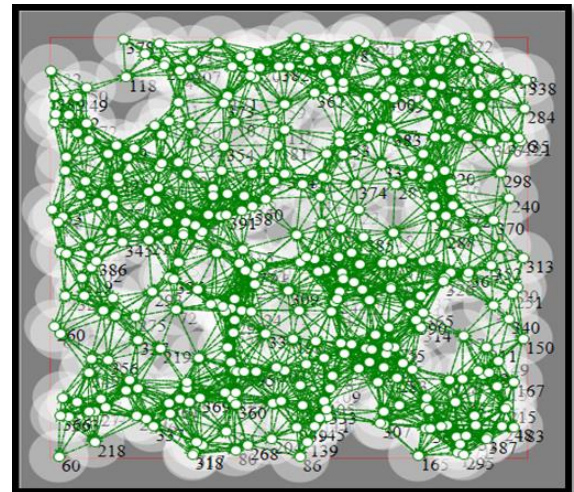
3.3.1. Длительность жизненного цикла сети.

Длительность жизненного цикла сети может быть определена одним из следующих способов:

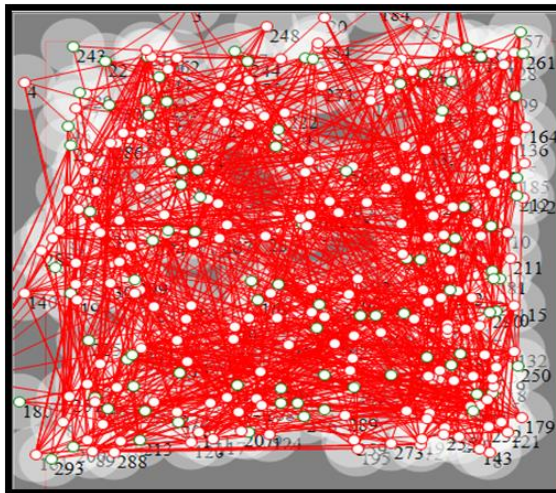
- время от начала функционирования сети до гибели первого БСУ (FND);
- время от начала функционирования сети до момента времени, когда в живых остается заданный процент БСУ (PNA);
- время от начала функционирования сети до гибели последнего БСУ [23].



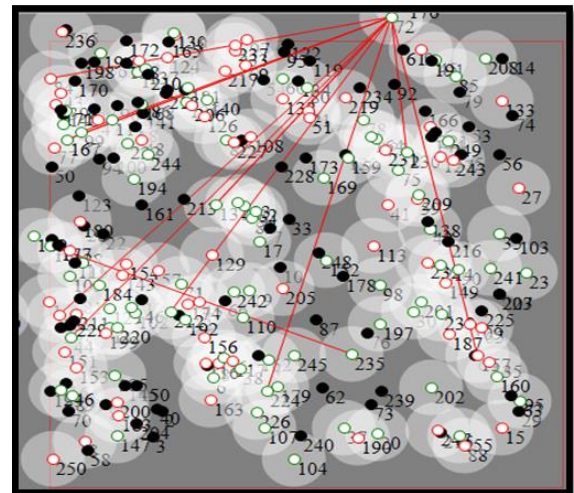
а)



б)



в)



г)

Рис. 3.2 – Экранные формы при моделировании:

а – случайное размещение БСУ; б – широкополосное оповещение РСС;
в – кластеризация; г – сенсорная сеть в процессе функционирования

На рисунке. 3.3 приведены результаты моделирования для случая, когда в живых остается 40% БСУ. Из рисунка видно, что во всем диапазоне изменения размера сети длительность жизненного цикла сенсорной сети при использовании алгоритма *MACA* существенно выше, чем для *LEACH-M* и *DCA*.

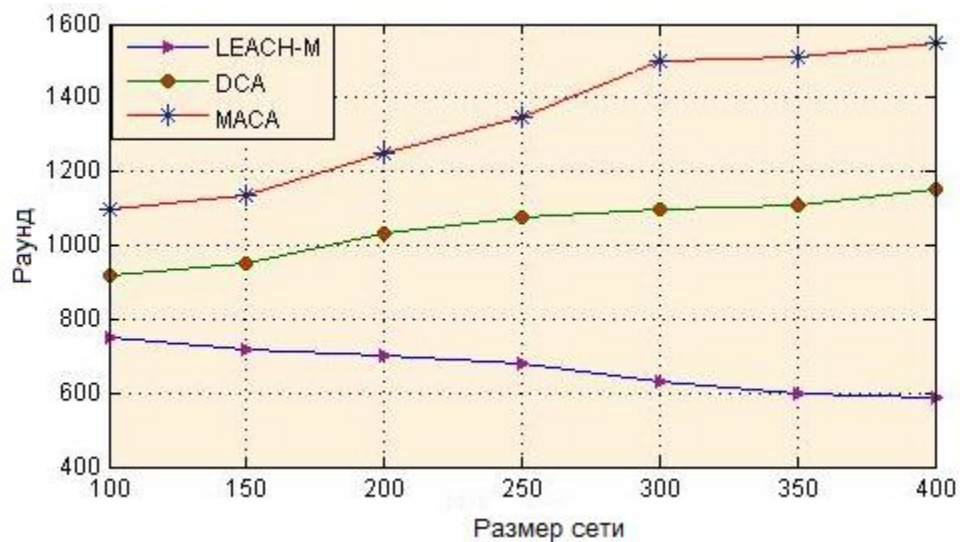


Рис. 3.3 – Длительность жизненного цикла для *MACA*, *DCA* и *LEACH-M*

3.3.2. Период стабильности.

Длительность интервала времени от начала функционирования сети до гибели первого БСУ (*FND*) называют еще периодом стабильности. На рисунке 3.4 приведены сравнительные характеристики периода стабильности для алгоритмов *MACA* и *DCA*. Алгоритм *LEACH* не выдерживает конкуренции с этими алгоритмами.

Как видно из результатов моделирования, период стабильности для нового алгоритма кластеризации больше, чем для алгоритма *DCA*. Результаты моделирования также показывают, что период стабильности для обоих алгоритмов возрастает с увеличением размера сети.

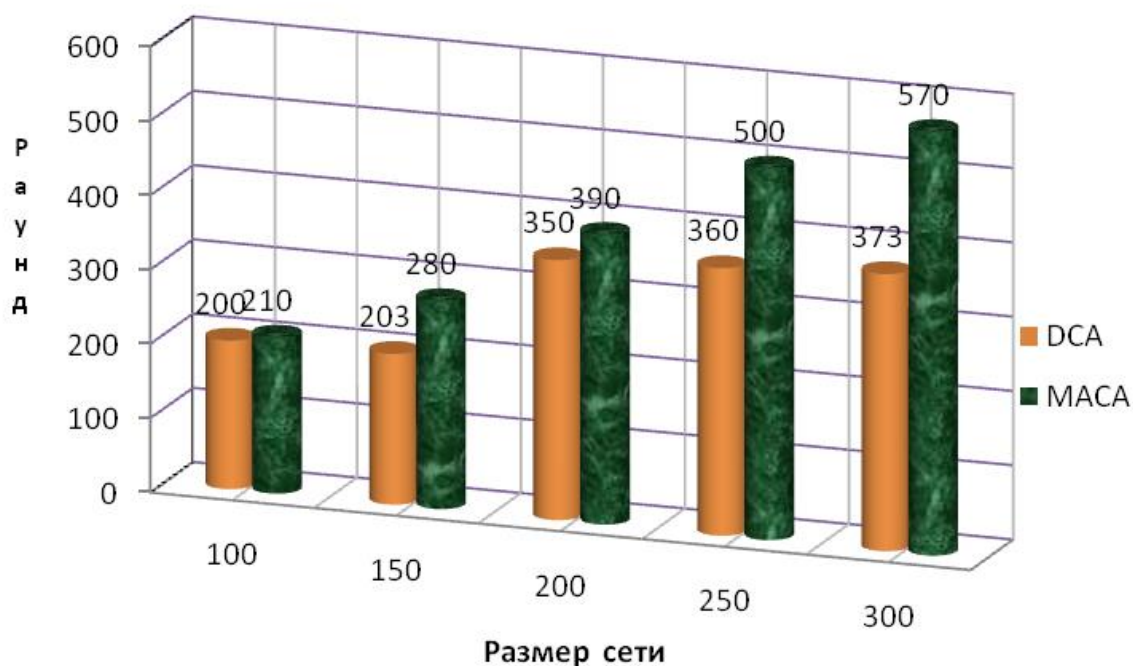


Рис. 3.4 – Период стабильности для *MACA* и *DCA*

3.3.3. Число успешно переданных пакетов.

Улучшение производительности сети, измеряемой в данном случае путем оценки числа переданных пакетов, является важным показателем для сравнения алгоритмов кластеризации. Как видно из рисунка 3.5, по значению среднего числа успешно переданных пакетов алгоритмы *MACA* и *DCA* практически равны.

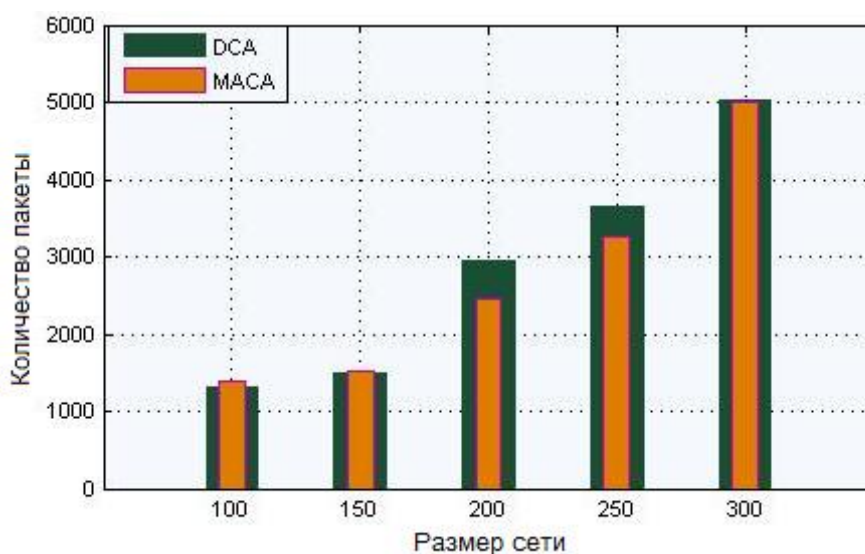


Рис. 3.5 – Среднее число успешно переданных пакетов для *MACA* и *DCA*

С учетом того, что разработанный алгоритм *MACA* превосходит по длительности жизненного цикла и периода стабильности алгоритм *DCA*, этот алгоритм рекомендуется к использованию, в том числе и для наземных фрагментов летающих сенсорных сетей [7].

Выводы.

1. Новый адаптивный алгоритм кластеризации для БСС с мобильными сенсорными узлами разработан путем сочетания положительных свойств двух известных ранее алгоритмов *DCA* и *MBC*. При выборе головного узла кластера используется комбинированный критерий прогнозирования из алгоритма *DCA*, а для присоединения членов кластера к головному узлу – значение пригодности из алгоритма *MBC*.

2. Результаты моделирования доказали, что предложенный алгоритм по длительности жизненного цикла намного превосходит широко известный алгоритм *LEACH-M*. Длительность жизненного цикла алгоритма *MACA* больше, чем и для алгоритма с прогнозированием *DCA*, который, в свою очередь, превосходит *LEACH-M*. Алгоритм *MACA* превосходит алгоритм *DCA* по длительности периода стабильности, а по числу успешно переданных пакетов оба алгоритма приблизительно равны.

4. ОТКАЗОУСТОЙЧИВЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ ДЛЯ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЕЙ

4.1. Обзор исследований в предметной области

В беспроводных сенсорных сетях (БСС) с кластерной архитектурой головные узлы кластеров играют ключевую роль в процессе функционирования сети и маршрутизации данных. Естественно, любые сбои в функционировании этих важных узлов влияют на качество услуг и надежность сети и могут привести даже к отказу сети в целом. Отказоустойчивость была определена в качестве ключевых проблем в области проектирования и эксплуатации беспроводных сенсорных сетей БСС. Отказ некоторых узлов в БСС практически неизбежен по следующим причинам: отказ аппаратных средств, энергетическое истощение, сбой ПО, ошибки связи, вредоносная атака и т.д.

В настоящее время развитие сетей связи осуществляется на основе концепции Интернета Вещей [1, 6, 14]. Технологической базой для реализации этой концепции являются беспроводные сенсорные сети (БСС) [11, 10]. БСС представляют собой самоорганизующиеся сети и состоят из множества распределенных в пространстве беспроводных сенсорных узлов (БСУ), предназначенных для мониторинга характеристик окружающей среды или объектов, расположенных в ней [12]. Ресурсы БСУ ограничены с точки зрения возможности обработки информации, пропускной способности, объема памяти, вычислительных возможностей, что существенно отличает БСС от других сетей [45]. Особенности, такие как низкая стоимость, низкое энергопотребление, компактный размер и надежность этих сенсорных узлов помогают им в том, чтобы быть используемыми для различных приложений [72]. Одно из важных приложений сенсорных сетей состоит в том, чтобы обнаружить, классифицировать и определить местонахождение определенных событий и целей. Как пример, можно привести развертывание сенсорной сети на поле боя для быстрого обнаружения танков [39]. В последние годы появилось еще одно новое применение сенсорных сетей, так называемые летающие сенсор-

ные сети [7, 8]. Объединение данных является типичной операцией во многих приложениях БСС, особенно для иерархических кластерных сетей. Кластеризация широко используется для увеличения жизненного цикла БСС, уменьшения энергопотребления, устранения избыточности данных. Существует достаточно большое число работ в области кластеризации БСС, посвященных указанным вопросам как для сенсорных полей на плоскости, так и для сенсорных сетей в трехмерном пространстве [41, 54, 1, 15, 2]. Наличие мобильности сенсорных узлов потребовало разработки специальных алгоритмов кластеризации [49, 32], в том числе и с предсказанием [55, 19, 4]. С ростом внедрения БСС все больший интерес вызывают и проблемы разработки отказоустойчивых алгоритмов [13].

Отказоустойчивость (*fault-tolerance*) в главе рассматривается на основе современных представлений ITU-T [81] как способность сети выполнять свои функции даже при наличии отказов. Отказ некоторых сенсорных узлов в БСС практически неизбежен вследствие различных причин: энергетическое истощение, отказ аппаратных средств, сбой программного обеспечения, ошибки в линиях связи, вредоносная атака и т.д. [70]. Поэтому обнаружение отказов и восстановление штатного функционирования должны быть рассмотрены для различных приложений БСС. Существует ряд работ, посвященных общим проблемам отказоустойчивости для БСС [56], алгоритмам (схемам) обеспечения отказоустойчивости в совокупности с экономией энергетических ресурсов [70, 34, 46, 25]. К сожалению, в этих решениях не используется уже накопленный положительный опыт по разработке алгоритмов кластеризации без учета отказоустойчивости. В иерархических алгоритмах кластеризации сенсорные узлы играют различную роль и появляются две категории сенсорных узлов: головной кластерный узел (*CH – Cluster Head*) и члены кластера. Головные узлы осуществляют сбор данных с узлов – членов кластера, производят их обработку и передачу информации на шлюз или базовую станцию. Такое агрегирование данных в головных узлах значительно уменьшает энергопотребление в сети и увеличивает длительность жизненного цикла и поз-

воляет достаточно просто реализовать расписание и избежать коллизий. Типичный пример иерархической кластеризации в БСС иллюстрируется на рисунке 4.1.

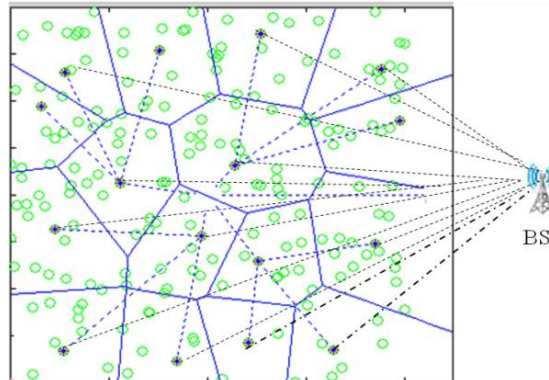


Рис. 4.1 – Кластеризация БСС.

Для БСС с гомогенными и гетерогенными стационарными сенсорными узлами на плоскости в настоящее время наиболее эффективным является алгоритм *TEEN* [22, 17]. Поэтому было решено разработать отказоустойчивый алгоритм на его основе с использованием всех положительных свойств этого алгоритма.

В главе предлагается новый отказоустойчивый алгоритм кластеризации для БСС *FT-TEEN* (*Fault-Tolerance Threshold-sensitive Energy Efficient Network algorithm*), который является модифицированной версией хорошо известного алгоритма *TEEN*. Результаты моделирования показали, что алгоритм *FT-TEEN* существенно увеличивает число успешно переданных пакетов от членов кластера к головному узлу и от головного узла к базовой станции (шлюзу) по сравнению с алгоритмом *TEEN*. Новый алгоритм предполагается использовать в наземных сегментах летающих сенсорных сетей.

4.2. Алгоритм *TEEN*

Алгоритм *TEEN* представляет собой реактивный, управляемый событиями алгоритм, который использует рандомизацию для распределения энергетической нагрузки равномерно среди узлов беспроводной сенсорной сети. Алгоритм *TEEN* превосходит по эффективности все известные алгоритмы

маршрутизации, как в гомогенных, так и в гетерогенных стационарных БСС. Алгоритм *TEEN* является *LEACH*-подобным алгоритмом [41] и его функционирование основано на разбиении всего периода времени на одинаковые раунды, по истечении которых происходит выбор нового головного узла в кластере. Алгоритм *TEEN* в отличие от алгоритма *LEACH* имеет несколько порогов. Головной узел может назначать своим узлам «жесткий» (*hard*) и «мягкий» (*soft*) пороги:

- Жесткий порог (*Hard Threshold*): Узел посылает информацию головному узлу только, если количество накопленных данных находится в заданных пределах;
- Мягкий порог (*Soft Threshold*): узел посылает информацию головному узлу только, когда количество накопленных данных изменилось как минимум на величину, равную или большую, чем мягкий порог.

4.3.1. Фаза формирования кластера.

Фаза формирования кластера в алгоритме *TEEN* подобна фазе формирования кластера в алгоритме и выглядит следующим образом.

В фазе формирования кластера каждый сенсорный узел генерирует случайное число от 0 до 1. Каждый сенсорный узел имеет порог $T(n)$, который соответствует предварительно определённом числу головных сенсорных узлов в сети. Если интегрированное случайное число меньше, чем $T(n)$, то сенсорный узел может стать головным в текущем раунде жизни БСС, в противном случае этот узел остаётся только членом кластера. Вычисление $T(n)$ является ключевой задачей при реализации алгоритмов *TEEN* и *LEACH*.

$$T(n) = \begin{cases} \frac{P}{1 - P * (r \bmod \frac{1}{P})} & \text{if } n \in G \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

В (4.1) P – предопределенный процент головных узлов среди всех сенсорных узлов. Оптимальное значение P оценивается в 5% от общего числа сенсорных узлов. Переменная r соответствует текущему интервалу функционирования сенсорной сети, G обозначает число сенсорных узлов, которые не

были выбраны головными за последние $1/p$ интервалов. Кроме того, при выборе головного узла другие сенсорные узлы выбирают к какому кластеру они хотят присоединиться на основе мощности получаемого сигнала (*RSS*– *Received Signal Strength*) от головного узла.

4.3.2. Фаза передачи данных (*Steady-state phase*).

Когда все узлы организованы в кластеры, головной узел создает расписание передачи информации на основе метода *TDMA*, что гарантирует отсутствие коллизий при передаче сообщений. Члены кластера собирают данные непрерывно. Узлы будут передавать данные в текущем периоде кластера только тогда, когда текущее значение:

- Больше, чем жесткий порог;
- отличается от некоторого заданного значения на величину, равную или большую, чем мягкий порог.

4.3. Алгоритм *FT-TEEN*

Предложенный алгоритм представляет собой модифицированный вариант алгоритма *TEEN*, вводит резервный механизм для кластеризации, обнаружение и восстановление ошибок, как видно на рисунке 4.2:

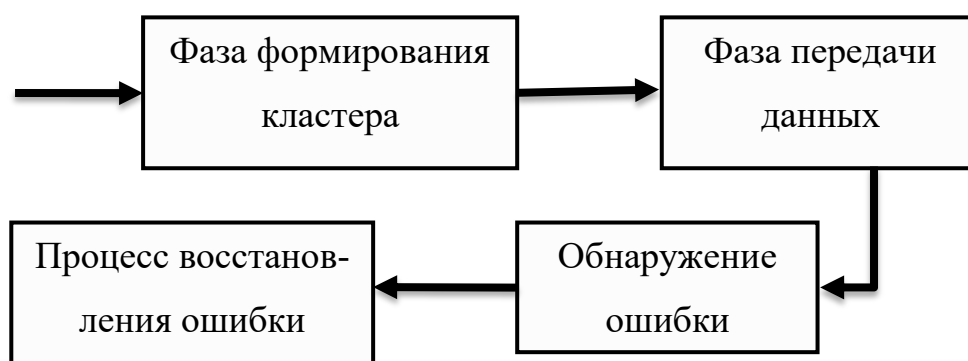


Рис. 4.2 – Блок-схема для предложенного алгоритма

4.3.1. Модель для исследования алгоритма *FT-TEEN*.

При разработке алгоритма *FT-TEEN* была принята следующая модель сети:

- Сенсорная сеть является гомогенной, т. е. все БСУ обладают одинаковыми базовыми параметрами (начальная энергия, радиус действия);
- БСУ осуществляют свою активность без централизованного управления;
- Предложенный механизм может быть применен ко многим иерархическим протоколам маршрутизации, таким как *LEACH* протокол и ему подобные;
- Базовая станция находится в стационарном состоянии и радиоканал симметричен.

Для распространения электромагнитных волн также принимается стандартная модель для беспроводных сенсорных сетей. При этом для оценки потерь учитывается положение сенсорных узлов, находятся ли они в зоне прямой видимости (*LOS*) или нет (*NonLOS*) [88].

Предложенный алгоритм представляет собой модифицированный вариант протокола *TEEN*, вводит резервный механизм для кластеризации и состоит из трех фаз.

4.4.2. Фаза формирования кластера. Выбор головного (*CH*) и резервного головного (*sub-CH*) узлов кластера.

В фазе формирования кластера выбираются головные и резервные головные узлы кластера. Когда все узлы организовались в кластеры, головной узел передает широковещательное сообщение о том, что он стал головным. Кроме того, после получения этих сообщений, каждый БСУ в сети выбирает наиболее подходящий головной узел кластера для присоединения на основе мощности получаемого сигнала *RSS* (*Received Signal Strength*) от головного узла. Далее головной узел *CH* находит среди членов кластера такой БСУ, у

которого остаточная энергия максимальна и определяет его как резервный головной узел *sub-CH* и передает эту информацию членам кластера. Рисунок 4.3 иллюстрирует фазу формирования кластера в разработанном алгоритме.

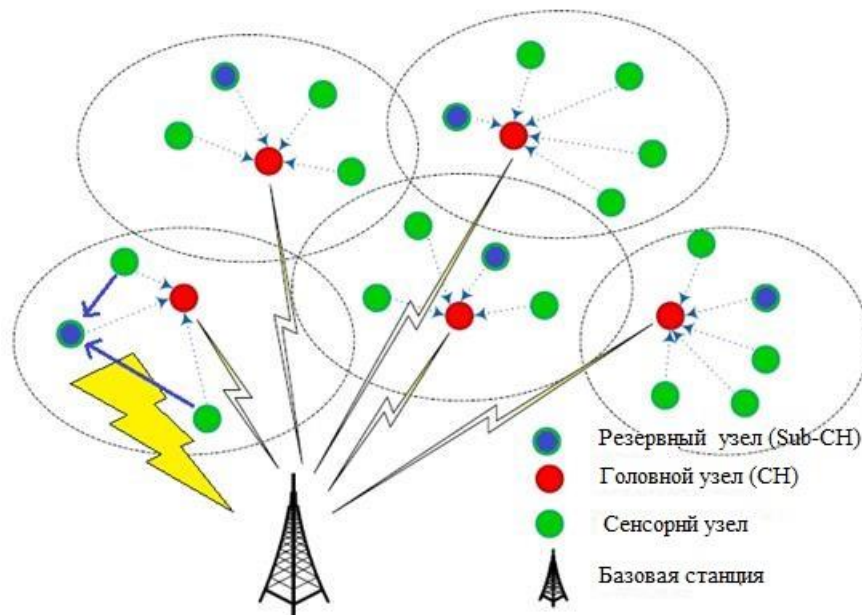


Рис. 4.3 – Кластеризация в разработанном алгоритме

В алгоритме 4.1 приведен алгоритм для фазы формирования кластера, а на рисунке 4.4 – блок-схема для разработанного алгоритма.

– Алгоритм 4.1. Фаза формирования кластера.

- 1: **сделать** { // повторить для R раундов,
- 2: **Если** ($E_{init}(s) > 0$ & s принадлежит G) **То**
- 3: $r \leftarrow$ случайный $(0, 1)$;
- 4: Вычислить $T(s)$; // вид (1)
- 5: **Если** ($r < T(s)$) **То**
- 6: $CH\{s\} = TRUE$; //узел становится головным узлом (истина)
- 7: **иначе**
- 8: $CH\{s\} = FALSE$; // узел не будет CH (ложь)
- 9: **конец если**
- 10: **конец если**
- 11: **Если** ($CH\{s\} = TRUE$) **То**
- 12: **ВС** (ADV) включает в себя $\leftarrow$ передачу рекламных сообщений;
- 13: Присоединить (ID_i , Остаточная энергия); // присоединение

неголового узла

14: Выберите узел с макс. энергией как резервный (sub-CH)

15: Кластер(с); //кластер сформирован с;

16: **конец если**

R: раунд

Einit(s): Начальная энергия

T(s): Пороговое значение

Msg: Проверка сообщения

ADV: Сообщение от CH

BC: Широковещательное сообщение

PCK: Пакет

Idi: Идентификатор узла *ID*



Рис. 4.4 – Блок-схема формирования кластера для разработанного алгоритма

4.3.2. Фаза передачи данных.

В фазе передачи данных *CH*, а при необходимости и *Sub-CH*, получают текущие данные от членов кластера. После получения всех данных головной узел агрегирует и обобщает эти данные, и только после этого направляет их на базовую станцию или шлюз. Точно так же, как при реализации алгоритма *TEEN*, наличие жесткого порога (*HT*) сокращает число сообщений, поскольку передача осуществляется только тогда, когда текущая энергия находится в заданном диапазоне. Кроме того, мягкий порог тоже уменьшает число сообщений путем исключения тех из них, во время формирования которых значение энергии изменилось как минимум на значение порога.

4.3.3. Обнаружение и восстановление ошибок.

Предложенный алгоритм *FT-TEEN* обнаруживает и восстанавливает ошибки согласно следующему механизму (алгоритм 4.2):

Шаг 1: Инициализация головного узла (*CH*) и резервного головного узла (*Sub-CH*) в каждом кластере;

Шаг 2: Если ответ не приходит от головного узла (*CH*) в пределах заданного временного интервала;

ТОГДА

Шаг3: Головной узел (*CH*) отмечается как неисправный и резервный головной узел (*Sub-CH*) становится головным узлом в кластере;

Шаг4: ЕСЛИ ответ не приходит от резервного головного узла (*Sub-CH*) в пределах заданного временного интервала;

ТОГДА

Шаг5: Резервный головной узел (*Sub-CH*) отмечается как неисправный и находится новый резервный головной узел из членов кластера.

– Алгоритм 4.2. Обнаружение и восстановление ошибок

Сделать { // Для каждого кластера (*c*)

1: Проверка состояния *CH* и *Sub CH*

- 2: **Если** ответ не приходит от головного узла (CH)
- 3: **отметить** CH как неисправный, $CH \leftarrow sub\ CH$ // заменить на резервный CH
- 4: **Если** нет ответа от резервного узла ($sub-CH$)
- 5: **отметить** $sub-CH$ как неисправный, CH найти новый ($sub-CH$)
- }

Для передачи данных в *FT-TEEN* используется следующий алгоритм (Алгоритм 4.3).

– Алгоритм 4.3 Передача данных

- 1: **Если** (узел i головной или резервный) **То**
- 2: Получить ($IDi, DataPCK$)// получить данные от членов кластера;
- 3: Обработать($IDi, DataPCK$)// обработать данные;
- 4: Передать на БС ($IDi, DataPCK$)// передать данные на БС;
- 5: иначе
- 6: **Если** (данные узла находятся в диапазоне интереса) **То**
- 7: Передать на CH & $sub-CH$ ($IDi, DataPCK$); // передача данных на CH и $sub-CH$
- 8: **иначе**
- 9: Состояние сна (i)= $TRUE$; // узел i в состоянии сна
- 10: **конец если**

11: **конец если**

12: // раунд завершен.

На рисунке 4.5 представлена блок-схема передачи данных для разработанного алгоритма.

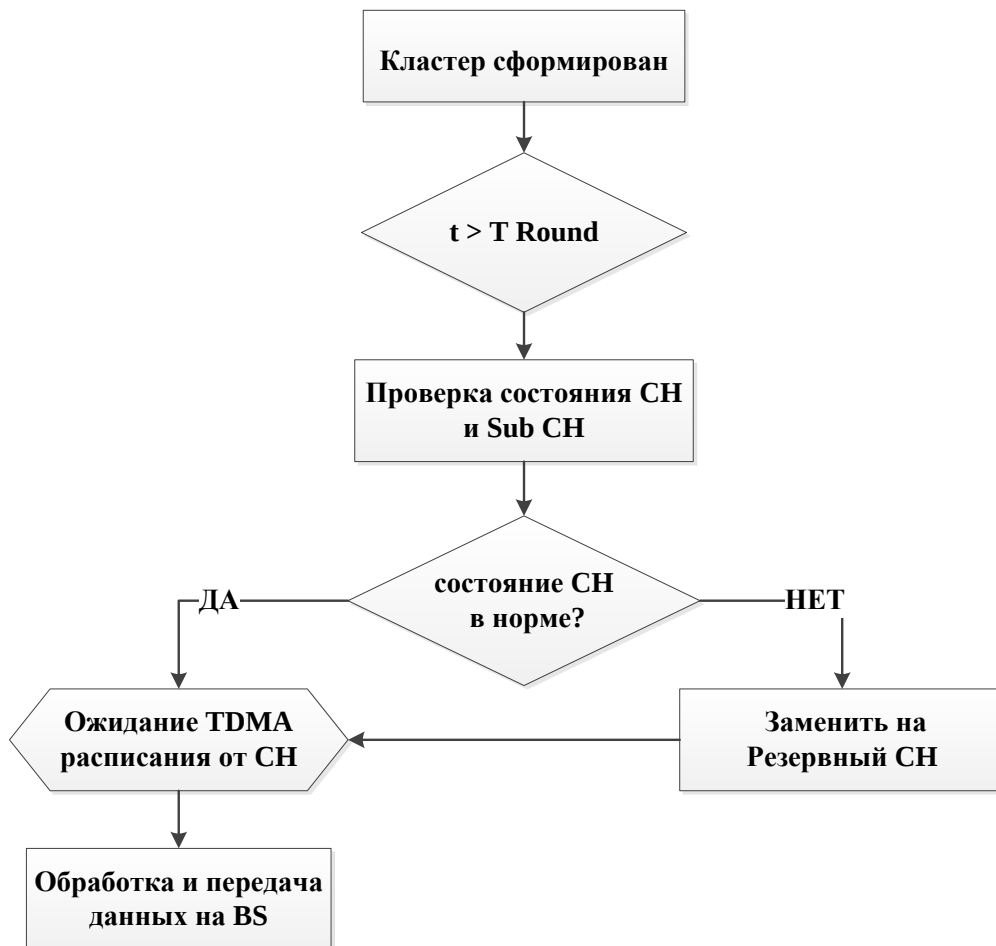


Рис. 4.5 – Блок-схема передачи данных для разработанного алгоритма

4.4. Результаты моделирования

Для сравнения характеристик разработанного алгоритма *FT-TEEN* с базовым алгоритмом *TEEN* было проведено моделирование в среде *MATLAB*. Моделирование выполнялось для тех же исходных данных, которые традиционно приняты для алгоритмов *TEEN* и *LEACH* [76, 93]. Сравнение алгоритмов проводилось по числу успешно переданных пакетов между членами кластера и головным узлом, а также между головным узлом и базовой стан-

цией. При моделировании рассматривалось пуассоновское поле со 100 сенсорными узлами площадью 100м*100м. Предполагалось, что базовая станция расположена за пределами поля с координатами $x=130$, $y=50$. Размер пакета данных был выбран величиной в 512, а число кадров в каждом раунде 10. Другие параметры, используемые при моделировании, приведены в таблице 4.4.

Таблица 4.4 – Параметры и их значения, используемые при моделировании.

Параметр	Обозначение	Значение
Первичная энергия на узел	E	0.5 j
Tx/Rx	E_{elec}	50 nj/bit
Постоянное усиление	ε_{fs}	10 pj/bit/m ²
Мультисетевая постоянная	ε_{mp}	0.0013 pj/bit/m ²
Потери на участках (экспоненциальные)	$\frac{\varepsilon_{fs}}{\varepsilon_{mp}}$	87
Мягкий порог (<i>Soft Threshold</i>)	ST	2
Жесткий порог (<i>HradThreshold</i>)	HD	100

На рисунке 4.6 и рисунке 4.7 показано число успешно переданных пакетов от членов кластера на головной узел и от головного узла на базовую станцию соответственно в течение одного раунда при вероятности ошибки в 10%.

На рисунке 4.8 и рисунке 4.9 показано число успешно переданных пакетов от членов кластера на головной узел и от головного узла на базовую станцию соответственно в течение одного раунда при вероятности ошибки в 20%.

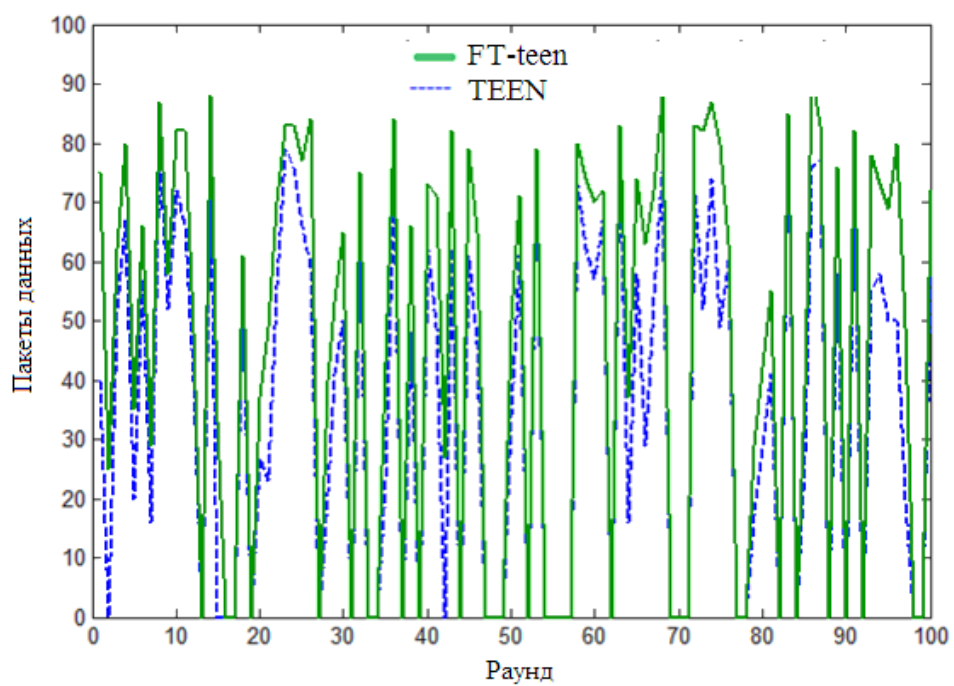


Рис. 4.6 – Число успешно переданных пакетов от членов кластера на *CH*.

(Вероятность ошибки 10%)

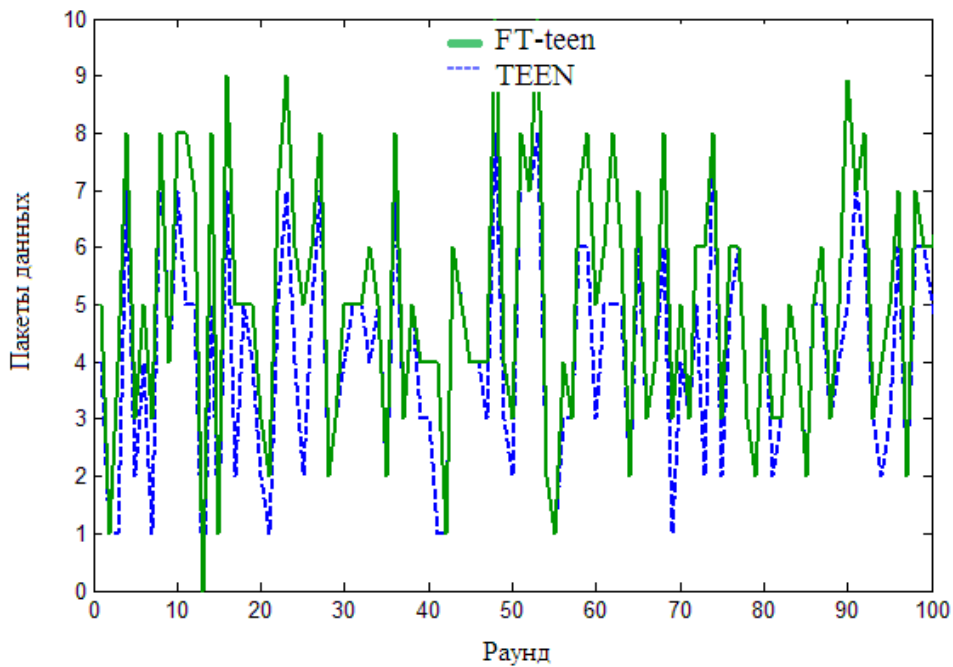


Рис. 4.7 – Число успешно переданных пакетов от *CH* на базовую станцию каждый раунд.

(Вероятность ошибки 10%)

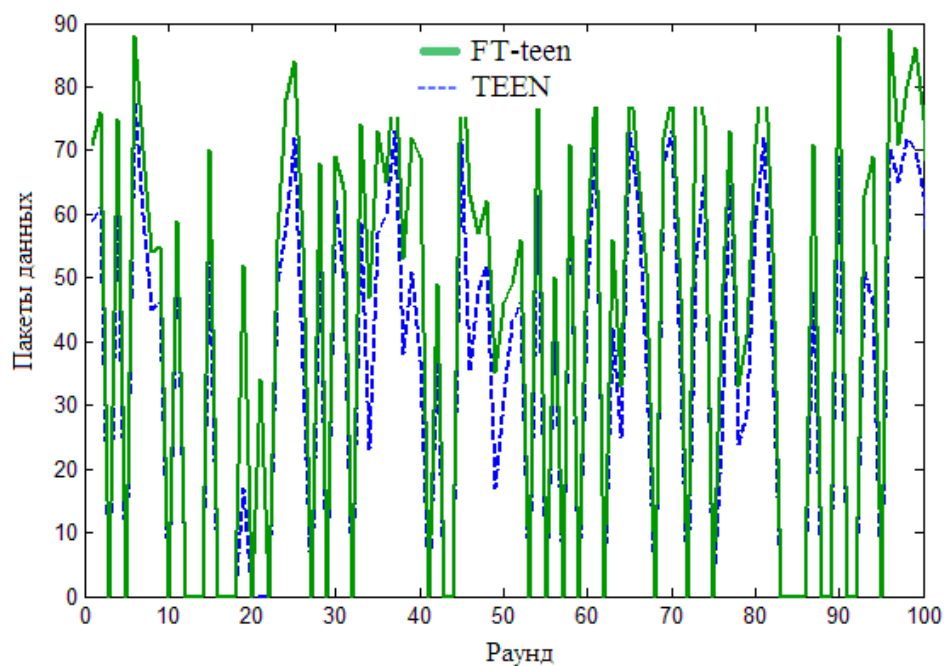


Рис. 4.8 – Число успешно переданных пакетов от членов кластера на *CH*

Вероятность ошибки 20%

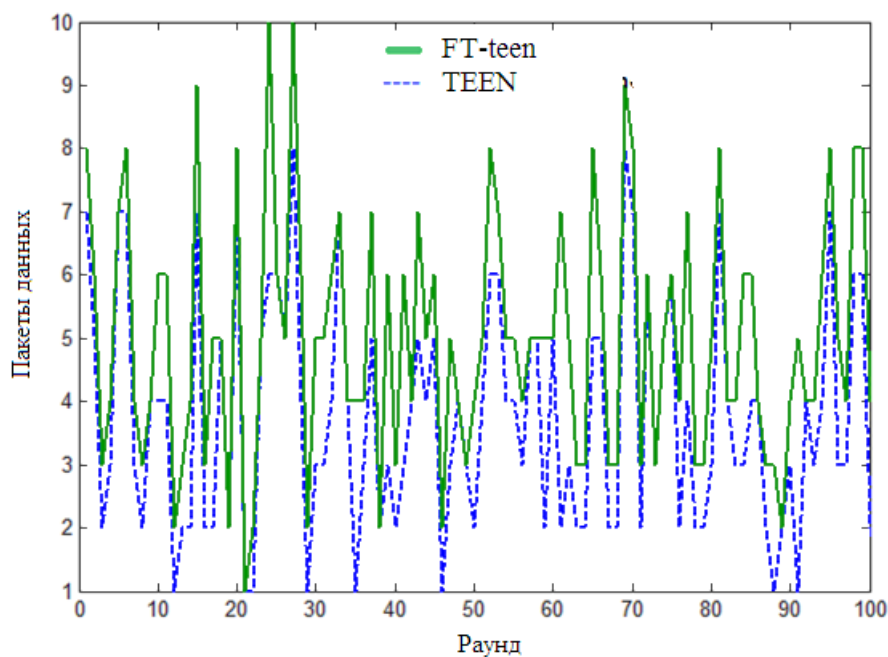


Рис. 4.9 – Число успешно переданных пакетов от *CH* на базовую станцию.

Вероятность ошибки 20%

Анализ результатов моделирования доказывает, что разработанный алгоритм обеспечивает увеличение числа успешно переданных пакетов как от

членов кластера к *CH*, так и от *CH* к базовой станции по сравнению с алгоритмом *TEEN*. При этом, чем больше вероятность ошибки, тем больше увеличивается число успешно переданных пакетов для разработанного алгоритма по сравнению с алгоритмом *TEEN*.

На рисунке 4.10 и рисунке 4.11 показано число успешно переданных пакетов от членов кластера на головной узел и от головного узла на базовую станцию соответственно в течение жизненного цикла сенсорной сети при различных вероятностях ошибки.

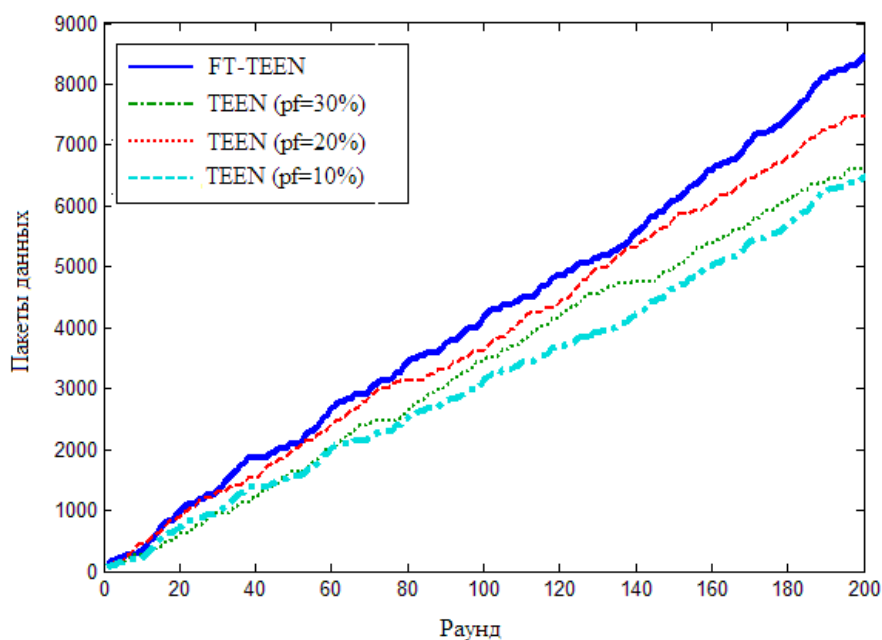


Рис. 4.10 – Число успешно переданных пакетов от членов кластера на *CH*

Как видим, и в том, и в другом случае число успешно переданных пакетов для разработанного алгоритма существенно больше, чем для базового алгоритма *TEEN*. Кроме того, разработанный алгоритм *FT-TEEN* обеспечивает толерантность к значению вероятности ошибки, в то время как для базового алгоритма *TEEN* число успешно переданных пакетов существенно снижается с ростом вероятности ошибки в рассмотренных пределах.

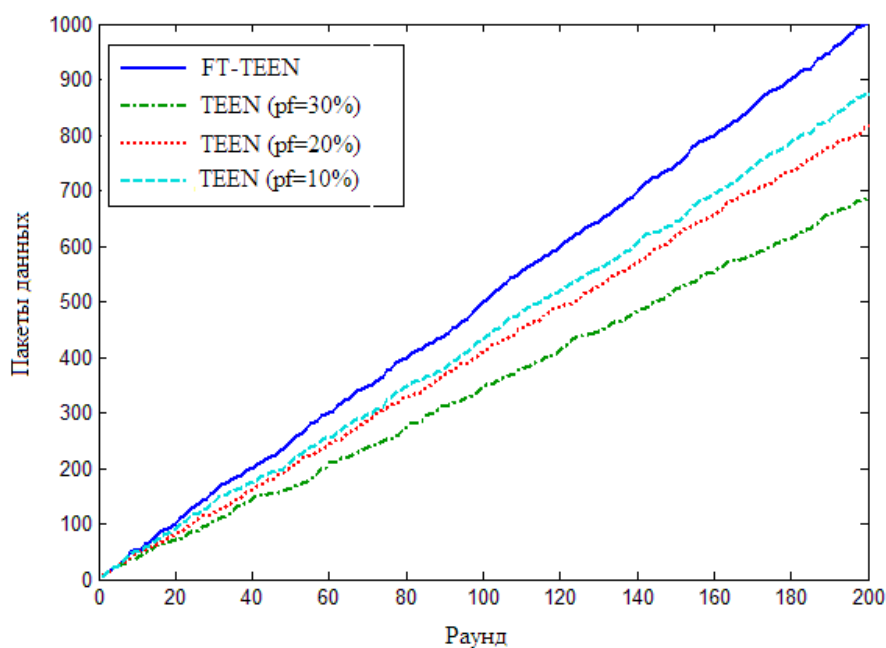


Рис. 4.11 – Число успешно переданных пакетов от *CH* на базовую станцию

Результаты моделирования для всего жизненного цикла сети сведены в таблицу 4.2.

Таблица 4.2 – Результаты моделирования

Вероятность ошибки P_f , %	$TEEN$		$FT-TEEN$		Доля улучшения для переданных пакетов %	
	Без обнаружения сбоев и восстановления, успешно переданные пакеты		С обнаружением сбоев и восстановлением, успешно переданные пакеты			
	в CHs	в BS	в CHs	в BS	в CHs	в BS
10	7500	876	8409	1005	12	15
20	6655	822	8409	1005	26	22
30	6405	692	8409	1005	31	45

Выводы

1. В главе предложен новый отказоустойчивый алгоритм, основанный на *LEACH*-подобном алгоритме *TEEN*, являющимся наилучшим из известных алгоритмов кластеризации для БСС со стационарными узлами.

2. Разработанный алгоритм обеспечивает увеличение числа успешно переданных пакетов как от членов кластера к *СН*, так и от *СН* к базовой станции по сравнению с алгоритмом *TEEN*.

3. Разработанный алгоритм *FT-TEEN* обеспечивает толерантность к значению вероятности ошибки, в то время как для базового алгоритма *TEEN* число успешно переданных пакетов существенно снижается с ростом вероятности ошибки в рассмотренных пределах.

5. ПОКРЫТИЕ, СВЯЗНОСТЬ И ПЛОТНОСТЬ В ДВУМЕРНЫХ И ТРЕХМЕРНЫХ БЕСПРОВОДНЫХ СЕНСОРНЫХ СЕТЯХ

5.1. Обзор исследований в предметной области

Несмотря на то, что большинство существующих работ в области беспроводных сенсорных сетей (БСС) в настоящее время посвящены двумерному пространству ($2D$), на самом деле такие сети работают в трехмерном пространстве ($3D$), особенно с учетом появления новых приложений, таких как летающие сенсорные сети. Переход от двумерного к трехмерному пространству порождает множество новых проблем в связи с иной топологией сети. Требуются новые подходы к оценке таких характеристик БСС как покрытие, связность и плотность. Исходя из сказанного, в настоящей главе мы фокусируем внимание на характеристиках плотности и связности для БСС с целью определения такой стратегии размещения сенсорных узлов, чтобы возможно было обеспечить, по крайней мере, 90% покрытие для двумерных ($2D$) и трехмерных ($3D$) БСС. При этом оцениваются также длительность жизненного цикла сети, период стабильности и пропускная способность сети на основе отношения между радиусом покрытия (R_s) и радиусом дальности связи (R_c). Результаты могут быть использованы при планировании как БСС, размещенных на плоскости, в трехмерном пространстве, так и летающих сенсорных сетей.

В последние годы исследования в области БСС постепенно переходят от изучения характеристик на плоскости к моделям в трехмерном ($3D$) пространстве [15, 24]. В [7] предлагается новая область применения технологий БСС — летающие сенсорные сети (ЛСС). Примерами приложений БСС в трехмерном пространстве помимо ЛСС может быть мониторинг многоэтажных зданий, складских помещений, подводный мониторинг и т.п. [62].

Переход от моделей двумерного пространства к трехмерному далеко не прост, так как решение многих проблем в $3D$ значительно сложнее, чем в $2D$.

Топология сети становится существенно сложнее, что непосредственно сказывается на планировании 3D БСС.

При планировании БСС существует множество взаимоувязанных показателей, оказывающих решающее воздействие на последующее функционирование сети. В первую очередь к ним относятся: стратегия размещения сенсорных узлов в трехмерном пространстве (детерминированная или случайная), плотность размещения сенсорных узлов, зона покрытия, качество связи, отношение между радиусом покрытия и дальностью связи [73]. Увеличение зоны покрытия является фундаментальным требованием для большинства приложений сенсорных сетей, например, мониторинга [87], управления, слежения за целью [59].

На рисунке 5.1 представлено изображение сенсорных сетей в двумерном и трехмерном пространстве.

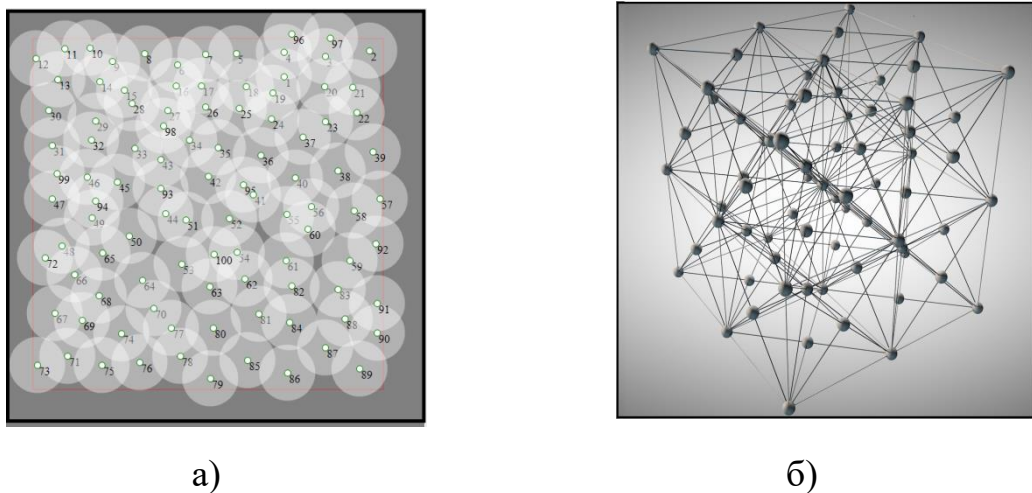


Рис. 5.1 – БСС в двумерном и трехмерном пространствах:

а - БСС в 2D; б - БСС в 3D

Заметим, что трехмерная сеть может быть представлена с помощью множества двумерных (см. Рисунок. 5.2). Каждое трехмерное пространство может быть разделено на n двумерных плоскостей, где $n \rightarrow \infty$.

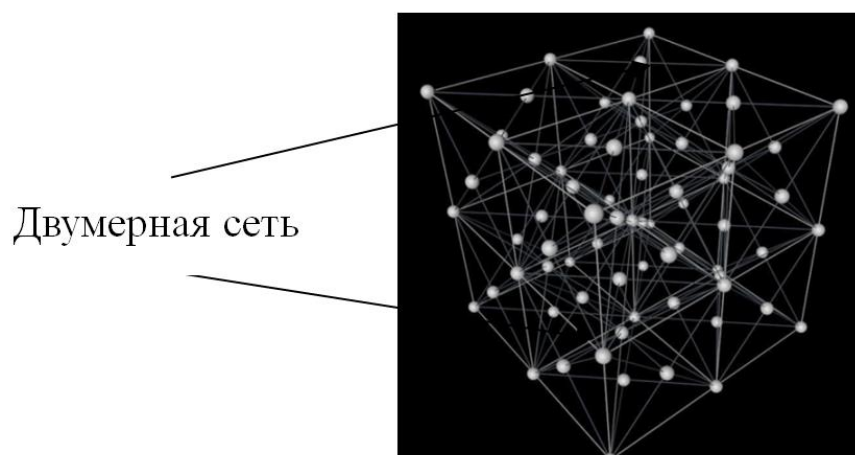


Рис. 5.2 – Трехмерная сеть включает двумерные сети

Цель главы – разработка такой методики размещения БСС, при которой можно обеспечить 90% и более покрытие пространства как для двумерных, так и трехмерных БСС. Для этого необходимо ответить на следующие вопросы:

- каким должны быть минимальный радиус покрытия (R_s) сенсорного узла и плотность, чтобы получить более 90% покрытие для двумерных и трехмерных БСС?
- какова должна быть плотность активных сенсорных узлов в текущий момент времени для гарантии 90%-го покрытия и полносвязной сети?
- каким должно быть отношение между радиусом покрытия и дальностью связи сенсорного узла R_c / R_s для обеспечения таких заданных параметров БСС, как длительность жизненного цикла, период стабильности и пропускная способность сети?

5.2. Стратегии расположения БСУ в двумерном и трехмерном пространствах.

Сенсорные узлы могут быть размещены в пространстве двумя способами: детерминированно и случайно [72]. В детерминированных методах БСУ располагаются только в определенных точках пространства, вычислен-

ных пользователем. Этот метод гарантирует полное покрытие пространства в течение определенного времени в зависимости от плотности сенсорных узлов, методов их кластеризации и т.п. Однако существуют области применения БСС, в которых детерминированное размещение узлов нецелесообразно или даже невозможно, например, из-за высокой плотности узлов или размещения их в недружелюбной или опасной окружающей среде. Тогда применение детерминированных методов невозможно и случайное расположение может быть единственно возможной стратегией.

При случайном размещении сенсорные узлы располагаются случайным образом с использованием летательных аппаратов, иных мобильных средств, замешиваются в бетон и т.д. При этом необходимо, с одной стороны, максимизировать покрытие в течение как можно большего времени и минимизировать энергопотребление за счет уменьшения числа активных узлов, выполняющих одни и те же функции в пределах своего радиуса действия. Последнее возможно реализовать только, когда известна необходимая плотность размещения сенсорных узлов.

В приложениях слежения за целью минимальное число узлов, требуемых для полного покрытия области, определяется как некое критическое значение плотности сенсорных узлов, позволяющее обнаружить любой объект, пересекающий пространство БСС. Поэтому для эффективного обнаружения любого объекта, пересекающего пространство, плотность БСУ должна быть выше критического значения (определенный порог плотности) [72].

5.3. Соотношение между радиусом покрытия сенсорного узла, долей покрытия и плотностью в 2D и 3D БСС.

В разделе исследуется соотношение между радиусом покрытия R_s , долей покрытия C и плотностью сенсорных узлов ρ в 2D и 3D БСС в случае, когда сенсорные узлы размещены случайным образом.

5.3.1. Двумерная БСС

Предположим, что n сенсорных узлов случайным образом распределены в $2D$ плоскости и зона покрытия сенсорного узла представляет собой диск с радиусом R_s , площадь которого:

$$S = \pi R_s^2 \quad (5.1)$$

Тогда плотность сенсорных узлов составляет $\rho = n/S$, Доля покрытия, определяемая как отношение зоны покрытия A или объема V к общей площади или объема в момент времени $t > 0$ после размещения сенсорных узлов, вычисляется по формуле [60]:

$$C = 1 - e^{-\rho S} \quad (5.2)$$

Выражение (5.2) может быть использовано при планировании БСС для определения требуемой плотности БСУ.

Для достижения желаемой целевой зоны покрытия C ($0,9 \leq c \leq 1$) при $c \geq 0,99 \dots 0,90 \Leftrightarrow \rho S \geq 4,6 \dots 2,3$ соответственно, необходимые плотности сенсорных узлов могут быть определены следующим образом:

$$c=0,98 \Leftrightarrow \rho S \geq 391; c=0,97 \Leftrightarrow \rho S \geq 3,50;$$

$$c=0,96 \Leftrightarrow \rho S \geq 3,21; c=0,95 \Leftrightarrow \rho S \geq 2,99;$$

$$c=0,94 \Leftrightarrow \rho S \geq 2,81; c=0,93 \Leftrightarrow \rho S \geq 2,66;$$

$$c=0,92 \Leftrightarrow \rho S \geq 2,53; c=0,91 \Leftrightarrow \rho S \geq 2,41;$$

$$c=0,90 \Leftrightarrow \rho S \geq 2,30.$$

В общем случае требуемая плотность БСС:

$$\rho = \frac{-\log(1-c)}{\pi r^2} \Leftrightarrow R_s = \sqrt{\frac{-\log(1-c)}{\pi \rho}} = \sqrt{\frac{-\log(1-c)S}{\pi n}} \quad (5.3)$$

На рисунке 5.3 показаны соотношения между радиусом покрытия, долей покрытия и плотностью размещения сенсорных узлов в $2D$ БСС для получения заданной доли покрытия.

Таблица 5.1 – Соотношения между радиусом покрытия R_S , долей покрытия C и плотностью ρ в 2D БСС

$C\%$ $\rho * e^{-3}$	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
$\rho_{(R_S=50m)}$	0,8	0,7	0,5	0,4	0,3	0,2	0,2	0,2	0,2	0,1
$\rho_{(R_S=45m)}$	0,8	0,8	0,5	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=40m)}$	0,9	0,8	0,5	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=35m)}$	0,9	0,8	0,6	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=30m)}$	0,9	0,8	0,6	0,4	0,4	0,3	0,3	0,2	0,2	0,2
$\rho_{(R_S=25m)}$	1	0,8	0,6	0,5	0,4	0,3	0,3	0,2	0,2	0,2
$\rho_{(R_S=20m)}$	1	1	0,7	0,5	0,4	0,3	0,3	0,3	0,2	0,2
$\rho_{(R_S=15m)}$	1	1,1	0,7	0,6	0,4	0,4	0,3	0,3	0,2	0,2
$\rho_{(R_S=10m)}$	1,1	1,2	0,8	0,6	0,5	0,4	0,4	0,3	0,3	0,2
$\rho_{(R_S=5m)}$	1,9	1,3	1	0,7	0,6	0,5	0,4	0,4	0,3	0,3

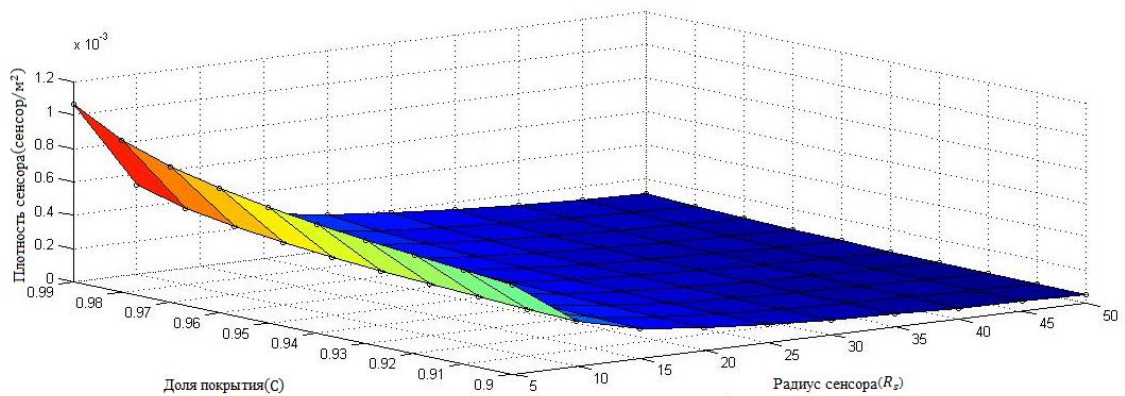


Рис. 5.3 – Соотношения между радиусом покрытия R_S , долей покрытия C и плотностью ρ в 2D БСС

5.3.2. Трехмерная БСС

Сенсорные узлы $S = \{S_1, S_2, \dots, S_n\}$, случайным образом распределены в трехмерной области V . Каждый сенсорный узел S_i , $i=1 \dots n$, имеет координаты

наты (x_i, y_i, z_i) в объеме V и радиус покрытия. Таким образом, зона покрытия для S_i может быть представлена сферой с центром в (x_i, y_i, z_i) и радиусом:

$$V_s = \frac{4}{3} \pi R_s^3 \quad (5.4)$$

Число БСУ, расположенных в объеме V , составит

$$n = V * \rho \quad (5.5)$$

где $\rho \cong \frac{n}{V_s}$

Доля покрытия C определяется аналогично (5.2):

$$c = 1 - e^{-\rho V_s} \quad (5.6)$$

Для достижения желаемой целевой зоны покрытия C ($0,9 \leq c \leq 1$) при $c \geq 0,99 \dots 0,90 \Leftrightarrow \rho S \geq 4,6 \dots 2,3$ соответственно, необходимые плотности сенсорных узлов могут быть определены следующим образом:

$$c=0,98 \Leftrightarrow \rho S \geq 391; c=0,97 \Leftrightarrow \rho S \geq 3,50;$$

$$c=0,96 \Leftrightarrow \rho S \geq 3,21; c=0,95 \Leftrightarrow \rho S \geq 2,99;$$

$$c=0,94 \Leftrightarrow \rho S \geq 2,81; c=0,93 \Leftrightarrow \rho S \geq 2,66;$$

$$c=0,92 \Leftrightarrow \rho S \geq 2,53; c=0,91 \Leftrightarrow \rho S \geq 2,41;$$

$$c=0,90 \Leftrightarrow \rho S \geq 2,30.$$

Плотность узлов может быть определена аналогично приведенному выше случаю. В общем случае необходимая плотность БСС:

$$\rho = \frac{-\log(1-c)}{\frac{4}{3}\pi R_s^3} \Leftrightarrow R_s = \left[\frac{-\log(1-c)}{\frac{4}{3}\pi \rho} \right]^{1/3} = \left[\frac{-\log(1-c)V_s}{\frac{4}{3}\pi n} \right]^{1/3}, \quad (5.7)$$

где C – доля покрытия; ρ – плотность; R_s – радиус зоны покрытия сенсорным узлом.

На рисунке 5.4 приведены соотношения между радиусом покрытия R_s , долей покрытия и плотностью размещения сенсорных узлов в трехмерной БСС для получения заданной доли покрытия. На рисунке 5.5 представлены сравнительные характеристики для 2D и 3D БСС.

Таблица 5.2 – Соотношения между радиусом покрытия, долей покрытия и плотностью в 3D БСС.

$\rho * e^{-3}$ \ C %	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
$\rho_{(R_S=50m)}$	1,3	0,7	0,5	0,4	0,3	0,2	0,2	0,2	0,2	0,1
$\rho_{(R_S=45m)}$	1,3	0,8	0,5	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=40m)}$	1,6	0,8	0,5	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=35m)}$	1,7	0,8	0,6	0,4	0,3	0,3	0,2	0,2	0,2	0,2
$\rho_{(R_S=30m)}$	1,8	0,8	0,6	0,4	0,4	0,3	0,3	0,2	0,2	0,2
$\rho_{(R_S=25m)}$	1,9	0,9	0,6	0,5	0,4	0,3	0,3	0,2	0,2	0,2
$\rho_{(R_S=20m)}$	2	1	0,7	0,5	0,4	0,3	0,3	0,3	0,2	0,2
$\rho_{(R_S=15m)}$	2,2	1,1	0,7	0,6	0,4	0,4	0,3	0,3	0,2	0,2
$\rho_{(R_S=10m)}$	2,5	1,2	0,8	0,6	0,5	0,4	0,4	0,3	0,3	0,2
$\rho_{(R_S=5m)}$	2,9	1,3	1	0,7	0,6	0,5	0,4	0,4	0,3	0,3

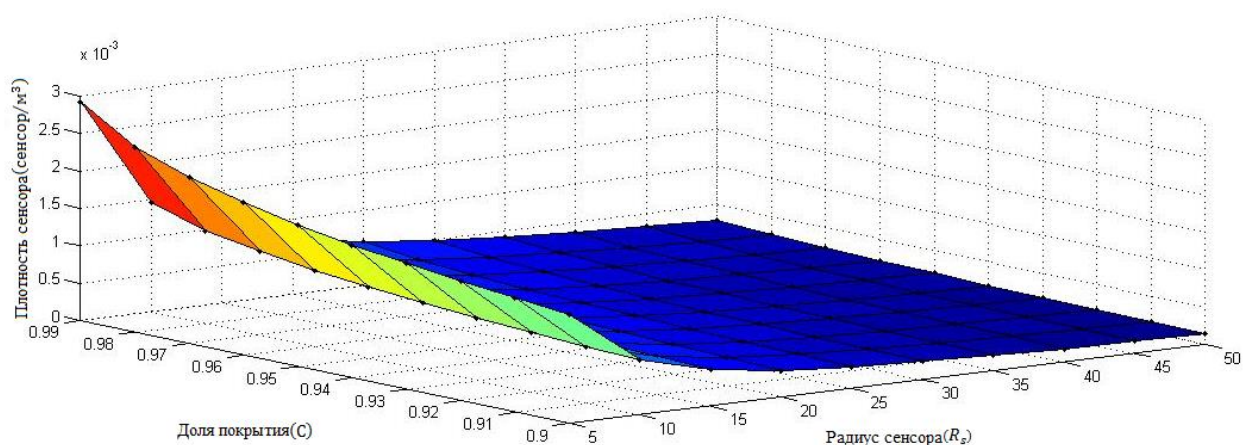


Рис. 5.4 – Соотношения между радиусом покрытия , долей покрытия и плотностью в 3D БСС

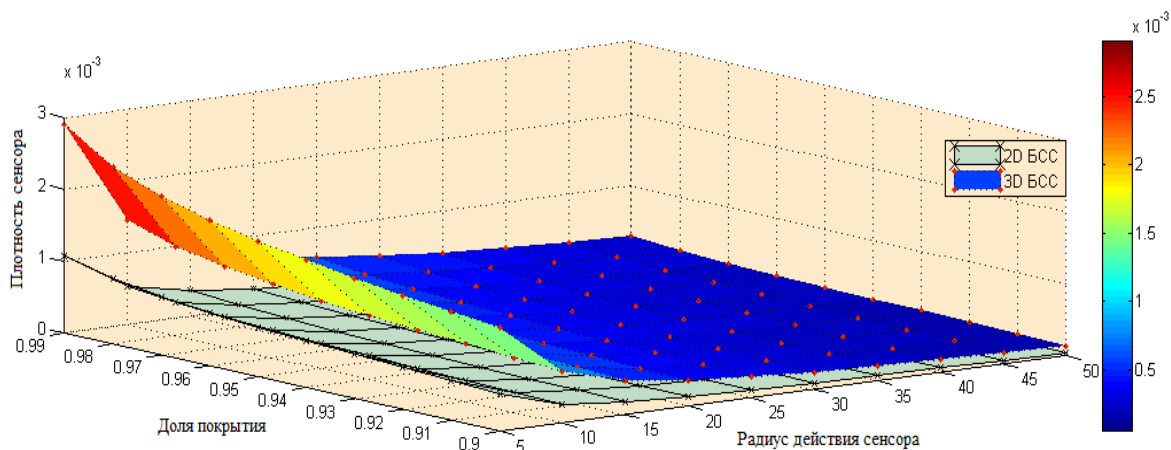


Рис. 5.5 – Сравнительные характеристики для 2D и 3D БСС

Как видно из рисунков 5.3, 5.4, 5.5 размерность сенсорного поля влияет на необходимую плотность БСС для обеспечения заданной доли покрытия. Причем, при прочих равных условиях, для трехмерных БСС необходимая плотность сенсорных узлов больше, чем для двумерных. На рисунке 5.6 показаны значения доли покрытия как функции плотности ρ для 2D и 3D БСС при $R_s = 30\text{м}$.

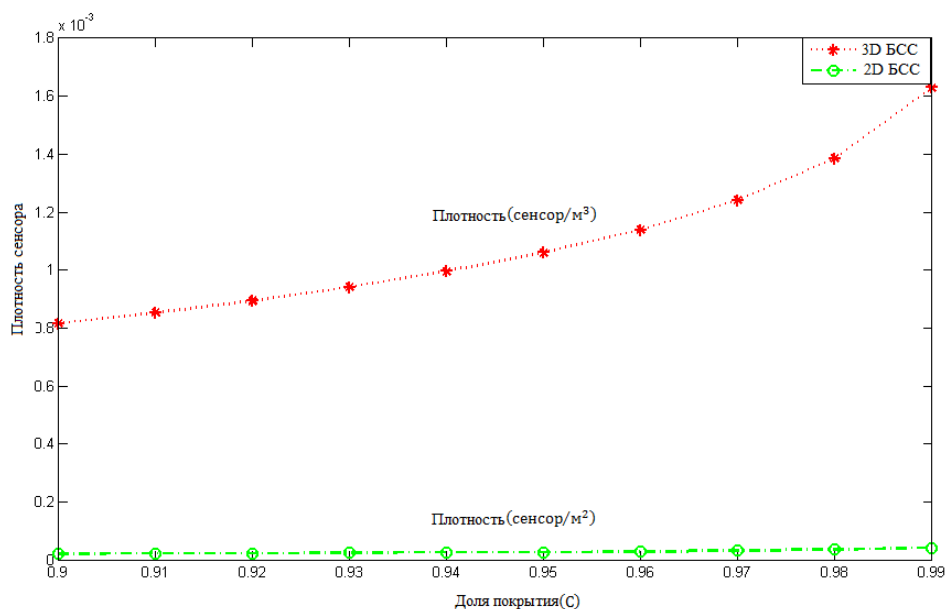


Рис. 5.6 – Доля покрытия C как функция плотности ρ в 2D и 3D БСС при $R_s=30\text{м}$

На рисунке 5.7 приведено сравнение числа близлежащих узлов, т.е. узлов, находящихся на расстоянии меньше $2R_s$ для $2D$ и $3D$ БСС. Поскольку необходимая плотность размещения сенсорных узлов для обеспечения заданной доли покрытия для $3D$ существенно выше, чем для $2D$ БСС, то и число близлежащих узлов для $3D$ БСС также намного больше, чем для $2D$.

На рисунке 5.7 приведено сравнение числа близлежащих узлов, т.е. узлов, находящихся на расстоянии меньше $2R_s$, для $2D$ и $3D$ БСС. Поскольку необходимая плотность размещения сенсорных узлов для обеспечения заданной доли покрытия для $3D$ существенно выше, чем для $2D$ БСС, то и число близлежащих узлов для $3D$ БСС также намного больше, чем для $2D$.

Для того, чтобы доля покрытия достигалась с вероятностью $C \geq 0.99$, плотность должна быть $\rho \geq \frac{4.6}{V_s}$. Область покрытия сенсорного узла будет пересекаться с областями покрытия всех узлов, находящихся на расстоянии меньше $2R_s$, и тогда в среднем число таких узлов будет $\rho * (\frac{4}{3}\pi \cdot (2R_s)^3) = 2^3 * \rho V_s \geq 4.6 * 2^3 \approx 37$ в $3D$. Соответствующее значение в $2D$ будет составлять ≈ 18 . Точно так же для других величин.

$$2^3 * \rho V_s \geq 3.912 * 2^3 \approx 32 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 16;$$

$$2^3 * \rho V_s \geq 3.500 * 2^3 \approx 28 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 14;$$

$$2^3 * \rho V_s \geq 3.210 * 2^3 \approx 26 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 13;$$

$$2^3 * \rho V_s \geq 2.995 * 2^3 \approx 24 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 12;$$

$$2^3 * \rho V_s \geq 2.813 * 2^3 \approx 23 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 11;$$

$$2^3 * \rho V_s \geq 2.660 * 2^3 \approx 22 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 11;$$

$$2^3 * \rho V_s \geq 2.526 * 2^3 \approx 21 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 10;$$

$$2^3 * \rho V_s \geq 2.408 * 2^3 \approx 20 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 10;$$

$$2^3 * \rho V_s \geq 2.300 * 2^3 \approx 19 \text{ в } 3D, \text{ в } 2D \text{ is } \approx 9.$$

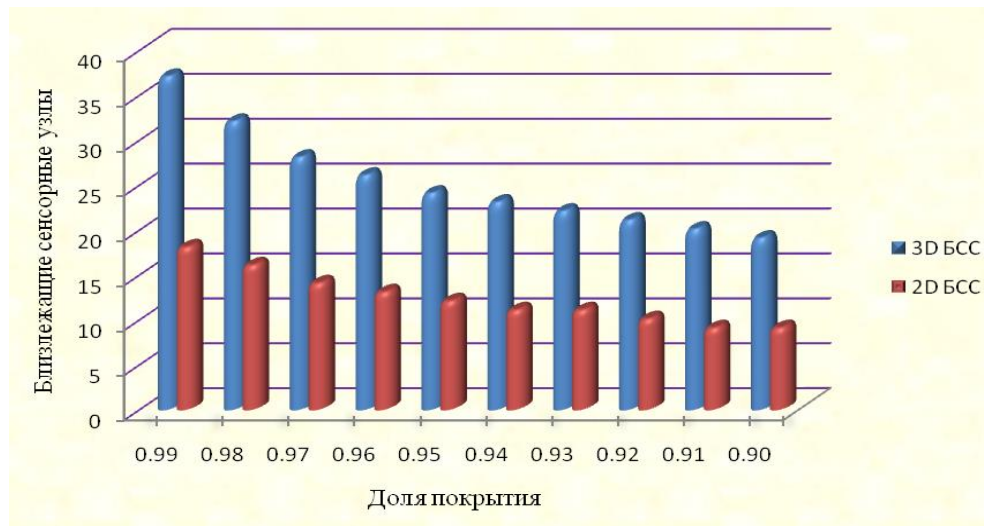


Рис. 5.7 – Соотношения между долей покрытия C и числом близлежащих сенсорных узлов, находящихся на расстоянии меньше $2R_s$, при случайном размещении узлов для 2D и 3D БСС

5.4. Связность сети

Связность определяет возможность взаимосвязи между любыми сенсорными узлами в сети.

Множество близлежащих к сенсорному узлу сенсоров представим как $N(s_i)$.

Тогда $N(s_i)$ определяется из следующего уравнения:

$$N(s_i) = \{s_j : d(s_i, s_j) < r_i\} \quad (5.8)$$

На рисунке 5.8 связность иллюстрируется с помощью зелёных линий между сенсорными узлами.

Для соединений в БСС важно гарантировать для каждого из них, что будет найден маршрут для передачи данных к базовой станции. Два узла являются связанными, если они могут передать и получить данные друг от друга непосредственно или с помощью транзитных связей через другие сенсорные узлы. Все БСУ изначально имеют идентичную дальность связи, область связи для каждого узла может быть представлена в виде диска для двумерных БСС или сферы для трехмерных.

Критический радиус для связности в одноранговой сети был рассмотрен в [78], а для многошаговых сетей в [57].

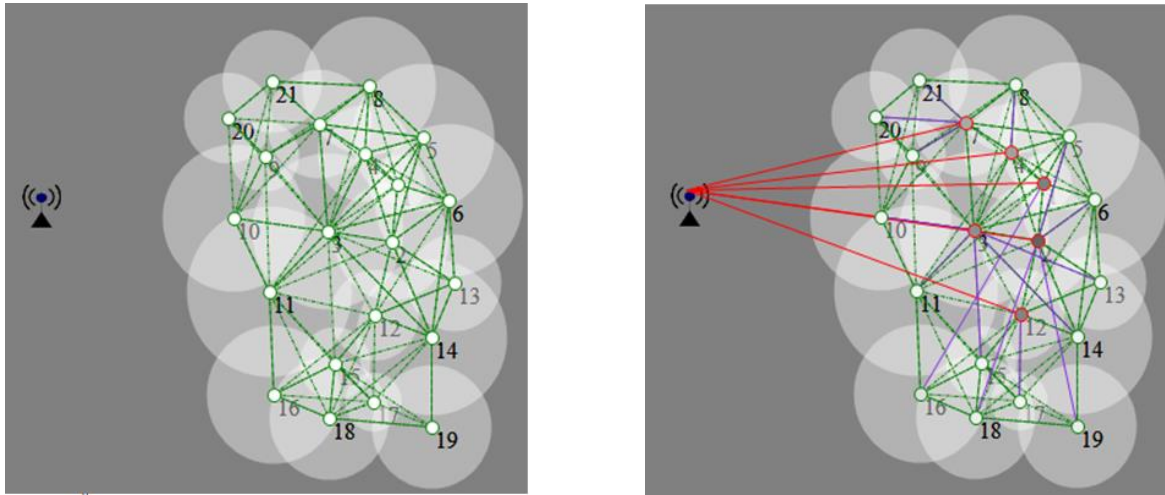


Рис. 5.8 – Связность БСУ

Для n узлов, размещенных в случайном порядке в единичном бее $[0, 1]^d$, критический радиус связности для обеспечения полносвязной сети имеет вид $O\left(\left(\frac{\log n}{n}\right)^{1/d}\right)$ [37].

Если $n = 1000$, то критический радиус связности в $2D$ БСС примерно равен 0,07, а в трехмерной БСС равен 0,2. Это означает, что при равномерном случайном размещении БСС в $3D$ пространстве для полносвязной сети требуется более высокая плотность БСУ [69]. При этом необходимый для $3D$ БСС радиус связи намного больше чем для $2D$.

5.5. Соотношения между радиусом покрытия и дальностью связи.

Соотношение между радиусом покрытия и радиусом связи R_c / R_s является очень важным фактором обеспечения непрерывного функционирования БСС до гибели последнего БСС. Для того чтобы изучить влияние R_c / R_s на производительность и срок службы сети рассмотрим

возможные соотношения между радиусом покрытия и радиусом связи для 2D БСС:

- радиус связи, по крайней мере, в два раза больше радиуса покрытия
 $R_c \geq 2R_s$;
- $R_c = R_s = 30\text{м}$;
- $R_c < R_s$.

Для анализа и сравнения указанных вариантов проводилось моделирование с использованием C# и VB.NET. Оценивалась производительность сети (период стабильности, длительность жизненного цикла и принятых пакетов на базовой станции, расположенной вне территории сети) на основе соотношения R_c / R_s .

Модель сети представляет собой 200 БСУ, равномерно размещенных на плоскости 400м * 400м. Предполагается, что БС фиксирована и расположена вне сети (150м, 150м), сенсорный узел функционирует с фиксированной скоростью передачи и всегда имеет данные для передачи на БС.

Таблица 5.3 – Влияние отношение R_c / R_s на производительность сети

Отношение R_c / R_s	Первый узел погиб в раунде	40% процентов узлов гибнут за n раундов	Число принятых пакетов
$R_c \geq 2R_s$	140	1200	1525
$R_c = R_s$	70	460	790
$R_c < R_s$	67	370	770

Как видно из таблицы, 40% узлов во втором и третьем случаях гибнут за меньшее время, чем в первом. Однако сети продолжают работать, поскольку есть покрытие и связность между узлами. Это означает, что БСС в сетях еще имеют необходимую энергию. Проблема заключается в возможности соединения между ними, и решить ее можно с помощью дополнительной БС.

Рисунок 5.9 иллюстрирует потери соединения между узлами. БС позволяет гарантировать возможность соединения между узлами после определенного периода времени, когда производительность сети будет недостаточной, и использовать подходящее отношение R_c / R_s .

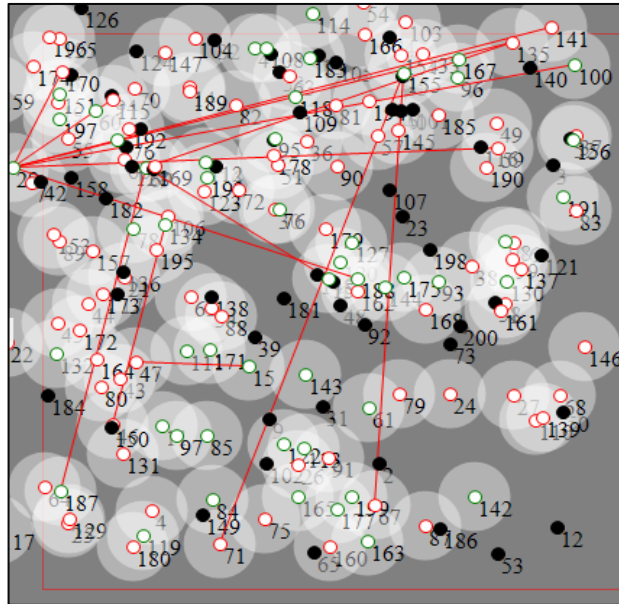


Рис. 5.9 – Потери соединения между узлами

Как видно из результатов моделирования, для обеспечения необходимой связности между узлами радиус связи должен быть, по крайней мере, в два раза больше радиуса покрытия в двумерных БСС.

В трехмерных БСС существует множество стратегий для размещения узлов, например, шестиугольная призма, додекаэдр или октаэдр. Для куба, в котором расстояние между любыми двумя соседними узлами равно $2R_s/\sqrt{3}$, радиус связи должен быть не менее $1,547R_s$ [73].

Выводы

1. Разработана методика размещения сенсорных узлов в двумерном и трехмерном пространствах, обеспечивающая, по крайней мере, 90% покрытие пространства в зависимости от соотношения плотности размещения и радиуса действия сенсорного узла R_s .

2. Размерность сенсорного поля существенно влияет на необходимую плотность БСУ для обеспечения заданной доли покрытия. При этом для трехмерных БСС необходимая плотность БСУ больше, чем для двумерных. Число близлежащих узлов для $3D$ БСС также значительно больше, чем для $2D$ БСС.

1. При равномерном случайном размещении БСУ в $3D$ пространстве для полносвязной сети необходима высокая плотность БСУ. При этом требуемый для $3D$ БСС радиус связи будет намного больше, чем для $2D$.

2. Для обеспечения необходимой связности между узлами радиус связи должен быть, по крайней мере, в два раза больше радиуса покрытия в $2D$ БСС.

3. В трехмерных БСС существует множество стратегий для размещения узлов, например, шестиугольная призма, додекаэдр или октаэдр. Для куба, где расстояние между любыми двумя соседними узлами составляет $2R_s/\sqrt{3}$, радиус связи должен быть не менее $1,547R_s$.

ЗАКЛЮЧЕНИЕ

Исследования, проведённые в диссертационной работе, позволили получить следующие основные результаты:

1. Анализ современного состояния в области исследований БСС, определение наиболее важных характеристик и структуры беспроводных сенсорных сетей показывают, что саморегулирующийся характер, особые условия по энергетической эффективности, устойчивость к отказам и большое значение числа узлов в БСС требуют разработки специфических алгоритмов маршрутизации для беспроводных сенсорных сетей.

2. Анализ существующих алгоритмов маршрутизации, самоорганизации и выбора головного узла кластера в беспроводных сенсорных сетях показывает, что требуется разработка новых эффективных алгоритмов кластеризации для мобильных и отказоустойчивых беспроводных сенсорных сетей.

3. Сравнительный анализ алгоритмов *DT*, *LEACH*, *SEP*, *DEEC*, *TEEN* показывает что *LEACH* алгоритм - крайне эффективный протокол для гомогенных БСС. Он помогает снизить энергозатраты в 7 раз по сравнению с прямым взаимодействием сенсорных узлов, а алгоритм *TEEN* превосходит по длительности периода стабильности и остаточной энергии все рассмотренные алгоритмы и рекомендуется к использованию как в гомогенных, так и в гетерогенных БСС.

4. На основе сочетания лучших свойств двух известных существующих алгоритмов – *DCA* и *MBC*, разработан адаптивный алгоритм кластеризации для БСС с мобильными узлами *MACA* (*Mobility Adaptive Clustering Algorithm*).

5. В разработанном алгоритме кластеризации мобильный сенсорный узел выбирает себя головным узлом кластера на основе простого точечного предиктора *SPP* (*Single Point Predictor*) для комбинированного критерия прогнозирования (алгоритм *DCA*). Другие узлы представляют

собой члены кластера, определяемые на основе некоторого значения, указывающего на их пригодность для подключения к конкретному головному узлу (алгоритм *MBC*).

6. Сравнение предложенного в диссертации алгоритма *MACA* на основе моделирования в системе *C#. NET* показало, что разработанный алгоритм по длительности жизненного цикла намного превосходит алгоритм *DCA*, который, в свою очередь, существенно превосходит *LEACH-M*. Алгоритм *MACA* превосходит алгоритм *DCA* также по длительности периода стабильности, а по числу успешно переданных пакетов оба алгоритма приблизительно равны.

7. На основе иерархического алгоритма кластеризации *TEEN* разработан отказоустойчивый алгоритм кластеризации для БСС *FT-TEEN*.

8. Разработанный алгоритм обеспечивает увеличение числа успешно переданных пакетов как от членов кластера к *CH*, так и от *CH* к базовой станции по сравнению с алгоритмом *TEEN*.

9. Разработанный алгоритм *FT-TEEN* обеспечивает толерантность к значению вероятности ошибки, в то время как для базового алгоритма *TEEN* число успешно переданных пакетов существенно снижается с ростом вероятности ошибки в рассмотренных пределах.

10. Размерность сенсорного поля существенно влияет на необходимую плотность БСУ для обеспечения заданной доли покрытия. При этом для трехмерных БСС необходимая плотность БСУ больше, чем для двумерных. Число близлежащих узлов для 3D БСС также значительно больше, чем для 2D БСС. При планировании БСС разработанная методика размещения БСУ позволяет обеспечить, по крайней мере, 90% покрытие для двумерных и трехмерных БСС.

11. Для обеспечения необходимой связности между узлами радиус связи должен быть, по крайней мере, в два раза больше радиуса покрытия в 2D БСС.

12. В трехмерных БСС существует множество стратегий для размещения узлов, например, шестиугольная призма, додекаэдр или октаэдр. Для куба, где расстояние между любыми двумя соседними узлами составляет $2R_s/\sqrt{3}$, радиус связи должен быть не менее $1,547R_s$.

По результатам исследования опубликовано 9 работ. Результаты диссертационной работы используются в СПбГУТ им. проф. М.А. Бонч-Бруевича при чтении лекций, проведении практических занятий и лабораторных работ по курсам “Современные проблемы науки в области телекоммуникаций” и “Интернет Вещей и самоорганизующиеся сети”.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БСС	Беспроводная сенсорная сеть
БС	Базовая станция
ВСС	Всепроникающая сенсорная сеть
ГЛОНАСС	Глобальная навигационная спутниковая система
ИВ	Интернет Вещей
МСЭ	Международный союз электросвязи
МСЭ-T	Сектор стандартизации электросвязи Международного союза электросвязи
ССОП	Сеть связи общего пользования
3G	технология сети подвижной связи третьего поколения
3GPP	3rd Generation Partnership Project
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union - Telecommunication sector
LEACH	Low-Energy Adaptive Clustering Hierarchy
LTE	Long Term Evolution
MatLab	Matrix Laboratory — пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете.
QoS	Quality of Service
RFID	Radio Frequency Identification
RSS	Received Signal Strength
TDMA	Time Division Multiple Access
USN	Ubiquitous Sensor Network
VANET	Vehicle Ad hoc NETWORK
WBAN	Wireless Body Area Network
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

СЛОВАРЬ ТЕРМИНОВ

Актор (actuator): Исполнительное устройство, воздействующее на окружающую среду в соответствии с получаемыми им командами.

Самоорганизация беспроводных сенсорных сетей (Wireless sensor network self-organization): изменения, которые должен осуществить каждый из узлов БСС в своем поведении, для того, чтобы взаимодействовать с соседними узлами для выполнения определенной задачи или достижения определенной цели.

Беспроводная или всепроникающая сенсорная сеть (Wireless/Ubiquitous Sensor Network): Самоорганизующаяся сеть связи, объединяющая большое количество сенсорных узлов с низким потреблением энергии, передающих на шлюз (ы) сети данные о состоянии окружающей среды или расположенных в ней объектов.

Вещь (thing): Применительно к интернету вещей означает предмет физического или информационного мира, который может быть идентифицирован и интегрирован в сети связи.

Головной узел кластера всепроникающей сенсорной сети (Ubiquitous sensor network cluster head node): узел, осуществляющий функцию коллектора данных для сенсорных узлов, входящих в кластер, передающий собранные данные в общие для всей сети шлюзы, а также в некоторых случаях осуществляющий функцию координации взаимодействия узлов кластера.

Интернет Вещей (Internet of Things): Концепция развития сетей связи, представляющая глобальную инфраструктуру для информационного общества, которая обеспечивает возможность предоставления услуг путем соединения друг с другом (физических и виртуальных) вещей на основе существующих и развивающихся функционально совместимых информационно-коммуникационных технологий.

Кластер всепроникающей сенсорной сети (Ubiquitous sensor network cluster): часть всепроникающей сенсорной сети, представляющая собой

группу сенсорных узлов, осуществляющих передачу данных в головной узел кластера и только посредством узлов, входящих в тот же кластер.

Сенсор (sensor): Устройство, предназначенное для сбора данных об окружающей среде.

Сенсорное поле (sensing field): Пространство, покрываемое всепроникающей сенсорной сетью.

Сенсорный узел (sensor node): Устройство, состоящее из сенсора, блока обработки информации, блока приема и передачи данных по беспроводному каналу, а также источника питания. Сенсорный узел является частью сенсорной сети и предназначен для фиксации состояния окружающей среды, обработки полученных данных и передачи их шлюзу сенсорной сети.

Шлюз всепроникающей сенсорной сети (Wireless sensor network gateway): Устройство, осуществляющее преобразование протоколов между всепроникающей сенсорной сетью и другими сетями связи, которое также может выполнять ряд других функций.

Список литературы

1. Абакумов П. А. Алгоритм выбора головного узла кластера сенсорной сети в трехмерном пространстве. Электросвязь, №4, 2014.
2. Абакумов П. А. Алгоритм кластеризации для мобильных беспроводных сенсорных сетей в трехмерном пространстве/ П. А. Абакумов, А.Е.Кучерявый // Электросвязь, №9, 2015.
3. Аль-Кадами Н. А. Оценка и сравнительный анализ алгоритмов маршрутизации для гомогенных и гетерогенных беспроводных сенсорных сетей/ Н. А. Аль-Кадами // Информационные технологии и телекоммуникации. Электронный научный журнал. – СПб ГУТ, 2014. – Вып. 3 (7) с. 4-22.
4. Аль-Кадами Н. А. Адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами / Н. А. Аль-Кадами, А. Е. Кучерявый // Электросвязь, №3, 2015.
5. Богданов И. А. Характеристики жизненного цикла мобильной сенсорной сети при различных потоках ложных событий / И. А. Богданов, А. И. Парамонов, А. Е. Кучерявый // Электросвязь. – 2013. – №1.
6. Гольдштейн Б.С. Сети связи пост-NGN / Б.С. Гольдштейн, А.Е. Кучерявый // БХВ, С.Петербург, 2013.
7. Кучерявый А. Е., Владыко А. Г., Киричек Р. В. и др. Летающие сенсорные сети // Электросвязь. — 2014. — № 9.
8. Кучерявый А. Е. Теоретические и практические направления исследований в области летающих сенсорных сетей / А. Е. Кучерявый, А. Г. Владыко, Р. В. Киричек // Электросвязь, №9, 2015.
9. Кучерявый А.Е. Интернет Вещей / А.Е. Кучерявый // Электросвязь, №1, 2013.
10. Кучерявый А.Е. От е-России к и-России: тенденции развития электросвязи / А.Е. Кучерявый, Е.А. Кучерявый // «Электросвязь», №5, 2005.

11. Кучерявый А.Е. Сети связи общего пользования / А.Е. Кучерявый, А.И. Парамонов, Е.А. Кучерявый //Тенденции развития и методы расчёта. ФГУП ЦНИИС, 2008.
12. Кучерявый А.Е. Самоорганизующиеся сети/ А.Е. Кучерявый, А.В. Прокопьев, Е.А. Кучерявый // СПб, “Любавич”, 2011.
13. Мочалов В.А. Построение отказоустойчивой структуры беспроводной сенсорной сети с учетом отказов центров сбора информации / В.А. Мочалов, Е.Н. Турута // Труды IS&IT’12. Научное издание в 4-х томах. М., Физматлит, 2012., т.2.
14. Росляков А.В. Интернет Вещей / А.В. Росляков, С.В. Ванияшин, А.Ю. Гребешков, М.Ю. Самсонов // ПГУТИ, Самара, 2014.
15. Abakumov, P. The Cluster Head Selection Algorithm in the 3D USN / P. Abakumov, A. Koucheryavy // Proceedings, International Conference on Advanced Communication Technology, 2014. ICACT 2014. Phoenix Park, Korea.
16. Abdulsalam, H.M. W-LEACH: Weighted Low Energy Adaptive Clustering Hierarchy aggregation algorithm for data streams in wireless sensor networks / H.M. Abdulsalam, L.K. Kamel // Proceedings of IEEE International Conference on Data Mining Workshops (ICDMW); Sydney, Australia. 14 December 2010; 1–8 pp.
17. Akyildiz, I.F., A Survey on Sensor Networks / I.F. Akyildiz, W. Su ,Y. Sankarasubramaniam , and E.Cayirci // IEEE Communications Magazine. - August 2002. - Vol. 40. - N 8. - 102–114 pp.
18. Compare of Routing Algorithms for Homogeneous and Heterogeneous Wireless Sensor Networks / N.A. Al-Kadami // (TELECOM IT) – УДК. 621.391, С. 4–22 (2014).
19. Al-Qadami N. Mobility Adaptive Clustering Algorithm for Wireless Sensor Networks with Mobile Nodes / N. Al-Qadami, I. Laila, A. Koucheryavy, A.S. Ahmad// ICACT-2015, Proceedings, 1-3 July, Phoenix Park, Korea.

20. Antoine Bagula, Application of Wireless Sensor Networks, A Talk Presented at WSN Applications, UCT, February 2012.
21. Antonio J. Extending the internet of things to the future internet through IPv6 support / J. Jara Antonio, Varakliotis Socrates, F. Antonio, Gmez-Skarmeta, T. Kirstein Peter // J. Mobile Inf. Syst. 10 (1) (2014) 3–17 pp.
22. Arati. M. A. TEEN: a routing protocol for enhanced efficiency in wireless sensor networks /. M Arati., and. P. A. Dharma., // in Proceedings of the 15th International Parallel & Distributed Processing Symposium, pp. 2009–2015, San Francisco, Calif, USA, April 2001.
23. Asharioun H. Lifetime Comparison on Location Base Routing in Wireless Sensor Networks / H. Asharioun, H. Asadollahi, A.S. Ismail, S. Ramadass // International Journal of Information and Education Technology. – April 2012. – Vol. 2, №2.
24. Attarzadeh N. A New Threee Dimensional Clustering Method for Wireless Sensor Networks / N. Attarzadeh, M. Mehrani // Global Journal of Computer Science and Technology. V.11, issue 6, version 1.0, April 2011.
25. Bellalouna M. A priori methods for fault tolerance in wireless sensor networks,"/ M. Bellalouna; A. Ghabri // Computer and Information Technology (WCCIT), 2013 World Congress on, vol., no., pp.16, 22-24 June 2013.
26. Ben Mansour T. GMAC: Group Mobility Adaptive Clustering Scheme for Mobile Wireless Sensor Networks / T. Ben Mansour, S. Moussaoui // International Symposium on Programming and Systems (ISPS) / Proceedings, Algiers, Algeria. – 25–27 April 2011.
27. Bhattassali T. Sleep Deprivation Attack Detection in Wireless Sensor Networks / T. Bhattassali, R. Chaki, S. Sanyal // International Journal of Computer Applications, v.40, №15, February 2012.
28. Bogdanov I. The mobile Sensor Network Life-Time under Different Spurious Flows Intrusion / I. Bogdanov, A. Koucheryavy, A. Paramonov // Internet of Things, Smart Spaces, and Next Generation Networks and Systems / 14th International conference, NEW2AN 2013 and 6th Conference,

- ruSMART 2013. LNCS 8121, Springer.— Russia, St. Petersburg. – 27–29 August 2013.
29. Boulanouar I. Energy-aware object tracking algorithm using heterogeneous Wireless Sensor Networks / I. Boulanouar, A. Rachedi., Stéphane Lohier, Gilles. Roussel // in: 4th IFIP/IEEE WD'2011, Ontario, Canada, October 2011.
 30. Culler D. Overview of sensor networks / D. Culler, D. Estrin, M. Srivastava // IEEE J. Comput.Soc. 37 (8) (2004) 41–49.
 31. Deng S. Mobility-based clustering protocol for wireless sensor networks with mobile nodes / S. Deng, J. Li, L. Shen // IET Wireless sensor system, 1(1): 39 - 47. DOI: 10.1049/iet-wss.2010.0084.
 32. Deng S. Mobility-based clustering protocol for wireless sensor networks with mobile nodes / S. Deng, J. Li, L. Shen // Hunan City University, Yiyang, Hunan 413000, 10.1049/iet-wss.2010.0084, 2011.
 33. Dietrich I. On the Lifetime of Wireless Sensor Networks / I. Dietrich, F. Dressler // Department of Computer Science, University of Erlangen, Germany – 2009.
 34. Doo-Soon Park. Fault Tolerance and Energy Consumption Scheme of a Wireless Sensor Network / International Journal of Distributed Sensor Networks, Volume 2013, Article ID 396850, 7 pp.
 35. Fan X. Improvement on LEACH Protocol of Wireless Sensor Network. Proceedings of International Conference on Sensor Technologies and Applications / X. Fan, Y. Song //Valencia, Spain. 14–20 October 2007; pp. 260–264.
 36. Ghosha A. Coverage and connectivity issues in wireless sensor networks: a survey / A. Ghosha, S.K. Dasb, // J. PMC 4 (3) (2008) 303–334.
 37. Goel A. and Rai S., Krishnamachari B. Sharp thresholds for monotone properties in random geometric graphs / Annals of Applied Probability. – 2006.

38. Govindan Ramesh, Estrin Deborah. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks / Ramesh Govindan, Deborah Estrin // Chalmers Intanagonwiwat, ACM New York, NY, USA ©2000.
39. Hai Liu. Fault-Tolerant Algorithms/Protocols in Wireless Sensor Networks / Liu. Hai, Amiya. Nayak and Stojmenovi Ivan // Chapter 10 Guide to Wireless Sensor Networks, Computer Communications 261 and Networks, DOI: 10.1007/978-1-84882-218-410, Springer-Verlag London Limited 2009.
40. Heinzelman W. B. “Application-specific protocol architectures for wireless networks,” Ph.D. dissertation, Massachusetts Institute of Technology, 2000.
41. Heinzelman W. Energy-efficient communication protocol for wireless micro sensor networks / W. Heinzelman, A. Chandrakasan, H. Balakrishnan // Proceedings 33rd Hawaii International Conference on System Sciences (HICSS), Wailea Maui, Hawaii, USA, Jan. 2000.
42. Heinzelman W.B. Application specific protocol architecture for wireless microsensor networks / W. B. Heinzelman, A. Chandrakasan, H. Balakrishnan // IEEE Trans. Wirel. Commun. 2002; 1: 660–670.
43. Heinzelman W.R. Application-Specific protocol architectures for wireless networks / W. B. Heinzelman // [D]. – Boston: MIT, Doctor Thesis. – 2000.
44. Heinzelman W.R. et all. Energy-Efficient Communication Protocol for Wireless Microsensor Networks / Proceeding of the 33rd Hawaii International Conference on System Sciences. – January 2000.
45. Hill J.L. System Architecture for Wireless Sensor Networks / J.L. Hill // University of California, Berkeley, Spring 2003.
46. Hla Yin M. Fault Management Using Cluster-Based Protocol in Wireless Sensor Networks / Yin M. Hla, Z. Win // International Journal of Future Computer and Communication vol. 2, 2014, no. 6, pp. 36-39.
47. Hong J. T-LEACH: The method of threshold-based cluster head replacement for wireless sensor networks / J. Hong, J. Kook, S. Lee, D. Kwon, S. Yi // Inf. Syst. Front. 2009; 11: 513–521.

48. http://www.routingprotokolle.de/Routing/mobility_random_walk.htm
49. Kim D. Self-Organization Routing Protocol Supporting Mobile Nodes for Wireless Sensor Network / D. Kim and Y. Chung // Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences, Volume 2, 2006.
50. Kim, D. Self-Organization Routing Protocol Supporting Mobile Nodes for Wireless Sensor Networks /, D. Kim, Y. Chung // IMSCCS'06. Proceedings. Zhejiang University, China, June 20-24, 2006.
51. Koucheryavy A. Prediction-based Clustering Algorithm for Mobile Wireless Sensor Networks / A. Koucheryavy, A. Salim //, Proc IEEE ICACT-10, pp 1203-1208, 2010.
52. Koucheryavy A. Prediction-based Clustering Algorithm for Mobile Wireless Sensor Networks / A. Koucheryavy, A. Salim // Proceedings, International Conference on Advanced Communication Technology, 2010. ICACT 2010. Phoenix Park, Korea.
53. Koucheryavy A. Cluster head selection for homogeneous Wireless Sensor Networks / A. Koucheryavy, A. Salim // Proceedings, International Conference on Advanced Communication Technology, 2009. ICACT 2009. Phoenix Park, Korea.
54. Koucheryavy A. Cluster-based Perimeter-coverage Technique for Heterogeneous Wireless Sensor Networks / A. Koucheryavy, A. Salim // Proceedings, ICUMT 2009 International Conference IEEE on Ultra-Modern Telecommunications, Saint-Petersburg, Russian –2009.
55. Koucheryavy A. Prediction-based Clustering Algorithm for Mobile Wireless Sensor Networks / A. Koucheryavy, A. Salim // Proceedings, International Conference on Advanced Communication Technology, 2010. ICACT 2010. Phoenix Park, Korea.
56. Koushanfar.F. Fault Tolerance in Wireless Sensor Networks /.F. Koushanfar,.M. Potkonjak, and A. Sangiovanni-Vincentelli // in Handbook of Sensor

- Networks, I. Mahgoub and M. Ilyas (Eds.), CRC press, Section VIII, no. 36, 2004.
57. Krishnamachar B. Critical Transmission Range for Channel Allocation in Multi-hop Wireless Networks / B. Krishnamachar, S. Wicker // University of Southern California, Cornell University. – July 2002.
 58. Lai W.K. Arranging cluster sizes and transmission ranges for wireless sensor networks / W.K. Lai, C.S. Fan, L.Y. Lin // Inf. Sci. 2012; 183:117–131.
 59. Li D. Detection, classification and tracking of targets in distributed sensor networks / D. Li, K. Wong, Y.H. Hu, A. M. Sayeed // IEEE Signal Processing Magazine. – Mar. 2002. – 19(2).
 60. Liu B. Mobility improves coverage of sensor networks / B. Liu, P. Brass, O. Dousse et al. // Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'05, Urbana-Champaign, IL. – May 2005.
 61. Liu B. Mobility Improves Coverage of Sensor Networks / B. Liu, P. Brass, O. Dousse, P. Nain, D. Towsley // ACM (MobiHoc'05), Illinois, USA, May 2005.
 62. Liu J. A Dual-Space Approach to Tracking and Sensor Management in Wireless Sensor Networks / J. Liu, P. Cheung, L. Guibas, F. Zhao // Proc. First ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA '02). – Sept. 2002. 131–139 pp.
 63. Liu L. On coverage of wireless sensor networks for rolling terrains. IEEE Transactions on Parallel and Distributed Systems / L. Liu, H. Ma // 23(1): 2012, 118–125 pp.
 64. Lopez P. Survey of Internet of Things Technologies for Clinical Environments / P. Lopez, D. Fernandez, A.J. Jara, A.F. Skarmeta, // AINA Workshops, 2013, pp. 1349–1354 pp.
 65. Loscri V. A Two-Level Hierarchy for Low-Energy Adaptive Clustering Hierarchy / V. Loscri, G. Morabito, S. A. Marano // Proceedings of the 2nd

- IEEE Semiannual Vehicular Technology Conference; Dallas, TX, USA. 25–28 September 2005; 1809–1813 pp.
66. Luís M.L. Routing and mobility approaches in IPv6 over LoWPAN mesh networks / M.L. Oliveira Luís, F. Amaro, Sousa de, J.P.C. Rodrigues Joel // IJCS 24 (11) (2011) 1445–1466 pp.
67. Manjeshwar A. TEEN: A Protocol for Enhanced Efficiency in Wireless Sensor Network / A. Manjeshwar, D. P. Agrawal, //1st international Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, 2001, p.189.
68. Manjeshwar A. and Grawal D.P. TEEN: A protocol for enhanced efficiency in wireless sensor networks / Proceedings of the 15th Parallel and Distributed Processing Symposium. – San Francisco. – 2001. –Vol. 3.
69. May M. Design of a Wireless Sensor Node Platform / M. May // University of Waikato. – 2012.
70. Meenakshi Panda. Energy Efficient Distributed Fault Identification Algorithm in Wireless Sensor Networks / Panda Meenakshi and P. M. Khilar // Journal of Computer Networks and Communications, Volume 2014 (2014), Article ID 323754, 16 pp.
71. Mhatre Vivek. Homogeneous vs Heterogeneous Clustered Sensor Networks: A Comparative Study / Vivek Mhatre, Catherine Rosenberg // School of Electrical and Computer Eng., Purdue University, West Lafayette, IN 47907-1285.
72. Misra S. Connectivity preserving localized coverage algorithm for area monitoring using wireless sensor networks / S. Misra, K. Manikonda, M. C. Obaidat // Kharagpur, India, Department of Computer Science, Monmouth University, New Jersey, USA. — 2011.
73. Nazrul Alam S. M. Coverage and Connectivity in Three-Dimensional Networks / Alam S. M. Nazrul, Z. J. Haas //Cornell University, MobiCom'06, Los Angeles, California, USA. – September 23–29. – 2006.

74. Ph.D. Dissertation. Zhejiang University; Hangzhou, China: 2006. Mo Xiaoyan. Study and Design on Cluster Routing Protocols of Wireless Sensor Networks.
75. Prasanna S. An Overview of Wireless Sensor Networks Applications and Security / S. Prasanna, S. Rao // IJSCE, vol-2(2), May 2012, ISSN: 2231–2307 pp.
76. Qing Li. Design of a distributed energy-efficient clustering algorithm for heterogeneous wireless sensor networks / Li. Qing, Qingxin Zhu, Mingwen. Wang // Computer Communications 29 (2006) 2230–2237 pp.
77. Nuno R. Mobility support in Low Power Wireless Personal Area Networks (MLoWPAN) / R Nuno., da. M Silva // Coimbra University, Portugal, 2010.
78. Radwan A. A. A Study of Critical Transmission Range for Connectivity in Ad Hoc Network / A. A. Radwan, M. H. Mohamed, M. A. Mofaddel, H.H. El-Sayed // Minia University, Assiut Univesity, Sohag University, Egypt. – 1 May 2013.
79. Raghavendra C. S. Distributed Signal Processing in Wireless Sensor Networks / C. S. Raghavendra, V. K. Prasanna // ADA437824. – August 2005. 17.
80. Ran G. Improving on LEACH protocol of wireless sensor networks using fuzzy logic / G. Ran, H. Zhang, S. Gong // J. Inf. Comput. Sci. 2010; 7: 767–775.
81. Recommendation ITU-T Y.2052. Framework of multi-homing in IPv6-based NGN. Geneva, February 2008.
82. Römer K. The design space of wireless sensor networks / K. Römer, F. Mattern // IEEE Wireless Commun. 11 (6) (2004) 54–61.
83. Samer A. B. Cluster based routing protocol for mobile nodes in wireless sensor network / A. B. Samer, N. K. CheeK, F. R. Mohd et al. //Proceedings, Third International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks. – Waterloo, Ontario, Canada. – August 2006.

84. Smaragdakis G. SEP: A Stable Election Protocol for clustered heterogeneous wireless sensor networks / G. Smaragdakis, I. Matta, A. Bestavros // in: Second International Workshop on Sensor and Actor Network Protocols and Applications (SANPA 2004), 2004.
85. Stankovic J.A. Realistic applications for wireless sensor networks / J.A. Stankovic, A.D. Wood, T. He // Theoretical Comput. Sci. (2011) 835–863.
86. Stankovic Stanislava. Medical applications based on Wireless Sensor Networks / Stanislava Stankovic // Trans. J. Int. Res. 5 (2) (2009) 19–23.
87. Varshney P. Distributed Detection and Data Fusion / P. Varshney // Springer-Verlag. New York, NY. – 1996.
88. Wang J. A Distance-Based Energy Aware Routing Algorithm for Wireless Sensor Networks / J. Wang, J.U. Kim, L. Shu et al. // Sensors (Basel). – October 2010. – Vol.10, issue 10.
89. Wang Y. C. Mobility Management Algorithms and Applications for Mobile Sensor Networks / Y. C. Wang, F. J. Wu, Y.C. Tseng // Wireless Communications and Mobile Computing. – 2012. – Vol.12, issue 1.
90. Welsh M. Wireless sensor networks for emergency Medical Care / M. Welsh, D. Malan, B. T. Duncan // Harvard and Boston Universities, A Talk Presented at GE Global Research, March 2004.
91. Xia H. Radio propagation characteristics for line-of-sight micro cellular and personal communications / H. Xia, H. L. Bertoni, L.R. Maciel // Antenna and Propagation, vol.41, no.10, pp.1439–1447, October1993.
92. Yassein M.B. Improvement on LEACH protocol of wireless sensor network / M. B. Yassein, A. Al-zou'bi, Y. Khamayseh, W. Mardini //(VLEACH) Int. J. Digit. Content Technol. Appl. 2009; 3: 132–136.
93. Younis O. HEED: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks/ O. Younis, S. Fahmy // IEEE Transactions on Mobile Computing. – 2004. – №3 (4).
94. Zach Shelby. 6LoWPAN: The Wireless Embedded Internet/ Shelby Zach, Bormann Carsten // Book, IETF (6LoWPAN-WG), 2009.

СПИСОК ИЛЛЮСТРАТИВНОГО МАТЕРИАЛА

- Рис. 1.1 – Структура БСС
Рис. 1.2 – Сенсорные узлы
Рис. 1.3 – Типовая архитектура сенсорного узла
Рис. 1.4 – Самоорганизация и передача данных в беспроводных сенсорных сетях
Рис. 1.5 – Приложения беспроводных сенсорных сетей
Рис. 1.6 – Классификация БСС в зависимости от структуры сети
Рис. 1.7 – Одношаговое и многошаговое взаимодействие в БСС
Рис. 1.8 – Гетерогенная модель БСС
Рис. 1.9 – Кластерная БСС
Рис. 1.10 – Блок-схема протоколов маршрутизации на основе кластеризации (процесс кластеризации)
Рис. 1.11 – Кластеризация с одноранговой сетью головных узлов (а) и кластеризация с многошаговой сетью головных узлов (б).
Рис. 2.1 – Категории протоколов маршрутизации в БСС
Рис. 2.2 – Алгоритм прямой передачи в БСС
Рис. 2.3 – Алгоритм наводнения в БСС
Рис. 2.4 – Топология LEACH
Рис. 2.7 – Случайное расположение узлов
Рис. 2.8 – Прямая передача. Начальное состояние.
Рис. 2.9 – Состояние после 180 раундов
Рис. 2.10 – Кластеризация БСС
Рис. 2.11 – Случайное расположение узлов
Рис. 2.12 – Длительность периода стабильности
Рис. 2.13 – Остаточная энергия
Рис. 2.14 – Число живущих узлов
Рис. 2.15 – Число погибших узлов
Рис. 2.16 – Случайное размещение узлов в гетерогенной сети
Рис. 2.17 – Длительность периода стабильности
Рис. 2.18 – Остаточная энергия.
Рис. 2.19 – Число живущих узлов
Рис. 2.20 – Число погибших узлов
Рис. 2.21 – Длительность периода стабильности
Рис. 2.22 – Остаточная энергия
Рис. 2.23 – Длительность периода стабильности во всех исследованных случаях
Рис. 3.1 – БСУ оставляет старый кластер и присоединяется к новому
Рис. 3.2 – Экранные формы при моделировании:
Рис. 3.3 – Длительность жизненного цикла для *MACA*, *DCA* и *LEACH-M*
Рис. 3.4 – Период стабильности для *MACA* и *DCA*
Рис. 3.5 – Среднее число успешно переданных пакетов для *MACA* и *DCA*

Рис. 4.1 – Кластеризация БСС.

Рис. 4.2 – Блок-схема для предложенного алгоритма

Рис. 4.3 – Кластеризация в разработанном алгоритме

Рис. 4.4 – Блок-схема формирования кластера для разработанного алгоритма

Рис. 4.5 – Блок-схема передачи данных для разработанного алгоритма

Рис. 4.6 – Число успешно переданных пакетов от членов кластера на *СН*.

Рис. 4.7 – Число успешно переданных пакетов от *СН* на базовую станцию каждый раунд.

Рис. 4.8 – Число успешно переданных пакетов от членов кластера на *СН*

Рис. 4.9 – Число успешно переданных пакетов от *СН* на базовую станцию.

Рис. 4.10 – Число успешно переданных пакетов от членов кластера на *СН*

Рис. 4.11 – Число успешно переданных пакетов от *СН* на базовую станцию

Рис. 5.1 – БСС в двумерном и трехмерном пространствах:

Рис. 5.2 – Трехмерная сеть включает двумерные сети

Рис. 5.3 – Соотношения между радиусом покрытия R_s , долей покрытия C и плотностью ρ в $2D$ БСС

Рис. 5.4 – Соотношения между радиусом покрытия , долей покрытия и плотностью в $3D$ БСС

Рис. 5.5 – Сравнительные характеристики для $2D$ и $3D$ БСС

Рис. 5.6 – Доля покрытия C как функция плотности ρ в $2D$ и $3D$ БСС при

Рис. 5.7 – Соотношения между долей покрытия C и числом близлежащих сенсорных узлов, находящихся на расстоянии меньше $2R_s$, при случайном размещении узлов для $2D$ и $3D$ БСС

ПРИЛОЖЕНИЕ А. АЛГОРИТМ МАСА

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace mmcc.GreedyVor
{
    public class CCDDCA
    {
        public Zone zone;
        public int rcl;
        public double alpha, beta, gamma, eta;
        public CCDDCA(ref Zone z, double alpha, double beta, double gamma, double eta, int rcl)
        {
            zone = z;

            setParam( alpha, beta, gamma, eta, rcl);
        }
        public void setParam(double alpha, double beta, double gamma, double eta, int rcl)
        {
            this.alpha=alpha;
            this.beta=beta;
            this.gamma=gamma;
            this.eta = eta;
            this.rcl = rcl;
        }
        public void MCCcalculations(Node sensor, List<Node> neighbors)
        {
            sensor.coverageAngleList.Clear();

            double def_angle = new double();
            CoverageAngle defaultNode = new CoverageAngle();
            Console.WriteLine("-----");
            Console.WriteLine("----| Mcc Calculations for node: " + sensor.id + " |-----");
            foreach (Node cNode in neighbors)
            {
                /*
                 * http://howtoproperly.com/rotate-points-using-polar-coordinates/
                 How to Convert
                 First, make the origin the point of rotation by performing what is called a transformation:
                 placing a new origin and adjusting all of the coordinates to reflect their new spatial relationship to the origin.
                 You may then apply to your coordinates the rule of Pythagoras to determine the distance from the origin to the point.
                 r = the square root of x2+y2.
                 And the angle q = tan-1(y/x).
                 */
                ////////////
                ////////// Search for default node that has greatest cover range
                //if (cNode.sensing_radius > defaultNode.sensing_radius && cNode.occupation == 0) defaultNode = cNode;
                Point def_p1 = new Point();
                Point def_p2 = new Point();
                double theta = new double();
                //// get the intersection information between query circle region
                //// and the default node
                if (intersect(sensor, cNode, out def_p1, out def_p2))
                // if (intersect(sensor, sensorB, out def_p1, out def_p2))
                // move interesction point coordinates to node center i.e. use node coordinates
                def_p1.X = def_p1.X - sensor.position.X;
                def_p1.Y = def_p1.Y - sensor.position.Y;
                def_p2.X = def_p2.X - sensor.position.X;
                def_p2.Y = def_p2.Y - sensor.position.Y;
                //get slop angle of each intersection point
                //def_angle = Math.Atan2((double)def_p1.Y, (double)def_p1.X);
                double angle1 = Math.Atan2((double)def_p1.Y, (double)def_p1.X);
                double angle2 = Math.Atan2((double)def_p2.Y, (double)def_p2.X);
                // from -pi <angle< pi convert to 0<angle<2pi
                angle1 += Math.PI;
                angle2 += Math.PI;
                //covert to degree
                angle1 *= (180 / Math.PI);
                angle2 *= (180 / Math.PI);
                // calculate theta
                if (angle2 < angle1)
                {
```

```

        //if (angle1 >= 270 && angle1 <= 360)
        theta = Math.Abs((angle2 + 360) - angle1);
    }
    else
        theta = Math.Abs(angle2 - angle1);
    //Console.WriteLine("node " + cNode.id + " angle1: " + angle1 + " angle2: " + angle2 + " theta:" + theta);
    //Console.WriteLine("node " + cNode.id + " point1:( " + def_p1.X + ", " + def_p1.Y + ") point2: (" + def_p2.X + ", " +
def_p2.Y + ") theta:" + theta);
    CoverageAngle cangle = new CoverageAngle();
    cangle = new CoverageAngle(cNode, angle1, angle2, theta);
    if (!sensor.coverageAngleList.Contains(cangle))
        sensor.coverageAngleList.Add(cangle);
    if (theta > defaultNode.angle)
    {
        defaultNode.node = cNode;
        defaultNode.angle = theta;
        defaultNode.startAngle = angle1;
        defaultNode.endAngle = angle2;
        def_angle = angle1; /* (Math.PI / 180);
    }
    Console.WriteLine(">Neighbor : " + cNode.id + " with start angle: " + angle1 + " & end angle" + angle2);
}
}
}
// check interesection between two circles
bool intersect(Node cA, Node cB, out Point intersectP1, out Point intersectP2)
{
    intersectP1 = new Point();
    intersectP2 = new Point();
    //cB.position.X = Math.Abs(cB.position.X - cA.position.X);
    //cB.position.X = Math.Abs(cB.position.Y - cA.position.Y);
    double dx = (double)cA.position.X - (double)cB.position.X;
    double dy = (double)cA.position.Y - (double)cB.position.Y;
    double d2 = dx * dx + dy * dy;
    double d = Math.Sqrt(d2);

    if (d > cA.sensing_radius + cB.sensing_radius || d < Math.Abs(cA.sensing_radius - cB.sensing_radius)) return false; // no solu-
tion

    double a = (cA.sensing_radius * cA.sensing_radius - cB.sensing_radius * cB.sensing_radius + d2) / (2 * d);
    double h = Math.Sqrt(cA.sensing_radius * cA.sensing_radius - a * a);
    double x2 = cA.position.X + a * (cB.position.X - cA.position.X) / d;
    double y2 = cA.position.Y + a * (cB.position.Y - cA.position.Y) / d;
    double paX = x2 + h * (cB.position.Y - cA.position.Y) / d;
    double paY = y2 - h * (cB.position.X - cA.position.X) / d;
    double pbX = x2 - h * (cB.position.Y - cA.position.Y) / d;
    double pbY = y2 + h * (cB.position.X - cA.position.X) / d;

    //Console.WriteLine(paX + ", " + paY);
    //Console.WriteLine(pbX + ", " + pbY);
    intersectP1.X = Convert.ToInt32(paX); intersectP1.Y = Convert.ToInt32(paY);
    intersectP2.X = Convert.ToInt32(pbX); intersectP2.Y = Convert.ToInt32(pbY);
    return true;
}
public Node determineMaxCC( List<Node> neighbors)
{
    Node MaxCC = new Node(); ;
    foreach (Node s in neighbors)
    {
        if (s.CCValue > MaxCC.CCValue)
            MaxCC = s;
    }
    return MaxCC;
}
private void runCalculateCCValue(Node sensor)
{
    MCCcalculations(sensor, sensor.neighbors);

    sensor.CCValue = sensor.neighbors.Count * alpha + sensor.CR * beta + gamma * (1 / (1 + sensor.MobilitySpeed)) + eta * sen-
sor.energy;
}
Sensor.Stability value= W_(ij)=a* E_(j-current)/(E_max *N_(j-current))+ b*(1- d_ij/R_tran)+ c* [(Δt)]_ij/t_frame
// get neighbor at given distance
private List<Node> getNeighborsAt(Node sensor, int hop)
{
    List<Node> distantNeighbors = new List<Node>();
    List<Node> temp = new List<Node>();

```

```

temp.Add(sensor);
while (hop > 0)
{
    for (int i=0;i< temp.Count;i++)
    {
        Node s= temp[i];
        foreach (Node node in s.neighbors)
        {
            if (!distantNeighbors.Contains(node) && node.id != sensor.id)
                distantNeighbors.Add(node);
        }
    }
    if(temp.Count>0) temp.RemoveAt(0);
    foreach (Node node in distantNeighbors)
        if (node.id != sensor.id && !temp.Contains(node)) temp.Add(node)
    hop--;
}

return distantNeighbors;
}
private void CCD(Node sensor, int rcl)
{
    Console.WriteLine(" ----- Current Node:" + sensor.id);

    List<Node> distantNeighbors = getNeighborsAt(sensor, rcl);
    Node maxCCNode=determineMaxCC( distantNeighbors);
    if (maxCCNode.CCValue > sensor.CCValue) sensor.MaxCC = maxCCNode;
    foreach (Node s in distantNeighbors)
    {
        Console.WriteLine("Node ID = " + s.id );
    }
}

}
public void runInitialPhase()
{
    foreach (Node s in zone.network)
    {
        runCalculateCCValue(s);
    }
    foreach (Node s in zone.network)
    {
        CCD(s, this.rcl);
    }

    foreach (Node s in zone.network)
    {
        Console.WriteLine("Node ID = " + s.id + " max id=" + s.MaxCC.id + " value:=" + s.MaxCC.CCValue );
        s.MaxCC.occupation = 1;
    }
}
}
}

```

Line 2D

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace mmcc.GreedyVor
{
    public class Line2D
    {
        private Point pointOne;
        private Point pointTwo;
        public int getX1() { return pointOne.X; }
        public int getX2() { return pointTwo.X; }
        public Point getStartPoint() { return pointOne; }
        public Point getEndPoint() { return pointTwo; }
    }
}

```

```

public int getY1() { return pointOne.Y; }
public int getY2() { return pointTwo.Y; }

public Line2D(Point pointOne, Point pointTwo)
{ this.pointOne = pointOne; this.pointTwo = pointTwo; }

```

```

/*
 * 2D Line intersection test:
 * By: Christopher Bartlett

```

Steps to Solve:

Basic Idea is to test the triangles formed by the end point of two line segments. Test the two triangles formed by Line one to line two's end points. If first two checks are same clockness then there is no possible intersect of line segment. If Clockness is different then check line two to one. If Clockness is different then Lines intersect.

check_tri_clock_dir does a determinate type calculation to find the clockness of a triangle made of the three points passed in. Could be Clockwise, CounterClockwise and Line (meaning the two line segments lie along the same line)
check_intersect return 1 for intersect and 0 for none.

```

*/
int check_tri_clock_dir(Point pt1, Point pt2, Point pt3)
{
    float test;
    test = (((pt2.X - pt1.X) * (pt3.Y - pt1.Y)) - ((pt3.X - pt1.X) * (pt2.Y - pt1.Y)));
    if (test > 0) return 1;
    else if (test < 0) return -1;
    else return 0;
}

public bool intersectsLine(Line2D line2)
{
    int test1_a, test1_b, test2_a, test2_b;

    test1_a = check_tri_clock_dir(this.pointOne, this.pointTwo, line2.pointOne);
    test1_b = check_tri_clock_dir(this.pointOne, this.pointTwo, line2.pointTwo);
    if (test1_a != test1_b)
    {
        test2_a = check_tri_clock_dir(line2.pointOne, line2.pointTwo, this.pointOne);
        test2_b = check_tri_clock_dir(line2.pointOne, line2.pointTwo, this.pointTwo);
        if (test2_a != test2_b)
        {
            return true;
        }
    }
    return false;
}
}
/**
 * converted from
 * http://www.koders.com/java/fid1182C9330FDC66A60312A33932E82F09B5C4CA9F.aspx?s=graph
 * The Intersection class provides methods for determining
 * the intersection point of two lines and the intersection
 * point of a line and a GeneralPath object.
 *
 * @author Jesus M. Salvo Jr.
 */
public class Intersection {

    public bool getIntersection( Line2D line1, Line2D line2, out Point intersectionPoint )
    {
        intersectionPoint = new Point();
        double dyline1, dxline1;
        double dyline2, dxline2, e, f;
        double x1line1, y1line1, x2line1, y2line1;
        double x1line2, y1line2, x2line2, y2line2;

        if ( !line1.intersectsLine( line2 ) )
            return false; // there are parallel
        /* first, check to see if the segments intersect by parameterization
         on s and t; if s and t are both between [0,1], then the
         segments intersect */
        x1line1 = (double)line1.getX1();
        y1line1 = (double)line1.getY1();
        x2line1 = (double)line1.getX2();
        y2line1 = (double)line1.getY2();

        x1line2 = (double)line2.getX1();
        y1line2 = (double)line2.getY1();

```

```

x2line2 = (double)line2.getX2();
y2line2 = (double)line2.getY2();

/* check to see if the segments have any endpoints in common. If they do,
   then return the endpoints as the intersection point */
if ((x1line1==x1line2) && (y1line1==y1line2))
{
    intersectionPoint = (new Point((int)x1line1, (int)y1line1)); return true;
}
if ((x1line1==x2line2) && (y1line1==y2line2))
{
    intersectionPoint = (new Point((int)x1line1, (int)y1line1)); return true;
}
if ((x2line1==x1line2) && (y2line1==y1line2))
{
    intersectionPoint = (new Point((int)x2line1, (int)y2line1)); return true;
}
if ((x2line1==x2line2) && (y2line1==y2line2))
{
    intersectionPoint = (new Point((int)x2line1, (int)y2line1)); return true;
}

dyline1 = -( y2line1 - y1line1 );
dxline1 = x2line1 - x1line1;

dyline2 = -( y2line2 - y1line2 );
dxline2 = x2line2 - x1line2;

e = -(dyline1 * x1line1) - (dxline1 * y1line1);
f = -(dyline2 * x1line2) - (dxline2 * y1line2);

/* compute the intersection point using
   ax+by+e = 0 and cx+dy+f = 0

   If there is more than 1 intersection point between two lines,
   */
if( (dyline1 * dxline2 - dyline2 * dxline1) == 0 )
    return false; // there are the same
intersectionPoint=(new Point(
    (int) (-e * dxline2 - dxline1 * f)/(dyline1 * dxline2 - dyline2 * dxline1)),
    (int) (-dyline1 * f - dyline2 * e)/(dyline1 * dxline2 - dyline2 * dxline1)));
return true;
}

}
}

CoverageAngle
using System;
using System.Collections.Generic;
using System.Text;

namespace mmcc
{
    [Serializable]
    public class CoverageAngle
    {
        public Node node;
        public double startAngle=0;
        public double endAngle=0;
        public double angle = 0;
        public CoverageAngle()
        {
            node = new Node();
            startAngle = 0;
            endAngle = 0;
            angle = 0;
        }
        public CoverageAngle(Node n,double sAngle,double eAngle, double Angle)
        {
            node = n;
            startAngle = sAngle;
            endAngle = eAngle;
            angle = Angle;
        }
    }
}

```

Generate_Node_Config.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using System.Drawing;

namespace mmcc
{
    /// <summary>
    /// Summary description for node.
    /// </summary>
    public class TemporalLocation
    {
        public int x;
        public int y;
        public int t;
        public TemporalLocation(int x, int y, int t)
        {
            this.x = x;
            this.y = y;
            this.t = t;
        }
        public TemporalLocation(){}
    }
    public class Node
    {
        public int id; //id of the node
        public Point position; // position of the node
        public bool isMCCNode = true;
        public bool isNormal = true;
        public bool isListening = true;
        public bool isRedundant = true;

        public int cluster_value; // used for skeletonize, assign value of the skeleton cluster
        public int sensing_radius; //The sensing zone of the node
        public int communication_radius; //The communication zone of the node
        public double energy; // The energy(unit) of the node
        public double energyScaleFactor = 1; // energy scale factor
        public int f_id; //the father node id of the node
        public int back_father;
        public int oDistance;

        public int bn_msg;
        public int no_of_empty_sets;
        public int pairing_msg;
        public int RxCost = 69;
        public int TxCost = 42;
        public int IdealCost = 100;
        public int iSensorDelay = 0;
        public Stopwatch stopWatch = new Stopwatch();
        public System.Timers.Timer timer = new System.Timers.Timer();
        public Tree recoveryTree = null;
        public List<Tree> receivedrecoveryTree = new List<Tree>();

        public double F; // sensor area size
        public double L; //sensor Perimeter length
        public double Rc_Rs_Ratio;
        public double CR=0; //cover ratio
        public double overlappingProbability;
        public bool nieghborDiscovered = false;
        public List<CoverageAngle> coverageAngleList = new List<CoverageAngle>();
        public List<CoverageAngle> minimumCoverageAngleList = new List<CoverageAngle>();
        public double averageDistanceBetweenNodes;
        public List<Node> neighbors = new List<Node>(); //Set of neighbor nodes
        public List<Node> coverageNeighbors = new List<Node>();
        public System.Drawing.Rectangle myRect;
        public List<RoutingRow> routingTable = new List<RoutingRow>();
        public List<WirelessSensorConnection> aConnections = new List<WirelessSensorConnection>();
        public WirelessSensorConnection connectionCurrent = null;
        public List<Packet> aPackets = new List<Packet>();

        // mobility information
        public ContinuousRandomWalkMobilityInfo crwmi = new ContinuousRandomWalkMobilityInfo();
        public TemporalLocation temporalLocation = new TemporalLocation();
    }
}
```

```

public double MobilitySpeed = 0;

//Voronoi Implementation Variables and parameters
// node quarters
public List<Node> Q1 = new List<Node>(), Q2 = new List<Node>(), Q3 = new List<Node>(), Q4 = new List<Node>();
// Local Voronoi Vertcies
public List<Point> IVorVertices = new List<Point>();
public List<GreedyVor.Line2D> IVor = new List<mmcc.GreedyVor.Line2D>();
//

/// Minimum Hop Routing parameters
///paper title " using A Minimum Hop Routing Protocol for Wireless Sensor Network"
///Shoa-Shan Chaiang - Chih-Hung Huang
public int hopCount = -1;
public List<Node> parentList = new List<Node>();
public List<Node> siblingList = new List<Node>();
public List<Node> childList = new List<Node>();
/// <summary>
/// </summary>
public Node MaxCC;
public double CCValue;

private Pen occupationColor = Pens.Black;
public int occupation; // 0: undefined
// 1: Border
// 2: Sub-border (unused)
// 3: Skeleton (main route)
// 4: forward (sensor off)
//5: base node
public bool base_node; //if the node is the BASE

public Node()
{
    //
    // TODO: Add constructor logic here
    //
    cluster_value = 0;
    back_father = 0;
    base_node = false;
    pairing_msg = 0;
    no_of_empty_sets = 0;
    MaxCC = this;
    CCValue = 0;
}

// Function to new node position
public void MoveTo(Node destinationSensor)
{
    double slop = (double)(destinationSensor.position.Y - this.position.Y) / (double)(destinationSensor.position.X -
this.position.X);
    double b = this.position.Y - this.position.X * slop;

    while (((this.position.X < destinationSensor.position.X) || (destinationSensor.position.X < this.position.X)))
    {
        if (this.position.X < destinationSensor.position.X)
            this.position.X ++;
        else
            this.position.X --;

        this.position.Y = (int)Math.Floor(slop * this.position.X + b);
    }
}

public void MoveTo(Point position)
{
    double slop = (double)(position.Y - this.position.Y) / (double)(position.X - this.position.X);
    double b = this.position.Y - this.position.X * slop;

    while (((this.position.X < position.X) || (position.X < this.position.X)))
    {
        if (this.position.X < position.X)
            this.position.X ++;
        else
            this.position.X --;
    }
}

```

```

        this.position.Y = (int)Math.Floor(slop * this.position.X + b);
    }
}

public bool UpdateNL()
{
    for (int i = 0; i < this.neighbors.Count; i++)
    {
        Node node = this.neighbors[i];
        if (!this.is_Communication_Boundary(node.position) || node.energy <= 0)
            this.neighbors.RemoveAt(i);
    }

    for (int i = 0; i < this.coverageNeighbors.Count; i++)
    {
        Node node = this.coverageNeighbors[i];
        if (!this.is_Sensing_Boundary(node.position) || node.energy <= 0)
            this.coverageNeighbors.RemoveAt(i);
    }

    Console.WriteLine("Update My neighbors List--> " + this.id);

    return true;
}

public void UpdateNL(Node sender)
{
    for (int i = 0; i < this.neighbors.Count; i++)
    {
        Node node = this.neighbors[i];
        if (!this.is_Communication_Boundary(node.position) || node.energy <= 0)
            this.neighbors.RemoveAt(i);
    }

    for (int i = 0; i < this.coverageNeighbors.Count; i++)
    {
        Node node = this.coverageNeighbors[i];
        if (!this.is_Sensing_Boundary(node.position) || node.energy <= 0)
            this.coverageNeighbors.RemoveAt(i);
    }

    Console.WriteLine("Update My neighbors List--> " + this.id);

    if (this.is_Communication_Boundary(sender.position) && sender.energy > 0)
        if (!this.neighbors.Contains(sender))
            this.neighbors.Add(sender);

    if (this.is_Sensing_Boundary(sender.position) && sender.energy > 0)
        if (!this.coverageNeighbors.Contains(sender))
            this.coverageNeighbors.Add(sender);
}

public double PeriodCost(int factor)
{
    double cost = factor * (((double)1 / (double)(1+(this.neighbors.Count * this.sensing_radius))));
    return cost;
}

//
//Set value for properties of the node.
//
public void setValue(int ID, int X, int Y, int SENSING_ZONE, int COMMUNICATION_ZONE, int ENERGY, int F_ID, int
OCCUPATION)
{
    id = ID;
    position.X = X;
    position.Y = Y;
    sensing_radius = SENSING_ZONE;
    communication_radius = COMMUNICATION_ZONE;
    energy = ENERGY;
    occupation = OCCUPATION;
    f_id = F_ID;
    oDistance = this.getDistanceFromOrigin();
    F = sensing_radius * sensing_radius * System.Math.PI;
    L = 2 *sensing_radius * System.Math.PI;
    Rc_Rs_Ratio = (double)sensing_radius / (double)communication_radius;

```

```

        myRect = new Rectangle(position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    }
    public void setValue(int ID, int X, int Y, int SENSING_ZONE, int COMMUNICATION_ZONE, double ENERGY, int F_ID, int
OCCUPATION)
    {
        id = ID;
        position.X = X;
        position.Y = Y;
        sensing_radius = SENSING_ZONE;
        communication_radius = COMMUNICATION_ZONE;
        energy = ENERGY;
        occupation = OCCUPATION;
        f_id = F_ID;
        oDistance = this.getDistanceFromOrigin();
        F = sensing_radius * sensing_radius * System.Math.PI;
        L = 2 * sensing_radius * System.Math.PI;
        Rc_Rs_Ratio = (double)sensing_radius / (double)communication_radius;
        myRect = new Rectangle(position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    }

    /// <summary>
    /// Draw the node method.
    /// </summary>

    public System.Drawing.Region getRegion()
    {
        Rectangle myRect = new Rectangle(position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sens-
ing_radius * 2);
        System.Drawing.Drawing2D.GraphicsPath p = new System.Drawing.Drawing2D.GraphicsPath();
        p.AddEllipse(myRect);
        System.Drawing.Region r = new Region(p);
        return r;
    }

    public Pen OccupationColor()
    {
        switch (occupation)
        {
            case 0: occupationColor = Pens.Green; break;
            case 1: occupationColor = Pens.Red; break;
            case 2: occupationColor = Pens.Green; break;
            case 3: occupationColor = Pens.Blue; break;
            case 4: occupationColor = Pens.DarkMagenta; break;
            case 5: occupationColor = Pens.HotPink; break;
            default: occupationColor = Pens.Black; break;
        }
        return this.occupationColor;
    }

    public void drawNode(Graphics g)
    {
        SolidBrush brush0 = new SolidBrush(Color.White);
        SolidBrush brush1 = new SolidBrush(Color.LightSteelBlue);
        SolidBrush brush2 = new SolidBrush(Color.Navy);
        SolidBrush brush3 = new SolidBrush(Color.Green);
        //SolidBrush brush4 = new SolidBrush(Color.WhiteSmoke);
        SolidBrush brush4 = new SolidBrush(Color.Green);
        SolidBrush brush5 = new SolidBrush(Color.Black);
        SolidBrush brush_base = new SolidBrush(Color.DarkBlue);
        Pen pen = new Pen(Color.Gray);
        Pen base_nodePen = new Pen(Color.Green);
        FontStyle style = FontStyle.Regular;
        Font arial = new Font(new FontFamily("Arial"), 12, style);

        if (base_node)
        {
            g.FillEllipse(brush_base, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
            g.DrawEllipse(base_nodePen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius *
2);
            g.DrawString(id.ToString(), arial, brush0, position.X, position.Y);
        }
        else
        {

```

```

if (occupation == 0)//undefined node, color = white
{
    // g.FillEllipse(brush0, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    //g.DrawEllipse(pen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen, position.X - 5, position.Y - 5, 20, 20);
    g.DrawString(id.ToString(), arial, brush5, position.X-5, position.Y-5);
}
else if (occupation == 1)// boder node, color = steel blue
{
    Pen pen2 = new Pen(Color.Red, 2);
    pen2.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
    //g.FillEllipse(brush1, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen2, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);

    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
}
else if (occupation == 2)//sub-border node, color = yellow
{
    g.FillEllipse(brush2, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);

    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
}
else if (occupation == 3)//skeleton or route node, color = red
{
    Pen pen2 = new Pen(Color.Green, 2);
    g.FillEllipse(brush3, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen2, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
}
else if (occupation == 4)//sleeping node, color = lightgrey
{
    g.FillEllipse(brush4, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);

    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
}
else //unknown node, color = white
{
    g.FillEllipse(brush1, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.FillEllipse(brush4, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
    g.DrawEllipse(pen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);

    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
}
}
//
//Draw the center point of Node
//

public void drawNodeCenter(Graphics g)
{
    SolidBrush brush0 = new SolidBrush(Color.White);
    SolidBrush brush1 = new SolidBrush(Color.LightPink);
    SolidBrush brush2 = new SolidBrush(Color.LightYellow);
    SolidBrush brush3 = new SolidBrush(Color.Pink);
    //SolidBrush brush4 = new SolidBrush(Color.WhiteSmoke);
    SolidBrush brush4 = new SolidBrush(Color.Blue);
    SolidBrush brush5 = new SolidBrush(Color.Black);
    SolidBrush brush_base = new SolidBrush(Color.DarkBlue);
    Pen pen = new Pen(Color.LightGray);
    Pen base_nodePen = new Pen(Color.Red,2);

    FontStyle style = FontStyle.Bold;
    Font arial = new Font(new FontFamily("Arial"), 10, style);

    if (base_node)
    {
        base_nodePen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
        base_nodePen.Width = 2;
        g.FillEllipse(brush_base, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        g.DrawEllipse(base_nodePen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);

        g.DrawString(id.ToString(), arial, brush0, position.X-5, position.Y-5);
    }
    else

```

2);

```

{

    if (occupation == 0)//undefined node, color = white
    {
        Pen penC = new Pen(Color.Black, 1);
        Pen pen1 = new Pen(Color.Red, 1);
        //penC.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
        // g.FillEllipse(brush0, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        // g.DrawEllipse(penC, position.X - 5, position.Y - 5, 20, 20 );
        // g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
        g.DrawEllipse(penC, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        //g.DrawEllipse(pen1, position.X - communication_radius, position.Y - communication_radius, communication_radius *
2, communication_radius * 2);
        g.DrawEllipse(penC, position.X - 5, position.Y - 5, 5, 5);
        //g.FillEllipse(brush0, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        // g.DrawEllipse(pen1, position.X - 5, position.Y - 5, 5, 5);
        g.DrawString(id.ToString(), arial, brush5, position.X - 2, position.Y - 2);

    }

    else if (occupation == 1)// border node, color = steel blue
    {
        //Brush B = Brushes.Tomato;
        Pen pen1 = new Pen(Color.Red, 2);
        Pen penC = new Pen(Color.Red, 2);

        //pen1.DashStyle = System.Drawing.Drawing2D.DashStyle.;
        // g.FillEllipse(brush3, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        // g.DrawEllipse(penC, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        g.DrawEllipse(pen1, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        // g.DrawEllipse(penC, position.X - communication_radius, position.Y - communication_radius, communication_radius *
2, communication_radius * 2);
        g.FillEllipse(brush1, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        g.DrawEllipse(pen1, position.X - 5, position.Y - 5, 5, 5);
        g.DrawString(id.ToString(), arial, brush5, position.X - 2, position.Y - 2);

    }

    else if (occupation == 2)//sub-border node, color = yellow
    {
        Pen pen2 = new Pen(Color.Black, 1);
        g.FillEllipse(brush2, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        g.DrawEllipse(pen2, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);

        g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
    }

    else if (occupation == 3)//skeleton or route node, color = red
    {
        Pen pen2 = new Pen(Color.Green, 2);
        g.FillEllipse(brush3, position.X - 2, position.Y - 2, 2 * 2, 2 * 2);
        g.DrawEllipse(pen2, position.X - 2, position.Y - 2, 2 * 2, 2 * 2);
        //g.DrawEllipse(pen2, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        //g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
    }

    else if (occupation == 4)//sleeping node, color = lightgrey
    {
        Pen pen4 = new Pen(Color.Blue, 1);
        g.FillEllipse(brush4, position.X - 2, position.Y - 2, 2 * 2, 2 * 2);
        g.DrawEllipse(pen4, position.X - 2, position.Y - 2, 2 * 2, 2 * 2);
        // g.DrawEllipse(pen4, position.X - communication_radius, position.Y - communication_radius, communication_radius *
2, communication_radius * 2);
        //g.DrawEllipse(pen4, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
    }

    else //unknown node, color = white
    {
        //g.FillEllipse(brush1, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        pen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;

        g.FillEllipse(brush5, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        g.DrawEllipse(pen, position.X - 5, position.Y - 5, 5 * 2, 5 * 2);
        //g.DrawEllipse(pen, position.X - sensing_radius, position.Y - sensing_radius, sensing_radius * 2, sensing_radius * 2);
        g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
    }

}

}

```

```

public void drawPoint(Graphics g)
{

    SolidBrush brush0 = new SolidBrush(Color.White);
    SolidBrush brush1 = new SolidBrush(Color.LightSteelBlue);
    SolidBrush brush2 = new SolidBrush(Color.Yellow);
    SolidBrush brush3 = new SolidBrush(Color.Pink);
    //SolidBrush brush4 = new SolidBrush(Color.WhiteSmoke);
    SolidBrush brush4 = new SolidBrush(Color.Wheat);
    SolidBrush brush5 = new SolidBrush(Color.Black);
    SolidBrush brush_base = new SolidBrush(Color.DarkBlue);
    Pen pen = new Pen(Color.Black);

    FontStyle style = FontStyle.Regular;
    Font arial = new Font(new FontFamily("Arial"), 6, style);

    Bitmap bm = new Bitmap(10, 10);

    if (base_node)
    {
        bm.SetPixel(0, 0, Color.Cyan);
    }
    else
    {
        if (this.occupation == 0)
            bm.SetPixel(0, 0, Color.Black);
        //bm.SetPixel(0,0,Color.Wheat);

        else if (this.occupation == 1)
            bm.SetPixel(0, 0, Color.SteelBlue);

        else if (this.occupation == 2)
            bm.SetPixel(0, 0, Color.Yellow);

        else if (this.occupation == 3)
            bm.SetPixel(0, 0, Color.Pink);

        else if (this.occupation == 4)
            bm.SetPixel(0, 0, Color.LightGray);
        //bm.SetPixel(0,0,Color.Wheat);
        else
            bm.SetPixel(0, 0, Color.White);

        //bm.SetPixel(0,0,Color.Wheat);
    }
    g.DrawString(id.ToString(), arial, brush5, position.X, position.Y);
    g.DrawImageUnscaled(bm, this.position.X, this.position.Y);
}

/// <summary>
/// Consider if a point belongs to the node.
/// </summary>

public bool is_Sensing_Boundary(Point point)
{
    if (Math.Sqrt(Math.Pow((point.X - position.X), 2) + Math.Pow((point.Y - position.Y), 2)) <= this.sensing_radius)
        return true;
    else
        return false;
}

public bool is_Sensing_Boundary(Node point)
{
    if (Math.Sqrt(Math.Pow((point.position.X - position.X), 2) + Math.Pow((point.position.Y - position.Y), 2)) <=
this.sensing_radius)
        return true;
    else
        return false;
}

public bool is_ItInsideSensingArea(Node node)
{
    if (Math.Sqrt(Math.Pow((node.position.X - position.X), 2) + Math.Pow((node.position.Y - position.Y), 2)) <=
node.sensing_radius > this.sensing_radius)
        return false;
    else
        return true;
}

/// <summary>

```

```

    /// Consider if a node can talk to other node.
    /// </summary>
    public bool is_Communication_Boundary(Point point)
    {
        if (Math.Sqrt(Math.Pow((point.X - position.X), 2) + Math.Pow((point.Y - position.Y), 2)) <= communication_radius)
            return true;
        else
            return false;
    }
    ///
    /// Is there sensensing overlapping with the given node
    ///
    public bool is_There_SensingOverlap(Node node)
    {
        if (Math.Sqrt(Math.Pow((node.position.X - position.X), 2) + Math.Pow((node.position.Y - position.Y), 2)) <= (sensing_radius
+ node.sensing_radius))
            return true;
        else
            return false;
    }

    /// <summary>
    /// Show information about the node.
    /// </summary>
    public string toString()
    {
        return ("ID: " + id + "\nPOS: (" + position.X + "," + position.Y + ") \n sRadius: " + sensing_radius + "\n cRadius: " + commu-
nication_radius)+" Energy: "+this.energy+"\n";
    }

    public int getDistanceFromOrigin()
    {
        return Convert.ToInt32(Math.Sqrt(Math.Pow((double)position.X, (double)2) + Math.Pow((double)position.Y, (double)2)));
    }

}

}

using System;
using System.Collections.Generic;
using System.Text;

namespace mmcc
{
    public class Node_Config
    {
        private int node_Communication_Radius = 60;
        private int node_Sensing_Radius = 30;
        private int m_speed;
        private double node_Energy = 2;// 2 joule

        public int nodeCommunicationRadius
        {
            get
            {
                return node_Communication_Radius;
            }
            set
            {
                node_Communication_Radius = value;
            }
        }

        public int speed
        {
            get
            {
                return m_speed;
            }
            set
            {
                this.m_speed = value;
            }
        }
    }
}

```

```

    }

    public int nodeSensingRadius
    {
        get
        {
            return node_Sensing_Radius;
        }
        set
        {
            node_Sensing_Radius = value;
        }
    }

    public double nodeEnergy
    {
        get
        {
            return node_Energy;
        }
        set
        {
            node_Energy = value;
        }
    }
    public Node_Config()
    {
        //
        // TODO: Add constructor logic here
        //
    }
}

}

Packets
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;

namespace mmcc
{
    public class Packet
    {
        public Node scr;
        public Node des;
        public string type; // multihop or direct
        public string action;

        public ArrayList data= new ArrayList();

        public Packet( Node source, Node destination, string action ,string type)
        {
            this.scr = source;
            this.des = destination;
            this.type = type;
            this.action = action;
        }
        public Packet(Node source, Node destination, string action)
        {
            this.scr = source;
            this.des = destination;
            this.action = action;
        }
    }
    public Packet(Node source, string type)
    {
        this.scr = source;

        this.type = type;
    }
}

```

```

        public Packet(Node source, Node destination, ArrayList data)
        { this.data = data; }
    }
}
==

using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace MACA
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new GreedyVor.Greedy());
        }
    }
}
=====

R
/*
 * http://www.routingprotokolle.de/Routing/mobility_random_walk.htm
 * The algorithm can be divided into the following five steps.
 *
 * 1. Select a random speed (uniform distributed)
 * 2. Select a random direction (uniform distributed)
 * 3. Move into that direction
 *     a. for a predefined amount of time
 *     b. for a certain distance
 *     c. if the border of the scenario is reached, select a new direction (Bouncing rule)
 * 4. Wait some time
 * 5. Go back to step one
 *
 * * implementaion is from
 * * http://www.people.cornell.edu/pages/il35/ContinuousRandomWalk.java
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace mmcc
{
    public class Bounds
    {
        public System.Drawing.Rectangle bounds = new Rectangle();
        public int X;
        public int Y;
        public bool inside(Point p)
        {
            if(bounds.Contains(p)) return true;
            return false;
        }
    }

    public Bounds(System.Drawing.Rectangle bounds)
    {
        this.bounds = bounds;
        X = bounds.X;
        Y = bounds.Y;
    }
}

public class RandomWalk
{
    private Bounds bounds ; // field boundaries.

    private double fixedRadius ; // fixed component of step size.

```

```

private long pauseTime ; // time wait between steps.

private double randomRadius; // random component of step size.

private long speed; //pixels per iteration

Random random = new Random();/** random number generator 0-1*/

/// <summary>
/// Create and initialize new random walk object.
/// </summary>
/// <param name="bounds">field boundaries</param>
/// <param name="fixedRadius">fixed component of step size</param>
/// <param name="randomRadius">random component of step size</param>
/// <param name="pauseTime">time wait between steps.</param>
public RandomWalk(Bounds bounds, double fixedRadius, double randomRadius, long pauseTime, long speed)
{
    this.init(bounds, fixedRadius, randomRadius, pauseTime, speed );
}
/// <summary>
/// Initialize random walk object.
/// </summary>
/// <param name="bounds"></param>
/// <param name="fixedRadius"></param>
/// <param name="randomRadius"></param>
/// <param name="pauseTime"></param>
private void init(Bounds bounds, double fixedRadius, double randomRadius, long pauseTime, long speed )
{
    this.bounds = bounds;
    this.fixedRadius = fixedRadius;
    this.randomRadius = randomRadius;
    this.pauseTime = pauseTime;
    this.speed = speed;
}

/// <summary>
/// Schedule next movement. This method will again be called after every movement on the field.
/// </summary>
/// <param name="f"></param>
public void movenext(Node f)
{
    Point loc = f.position;
    f.temporalLocation = new TemporalLocation(loc.X, loc.Y, DateTime.Now.Millisecond);
    Console.WriteLine("Current Position x= " + loc.X + " , y= " +
        loc.Y + " Current time t=" + f.temporalLocation.t);

    // compute new random position with fixedRadius+randomRadius() distance
    double randomAngle = 2 * Math.PI * this.random.NextDouble();
    double r = this.fixedRadius + this.random.NextDouble() * this.randomRadius;

    r = this.speed;

    double x = r * Math.Cos(randomAngle), y = r * Math.Sin(randomAngle);

    //double x = 20;
    // double y = 20;

    //x = x + bounds.X; y = y + bounds.Y;
    double lx = loc.X + x, ly = loc.Y + y;

    // bounds check and reflect

    if (lx < bounds.X)
        lx = lx + bounds.X;

    if (ly < bounds.Y)
        ly = ly + bounds.Y;

    if (lx > bounds.bounds.Width)
        lx = bounds.bounds.Width - (lx - bounds.bounds.Width); //Changed by Noam Mori

    if (ly > bounds.bounds.Height)
        ly = bounds.bounds.Height - (ly - bounds.bounds.Height); //Changed by Noam Mori

    // move

```

```

        if (pauseTime > 0)
        {
            System.Threading.Thread.Sleep((int)pauseTime);

            {
                Console.WriteLine("Node CRWM: " + f.id + " Pos: " + f.position.X + ", " + f.position.Y +
                                "Energy: " + f.energy);

                Point l = new Point((int)lx, (int)ly);

                f.MoveTo(l);

                double mX=Convert.ToDouble(l.X-f.temporalLocation.x);
                double mY=Convert.ToDouble(l.Y-f.temporalLocation.y);

                f.MobilitySpeed=Math.Sqrt( Math.Pow(mX,2)+Math.Pow(mY,2)); //Convert.ToDouble(DateTime.Now.Millisecond-
f.temporalLocation.t);

                Console.WriteLine("after moving Position x= " + f.position.X + " , y= " + f.position.Y + " Current time t=" +
DateTime.Now.Millisecond);
            }
        }
    }

    public class ContinuousRandomWalkMobilityInfo
    {
        public PointF velocityVector;
        public int steps;//number of steps remaining to waypoint
        public long stepTime;//duration of each step.
        public long pauseTime;
        public ContinuousRandomWalkMobilityInfo()
        {

            this.velocityVector = new PointF();
            this.steps = 0;
            this.stepTime = 0;
            this.pauseTime = 0;

        }
        public ContinuousRandomWalkMobilityInfo(PointF velocityVector, int steps, long stepTime, long pauseTime)
        {

            this.velocityVector = velocityVector;
            this.steps = steps;
            this.stepTime = stepTime;
            this.pauseTime = pauseTime;

        }
    }

    class ContinuousRandomWalk
    {
        /** Step granularity [meters].*/
        private float precision;

        /** Movement boundaries. */
        private Bounds bounds;

        /** Pause time at each stop .*/
        private long pauseTime;

        /** Minimum movement speed [meters/second].*/
        private float minspeed;

        /** Maximum movement speed [meters/second].*/
        private float maxspeed;

        /**epochDuration long duration of a straight-line movement interval*/
        private long epochDuration;

        /** random number generator 0-1*/
        Random random = new Random();
    }

```

```

/**
 * Generate new random velocity vector.
 *
 * @return Location2D new velocity vector
 */
private PointF generateNewVelocityVector()
{
    float angle = (float)(2 * Math.PI * random.NextDouble());
    float velocity = (float)(minspeed + random.Next() * (maxspeed - minspeed));
    return new PointF(velocity * (float)Math.Sin(angle), velocity * (float)Math.Cos(angle));
}

/// <summary>
///
/// <Initialize "continuous" Random Walk mobility model
/// <param name="bounds"> bounds Location2D simulation area boundary</param>
/// <param name="pauseTime"> pauseTime long pause time [in JiST time units]</param>
/// <param name="precision"> precision float mobility precision [meters] (a node moves in "jumps" of this size)</param>
/// <param name="minspeed"> minspeed float minimum speed [meters/second]</param>
/// <param name="maxspeed"> maxspeed float maximum speed [meters/second]</param>
/// <param name="epochDuration"> epochDuration long duration of a straight-line movement interval</param>
/// </summary>

public ContinuousRandomWalk(Bounds bounds, long pauseTime,
    float precision, float minspeed, float maxspeed, long epochDuration)
{
    init(bounds, pauseTime, precision, minspeed, maxspeed, epochDuration);
}

/**
 * Initialize "continuous" Random Walk mobility model
 *
 * @param bounds Location2D simulation area boundary
 * @param pauseTime long pause time
 * @param precision float mobility precision [meters] (a node moves in "jumps" of this size)
 * @param minspeed float minimum speed [meters/second]
 * @param maxspeed float maximum speed [meters/second]
 * @param epochDuration long duration of a straight-line movement interval
 */
private void init(Bounds bounds, long pauseTime,
    float precision, float minspeed, float maxspeed, long epochDuration)
{
    this.precision = precision;
    this.bounds = bounds;
    this.minspeed = minspeed;
    this.maxspeed = maxspeed;
    this.pauseTime = pauseTime;
    this.epochDuration = epochDuration;
}

/**
 * Find next location. If next location is outside simulation area, it is reflected from the boundary
 * and appropriate velocity component is adjusted accordingly.
 *
 * @param curLoc Location2D current node location
 * @param crwmi ContinuousRandomWalkMobilityInfo node's mobility parameters
 * @return Location2D new location
 */
private Point findNextLoc(Point curLoc, ContinuousRandomWalkMobilityInfo crwmi)
{
    // find next "destination" location, without considering boundaries...
    float velocityX = crwmi.velocityVector.X;
    float velocityY = crwmi.velocityVector.Y;
    float newLocX = curLoc.X + velocityX * crwmi.stepTime; // Constants.SECOND;
    float newLocY = curLoc.Y + velocityY * crwmi.stepTime; // Constants.SECOND;
    // check if the location is within boundaries of the simulation field,
    // if not, bounce the node location and reflect the velocity vector
    Point newLocation = new Point((int)newLocX, (int)newLocY);
    bool reflected = false;
    //if(!bounds.inside(newLocation))
    {
        if (newLocX > bounds.X )
        {
            newLocX = 2 * bounds.X - newLocX;
            velocityX = (-1) * velocityX;

```

```

        reflected = true;
    }
    if (newLocX < 0.0f)
    {
        newLocX = (-1) * newLocX;
        velocityX = (-1) * velocityX;
        reflected = true;
    }
    if (newLocY > bounds.Y)
    {
        newLocY = 2 * bounds.Y - newLocY;
        velocityY = (-1) * velocityY;
        reflected = true;
    }
    if (newLocY < 0.0f)
    {
        newLocY = (-1) * newLocY;
        velocityY = (-1) * velocityY;
        reflected = true;
    }
    }
    if (reflected)
    {
        //update velocity vector
        crwmi.velocityVector = new PointF(velocityX, velocityY);
        Console.WriteLine("velocity vector updated");
    }
    return new Point((int)newLocX, (int) newLocY);
}

/**
 * Schedule next movement.
 */
@param f FieldInterface field interface
@param id Integer id of the node
@param loc Location current location
@param info MobilityInfo node's mobility info (parameters)
*/
public void next(Node f)
{
    Point loc = f.position;
    Console.WriteLine("Node CRWM: " + f.id);
    try
    {

        if (f.crwmi.steps == 0) //if already reached destination point, pause and produce a new velocity vector
        {
            // reached turning point

            System.Threading.Thread.Sleep((int)pauseTime);
            f.crwmi.velocityVector = generateNewVelocityVector();
            float distX = f.crwmi.velocityVector.X * epochDuration / 100000; // Constants.SECOND;
            float distY = f.crwmi.velocityVector.Y * epochDuration / 100000; // Constants.SECOND;
            float dist = (float)Math.Sqrt(distX * distX + distY * distY);
            f.crwmi.steps = (int)Math.Max(Math.Floor(dist / precision), 1);
            f.crwmi.stepTime = (long)(epochDuration / f.crwmi.steps);
        }
        // make a step
        System.Threading.Thread.Sleep((int)f.crwmi.stepTime);
        f.crwmi.steps--;
        Point nextLoc = findNextLoc((Point)loc, f.crwmi);
        f.MoveTo(nextLoc);

        Console.WriteLine("Node CRWM Pos: " + f.position.X + "," + f.position.Y);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}

public void move(Node f)
{
    Point loc = f.position;
    Console.WriteLine("Node CRWM: " + f.id + " Pos: " + f.position.X + "," + f.position.Y);
    try

```

```

{
    float distX=0, distY=0;

    // reached turning point
    if (f.crwmi.pauseTime == 0)
    {
        //System.Threading.Thread.Sleep((int)pauseTime);
        f.crwmi.velocityVector = generateNewVelocityVector();
        distX = f.crwmi.velocityVector.X; // *epochDuration; // Constants.SECOND;
        distY = f.crwmi.velocityVector.Y; // *epochDuration; // Constants.SECOND;
        float dist = (float)Math.Sqrt(distX * distX + distY * distY); // circle radius r
        f.crwmi.steps = (int)Math.Max(Math.Floor(dist / precision), 1);
        f.crwmi.stepTime = (long)(epochDuration / f.crwmi.steps);
        f.crwmi.pauseTime = this.pauseTime;
    }
    f.crwmi.pauseTime--;

    // make a step
    //System.Threading.Thread.Sleep((int)f.crwmi.stepTime);
    if (f.crwmi.stepTime == 0)
    {
        f.crwmi.steps--;
        Point nextLoc = findNextLoc((Point)loc, f.crwmi);

        // move
        f.MoveTo(nextLoc);

        Console.WriteLine("Node CRWM Pos: " + f.position.X + "," + f.position.Y);
    }
    f.crwmi.stepTime--;
}
catch (Exception e)
{
    Console.WriteLine(e);
}
}

public void movenext(Node f)
{
    Point loc = f.position;

    // compute new random position with fixedRadius+randomRadius() distance
    double randomAngle = 2 * Math.PI * this.random.NextDouble();
    double r = f.sensing_radius + this.random.NextDouble() * 1.2;
    double x = r * Math.Cos(randomAngle), y = r * Math.Sin(randomAngle);
    Console.WriteLine("x= " + x + ", y= " + y + "bound x=" + bounds.X + "bound y" + bounds.Y);
    //x = x + bounds.X; y = y + bounds.Y;
    double lx = loc.X + x, ly = loc.Y + y;

    // bounds check and reflect
    //if (lx < bounds.X) lx = loc.X;
    //if (ly < bounds.Y) ly = loc.Y;
    if (lx < bounds.X) lx = lx + bounds.X;
    if (ly < bounds.Y) ly = ly + bounds.Y;

    if (lx > bounds.bounds.Width) lx = bounds.bounds.Width - (lx - bounds.bounds.Width); //Changed by Noam Mori
    if (ly > bounds.bounds.Height) ly = bounds.bounds.Height - (ly - bounds.bounds.Height); //Changed by Noam Mori
    // move
    if (pauseTime > 0)
    {
        System.Threading.Thread.Sleep((int)pauseTime);
        //System.Threading.Thread.Sleep(20);
        for (int i = 0; i < pauseTime; i++) { };
    }
    {
        Console.WriteLine("Node CRWM: " + f.id + " Pos: " + f.position.X + "," + f.position.Y);
        Point l = new Point((int)lx, (int)ly);
        //System.out.println("move at t="+JistAPI.getTime()+" to="+l);
        f.MoveTo(l);
    }
}
}
}
}

```

```

    }

using System;
using System.Collections.Generic;
using System.Text;

namespace mmcc
{
    public class RoutingRow
    {
        public int hopCount;
        public Node node;
        public double energyLevel;
        public RoutingRow(int hopCount, Node node, double energyLevel)
        {
            this.energyLevel = energyLevel;
            this.hopCount = hopCount;
            this.node = node;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace mmcc
{
    public class Stopwatch
    {
        private DateTime startTime;
        private DateTime stopTime;
        public bool running = false;

        public void Start()
        {
            this.startTime = DateTime.Now;
            this.running = true;
        }

        public void Stop()
        {
            this.stopTime = DateTime.Now;
            this.running = false;
        }

        // elapsed time in milliseconds
        public double GetElapsedTime()
        {
            TimeSpan interval;

            if (running)
                interval = DateTime.Now - startTime;
            else
                interval = stopTime - startTime;

            return interval.TotalMilliseconds;
        }

        // elapsed time in seconds
        public double GetElapsedTimeSecs()
        {
            TimeSpan interval;

            if (running)
                interval = DateTime.Now - startTime;
            else
                interval = stopTime - startTime;

            return interval.TotalSeconds;
        }

        public void countDownTimer(double period)

```

```

    {
        this.startTime = DateTime.Now;
        this.stopTime = startTime;
        this.stopTime = stopTime.AddSeconds(period);
        this.running = true;
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using System.Threading;

namespace mmcc
{
    public class TargetList
    {
        // This class represents (and manages) a list of targets.

        #region Variables and initialization code

        public ArrayList aTargets;           // the array of targets
        public Mutex mutexTarget;           // a mutex to control access to the target list
        private int iMaxX, iMaxY;           // the maximum X and Y coordinates of a target
        private Random r;                   // a random number
        generator (the same one used by the WirelessSensorNetwork object that houses this object)

        public TargetList(int iMaxX, int iMaxY, Random r)
        {
            this.iMaxX = iMaxX;
            this.iMaxY = iMaxY;
            aTargets = new ArrayList();
            mutexTarget = new Mutex();
            this.r = r;
        }

        #endregion

        #region Target list management functions

        public void AddTarget()
        {
            // this function creates a new target and adds it to the target list
            int iBorder = r.Next(4);
            if (iBorder == 0) // add to left border, traveling right and up or down
                aTargets.Add(new Target(-5, r.Next(iMaxY), r.Next(3) + 1, r.Next(7) - 3));
            else if (iBorder == 1) // add to top border, traveling down and left or right
                aTargets.Add(new Target(r.Next(iMaxX), -5, r.Next(7) - 3, r.Next(3) + 1));
            else if (iBorder == 2) // add to right border, traveling left and up or down
                aTargets.Add(new Target(iMaxX + 5, r.Next(iMaxY), r.Next(3) - 3, r.Next(7) - 3));
            else if (iBorder == 3) // add to bottom border, traveling up and left or right
                aTargets.Add(new Target(r.Next(iMaxX), iMaxY + 5, r.Next(7) - 3, r.Next(3) - 3));
        }

        public void Update()
        {
            // this function moves the targets around, and removes those that have traveled out of bounds
            ArrayList aRemoveTargets = new ArrayList();
            foreach (Target target in aTargets)
            {
                target.x += target.dx;
                target.y += target.dy;
                if ((target.x < -5) || (target.y < -5) || (target.x > iMaxX + 5) || (target.y > iMaxY + 5))
                    aRemoveTargets.Add(target);
            }
            foreach (Target target in aRemoveTargets)
                aTargets.Remove(target);
        }

        #endregion
    }

    public class Target

```

```

{
    // A lightweight class - really just a structure - for a target.

    #region Variables and initialization code

    public int x, y, dx, dy;
    public Target(int x, int y, int dx, int dy)
    {
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
    }

    #endregion
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace mmcc
{
    public class Tree
    {
        public List<Leaf> tree = new List<Leaf>();

        public int getIndex(Node node)
        {
            foreach (Leaf leaf in this.tree)
            {
                if (node.id == leaf.node.id) return tree.IndexOf(leaf);
            }
            return -1;
        }

        public Leaf getRedundant()
        {
            foreach (Leaf L in this.tree)
            {
                if (L.node.isRedundant)
                    return L;
            }
            return null;
        }

        public void buildTree(Node sensor, List<Node> sensors)
        {
            List<Leaf> rBranch = new List<Leaf>();
            List<Leaf> MccBranch = new List<Leaf>();
            foreach (Node A in sensors)
            {
                if (A.isRedundant)
                {
                }
                else
                {
                    if (A.isMCCNode)
                    {
                        MccBranch.Add(new Leaf(A, sensor, 1));
                    }
                }
            }
            LeafComparerByDistance compare = new LeafComparerByDistance();
            rBranch.Sort(compare);
            MccBranch.Sort(compare);

            foreach (Leaf leaf in rBranch)
                tree.Add(leaf);
            foreach (Leaf leaf in MccBranch)
                tree.Add(leaf);
        }
    }
}

```

```

    }

    public void merge(Node root, Tree recoveryTree)
    {

        for (int i = 0; i < this.tree.Count; i++)
        {
            Leaf leaf = this.tree[i];
            if (leaf.level == 1)
            {

                List<Leaf> children = new List<Leaf>();

                //int index = getIndex(leaf.parentNode); //get parentnode index from tree; -1 if not founded
                //int childIndex = getIndex(leaf.node); // get child node index as leaf; -1 if not founded
                //if (index != -1 && childIndex == -1) // exist parent node as leaf and not exist child node

                children = recoveryTree.getChildren(leaf.node);

                if (children != null)
                {
                    removeChildren(leaf.node);
                    foreach (Leaf c in children)
                    {
                        if (c.node.id != root.id)
                        {
                            int cIndex = getIndex(c.node);
                            if ((cIndex == -1 ? true : leaf.level != tree[cIndex].level))
                            {

                                if (!isContains(c))
                                {
                                    if (leaf.level < 2)
                                    {
                                        this.tree.Add(new Leaf(c.node, leaf.node, (leaf.level + 1)));
                                        leaf.hasChildren = true;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        //if (index != -1 && (childIndex == -1 ? true : tree[childIndex].level != tree[index].level))
        //{
        //    if (tree[index].level < 2)
        //    {
        //        // new child with existence parent
        //        this.tree.Add(new Leaf(leaf.node, tree[index].node, (tree[index].level + 1)));
        //        tree[index].hasChildren = true;
        //    }
        //}

    }
}

//public void merge(Node root, Tree recoveryTree)
//{

//    foreach (Leaf leaf in recoveryTree.tree)
//    {

//        if ((leaf.node.id != root.id))
//        {
//            if (!isContains(leaf))
//            {
//                int index = getIndex(leaf.parentNode); //get parentnode index from tree; -1 if not founded
//                int childIndex = getIndex(leaf.node); // get child node index as leaf; -1 if not founded
//                //if (index != -1 && childIndex == -1) // exist parent node as leaf and not exist child node
//                if (index != -1 && (childIndex == -1 ? true : tree[childIndex].level != tree[index].level))
//                {
//                    // new child with existence parent

```

```

//          this.tree.Add(new Leaf(leaf.node, tree[index].node, (tree[index].level + 1)));
//          tree[index].hasChildren = true;
//      }
//  }

//  }
//  }

//}
/// <summary>
/// Check leaf existence in tree
/// </summary>
/// <param name="leaf"></param>
/// <returns></returns>
bool isContains(Leaf leaf) {
    foreach (Leaf inleaf in this.tree) {
        if ((inleaf.node.id == leaf.node.id) && (inleaf.parentNode.id == leaf.parentNode.id)) return true;
    }
    return false;
}

public void removeChildren(Node parent)
{
    for (int i = 0; i < this.tree.Count; i++)
    {
        Leaf leaf = tree[i];
        if (leaf.parentNode.id == parent.id && leaf.level > 1)
            this.tree.Remove(leaf);
    }
}

public List<Leaf> getChildren(Node parent)
{
    List<Leaf> children=new List<Leaf>();
    foreach (Leaf leaf in tree)
        if (leaf.parentNode.id == parent.id)
            children.Add(leaf);
    if (children.Count > 0)
        return children;
    return null;
}

}
public class LeafComparerByDistance : IComparer<Leaf>
{
    public int Compare(Leaf A, Leaf B)
    {
        return A.distance.CompareTo(B.distance);
    }
}

public class Leaf
{
    public Node parentNode = null;
    public Node node;
    public double distance;// distance to parent
    public bool hasChildren=false;
    public int level;

    public Leaf(Node child,Node parentNode,int level)
    {
        this.parentNode = parentNode; this.node = child; this.level = level;
        this.distance = Math.Sqrt(Math.Pow((this.parentNode.position.X - this.node.position.X), 2) +
Math.Pow((this.parentNode.position.Y - this.node.position.Y), 2));
    }
}

}

using System;
using System.Collections.Generic;

```

```

using System.Text;

namespace mmcc
{
    public class WirelessSensorConnection {
        // This class represents a communications link between two wireless sensors.

        #region Variables and initialization code

        public Node sSender; // the upstream sensor
        public Node sReceiver; // the downstream sensor - every node in the data collector/uplink zone will have a connection with a NULL sReceiver.
        public Packet packet = null;
        // the packet currently being transmitted on this connection (only one at a time, of course)
        public double dTransmitCost, dReceiveCost; // the energy costs of transmitting and receiving the packet
        public int iTransmitterDelay; // the total time this node would normally wait to complete delivery of a packet
        public int iTransmitting; // the timer for completing delivery of a packet
        public WirelessSensorConnection(Node sSender, Node sReceiver, double dTransmitCost, double dReceiveCost, int iTransmitterDelay) {
            this.sSender = sSender;
            this.sReceiver = sReceiver;
            this.dTransmitCost = dTransmitCost;
            this.dReceiveCost = dReceiveCost;

            this.iTransmitting = 0;
            this.iTransmitterDelay = iTransmitterDelay;
        }

        public WirelessSensorConnection(Node sSender, Node sReceiver, int iTransmitterDelay)
        {
            this.sSender = sSender;
            this.sReceiver = sReceiver;
            this.iTransmitting = 0;
            this.iTransmitterDelay = iTransmitterDelay;
        }
    }

    #endregion

    #region Connection simulation functions

    public void BeginTransmission()
    {
        packet = sSender.aPackets[0];
        //dReceiveCost = calcCost_Rxpower();
        //dTransmitCost = calcCost_Txpower(packet);
        // this function begins transmission of a data packet between these nodes.
        if ((sSender.aPackets.Count > 0) && (iTransmitting == 0))
        {
            if (sSender.energy <= dTransmitCost) // transmission failed - sender has run out of energy
                sSender.energy = 0;
            else if ((sReceiver != null) && (sReceiver.energy <= dReceiveCost)) // transmission failed - receiver has run out of energy
                sReceiver.energy = 0;
            else
            { // success - accept the packet and start transmitting it.
                Console.WriteLine("Successfully Transmitted from: " + sSender.id + " --> " + sReceiver.id);
                iTransmitting = iTransmitterDelay;
                sSender.energy -= dTransmitCost;
                packet = sSender.aPackets[0];
                sSender.aPackets.RemoveAt(0);
                if (sReceiver != null)
                    sReceiver.energy -= dReceiveCost;
            }
        }
    }

    public Packet Transmit()
    {
        // this function continues transmission of a previously accepted packet, and completes transmission if appropriate.
        if ((sSender.energy <= 0) || ((sReceiver != null) && (sReceiver.energy <= 0))) // failed due to depleted energy
            iTransmitting = 0;
        else if (iTransmitting > 0)
        { // transmission in progress
            iTransmitting--;

            if (iTransmitting == 0)
            {

```

```

        if (sReceiver != null)
        {
            sReceiver.aPackets.Add(packet);
            Console.WriteLine("Successfully Received by: " + sReceiver.id + " from " + sSender.id);
            return packet;
        }
        else
        {
            Packet returnPacket = packet;
            packet = null;
            return returnPacket;
        }
    }
    return null;
}

}

#endregion

}
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using System.Drawing;
using System.Threading;

namespace mmcc
{
    public class Zone
    {
        public bool support_N_N = false; //flag to check if user need node to node communication
        public bool base_node_flag = false; //flag to check if the base node exists
        public Node base_node = new Node();
        public Node_Config node_config = new Node_Config();
        public Zone_Config zone_config = new Zone_Config();
        public Queue<Packet> packets = new Queue<Packet>();
        public Thread tSimulation = null;
        public bool bAbort = false;
        // sepcifies whether or not the simulation thread should be aborted
        public int ClusterRaduis=30;
        public DateTime timeStart;
        public int round = 1;
        public Random random;

        public double EnergyThreshold=0.05; // a random number producer
        public int periodFactor=10000;
        public double dMaxEnergy = 2;
        public int iSensorDelay=50;
        public int available_ID = 0;
        public Node sink = null;
        public bool bMultiHopRouting = false;
        public bool bAutoNeighborDiscovery = true;
        public int iLastUpdated = 0;
        public int iTransmitterDelay = 10;
        // the duration of the packet transmission
        // a timer for updating routing information
        public bool bRunningSimulation = false;
        public bool bPaint = false, bPainting = false; // specifies whether or not the map should be painted,
        and whether or not it is currently being painted
        public int iPacketsDelivered = 0;
        //public ImagePoint[] image; //image = array of image points
        public List<Node> network = null; //set of zone nodes
        public double energyScaleFactor = 1; //(double)node_sensing_radius / (double)20; used when we need heterogeous Energy
        // Targets
        public TargetList targets;
        //public double iReceiveCost = 0.1;
        // the energy cost of receiving a packet of information
        public double iSensorCost = 0.1;
        // the energy cost of a tripped sensor

        public Zone()
        {
            // this.network = new List<Node>();

            //

```

```

// TODO: Add constructor logic here
//

}

public void setSink(ref Node sink)
{
    this.sink= sink;
}
public double optimalRadius()
{
    double areaXcord = Convert.ToDouble(zone_config.maxX);
    double areaYcord = Convert.ToDouble(zone_config.maxY);
    double n = Convert.ToDouble(zone_config.numOfNodes);
    double A = (double)(areaXcord * areaYcord);
    double ln = Math.Log(n, Math.Exp(1)) * -1;
    double value = (ln * A) / (Math.PI * (1 - n));
    double R = Math.Sqrt(value);
    if (R <= 0) R = 1;
    return R;
}
/// <summary>
/// Calculate the id for new node to make sure that evrey node have a unique id.
/// </summary>
public int getStartNodeId()
{
    int max = 0;
    foreach (Node n in network)
        max = Math.Max(max, n.id);
    return max + 1;
}
public Point get_Zone_Max_Point()
{
    Point p = new Point(0, 0);

    foreach (Node n in network)
    {
        if (n.position.X + n.sensing_radius > p.X)
            p.X = n.position.X + n.sensing_radius;
        if (n.position.Y + n.sensing_radius > p.Y)
            p.Y = n.position.Y + n.sensing_radius;
    }
    p.X += 1; //set margin =1
    p.Y += 1; //set margin =1
    return p;
}
//Pre-load all nonoverlapping nodes for optimize node distribution
//
//Get all nodes in the communication range of n in node_list
//
public void load_neighbor_nodes(Node n, List<Node> neighbor_nodes, List<Node> node_list)
{
    neighbor_nodes.Clear();
    foreach (Node nd in node_list)
        if (n.is_Communication_Boundary(nd.position) && nd.id != n.id)
        {
            neighbor_nodes.Add(nd);
        }
}
//
//Get all nodes in the sensing range of n in node_list
//
public void load_coverage_neighbor_nodes(Node n, List<Node> neighbor_nodes, List<Node> node_list)
{
    neighbor_nodes.Clear();
    foreach (Node nd in node_list)
        if (n.is_Sensing_Boundary(nd.position) && nd.id != n.id)
            neighbor_nodes.Add(nd);
}
private double distance(Point p1, Point p2)
{
    return Math.Sqrt(Math.Pow((p1.X - p2.X), 2) + Math.Pow((p1.Y - p2.Y), 2));
}
//
//Return the node contains the point and having the distance from the point to the node center is smallest

```

```

//
private Node closest_node(Point p)
{
    Node node = new Node();
    double smallest = 0;
    foreach (Node n in network)
    {
        if (smallest != 0)
        {
            if (smallest > distance(p, n.position))
            {
                smallest = distance(p, n.position);
                node = n;
            }
        }
        else
        {
            smallest = distance(p, n.position);
            node = n;
        }
    }
    return node;
}
//
//Draw arrows for all nodes in are_Nodes from itself to its father
//
public void drawPath(Graphics g, ArrayList processing_list)
{
    foreach (Node an_source in processing_list)
    {
        if (an_source.f_id != 0)
        {
            foreach (Node an_father in processing_list)
            {
                if (an_father.id == an_source.f_id)
                {
                    drawLink(g, an_source, an_father);
                    break;
                }
            }
        }
    }
}
//
//Draw an arrow connecting source point and destination point
//
public void drawLink(Graphics g, Node source, Node destination)
{
    Pen pen = new Pen(Color.Green);
    Pen skeleton_pen = new Pen(Color.Red);
    float x, y;
    y = destination.position.Y;
    x = destination.position.X;

    g.DrawLine(pen, source.position.X, source.position.Y, destination.position.X, destination.position.Y);

    //-----draw arrow-----//
    double slopy, cosy, siny;
    slopy = Math.Atan2((source.position.Y - destination.position.Y), (source.position.X - destination.position.X));
    cosy = Math.Cos(slopy);
    siny = Math.Sin(slopy);

    //---length of arrow is 6

    g.DrawLine(pen, x, y, x + (float)(6 * cosy - (6 / 2 * siny)), y + (float)(6 * siny + (6 / 2 * cosy)));
    g.DrawLine(pen, x, y, x + (float)(6 * cosy + (6 / 2 * siny)), y - (float)(6 / 2 * cosy - 6 * siny));
}

public void drawNeighborsLink(Graphics g, Node source, Node destination)
{
    if (source.is_Communication_Boundary(destination.position))
    {
        Pen pen = new Pen(Color.Green, 1);
        pen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
        Pen skeleton_pen = new Pen(Color.Green);
        float x, y;
    }
}

```

```

        y = destination.position.Y;
        x = destination.position.X;

        g.DrawLine(pen, source.position.X, source.position.Y, destination.position.X, destination.position.Y);

        //-----draw arrow-----//
        double slopy, cosy, siny;
        slopy = Math.Atan2((source.position.Y - destination.position.Y), (source.position.X - destination.position.X));
        cosy = Math.Cos(slopy);
        siny = Math.Sin(slopy);

        //---length of arrow is 6

        g.DrawLine(skeleton_pen, x, y, x + (float)(6 * cosy - (6 / 2 * siny)), y + (float)(6 * siny + (6 / 2 * cosy)));
        g.DrawLine(skeleton_pen, x, y, x + (float)(6 * cosy + (6 / 2 * siny)), y - (float)(6 / 2 * cosy - 6 * siny));
    }
}

public void drawNeighborsLink(Graphics g, Node source, Node destination, Color color)
{
    Pen pen = new Pen(color);
    Pen skeleton_pen = new Pen(Color.Red);
    float x, y;
    y = destination.position.Y;
    x = destination.position.X;

    g.DrawLine(pen, source.position.X, source.position.Y, destination.position.X, destination.position.Y);

    //-----draw arrow-----//
    double slopy, cosy, siny;
    slopy = Math.Atan2((source.position.Y - destination.position.Y), (source.position.X - destination.position.X));
    cosy = Math.Cos(slopy);
    siny = Math.Sin(slopy);

    //---length of arrow is 6

    g.DrawLine(pen, x, y, x + (float)(6 * cosy - (6 / 2 * siny)), y + (float)(6 * siny + (6 / 2 * cosy)));
    g.DrawLine(pen, x, y, x + (float)(6 * cosy + (6 / 2 * siny)), y - (float)(6 / 2 * cosy - 6 * siny));
}

public void drawNeighborsPerimeterLink(Graphics g, Node source, Node destination)
{
    Pen pen = new Pen(Color.Green);
    Pen skeleton_pen = new Pen(Color.Red);
    float x, y;
    y = destination.position.Y;
    x = destination.position.X;

    g.DrawLine(skeleton_pen, source.position.X, source.position.Y, destination.position.X, destination.position.Y);

    //-----draw arrow-----//
    double slopy, cosy, siny;
    slopy = Math.Atan2((source.position.Y - destination.position.Y), (source.position.X - destination.position.X));
    cosy = Math.Cos(slopy);
    siny = Math.Sin(slopy);

    //---length of arrow is 6

    g.DrawLine(skeleton_pen, x, y, x + (float)(6 * cosy - (6 / 2 * siny)), y + (float)(6 * siny + (6 / 2 * cosy)));
    g.DrawLine(skeleton_pen, x, y, x + (float)(6 * cosy + (6 / 2 * siny)), y - (float)(6 / 2 * cosy - 6 * siny));
}

/// <summary>
/// get list nodes positions from given list of nodes
/// </summary>
/// <param name="node_list"></param>
/// <returns></returns>
public ArrayList getPositionPointsFromNodeList(ArrayList node_list)
{
    ArrayList pointsPos = new ArrayList();
    foreach (Node node in node_list)
    {
        pointsPos.Add(node.position);
    }
    return pointsPos;
}

```

```

    }

    public string statistic()
    {
        return "Total " + network.Count.ToString() + " sensor nodes in area (" + zone_config.maxX.ToString() + "," +
zone_config.maxY.ToString() + "). " + network.Count.ToString() + " nodes are deployed";
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace mmcc
{
    public class Zone_Config
    {
        private int max_X = 100;
        private int max_Y = 100;
        private int num_of_nodes = 50;

        public int maxX
        {
            get
            {
                return max_X;
            }
            set
            {
                max_X = value;
            }
        }

        public int maxY
        {
            get
            {
                return max_Y;
            }
            set
            {
                max_Y = value;
            }
        }

        public int numOfNodes
        {
            get
            {
                return num_of_nodes;
            }
            set
            {
                num_of_nodes = value;
            }
        }

        public double optimalRadius()
        {
            double areaXcord = Convert.ToDouble(maxX);
            double areaYcord = Convert.ToDouble(maxY);
            double n = Convert.ToDouble(numOfNodes);
            double A = (double)(areaXcord * areaYcord);
            double ln = Math.Log(n, Math.Exp(1)) * -1;
            double value = (ln * A) / (Math.PI * (1 - n));
            double R = Math.Sqrt(value);
            if (R <= 0) R = 1;
            return R;
        }

        public int ExpectedNumbersOfNeighbors(int communicationRaduis)
        {
            double areaXcord = Convert.ToDouble(maxX);
            double areaYcord = Convert.ToDouble(maxY);
            double n = Convert.ToDouble(numOfNodes);

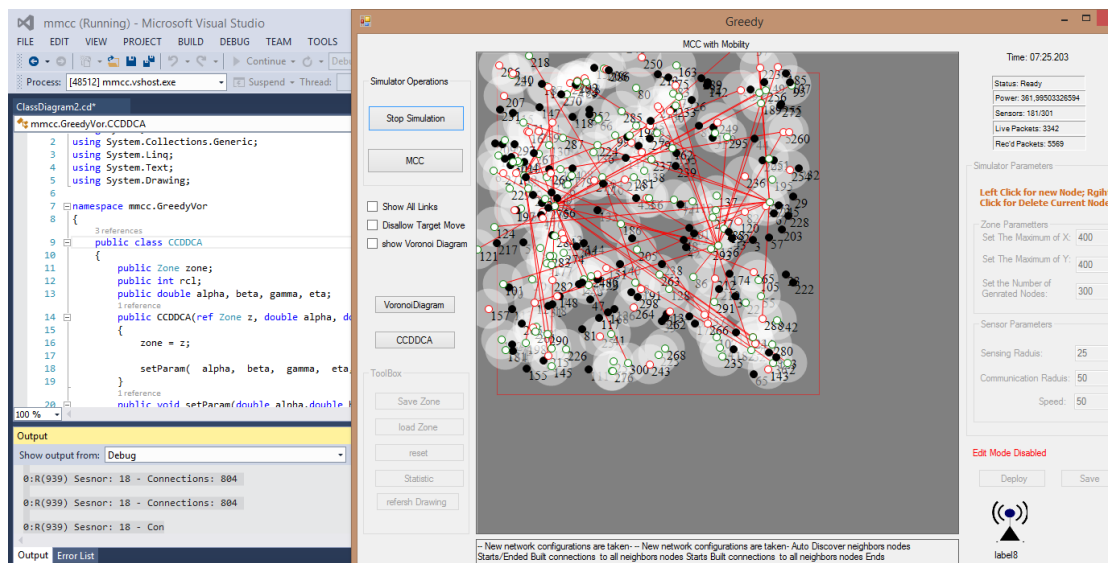
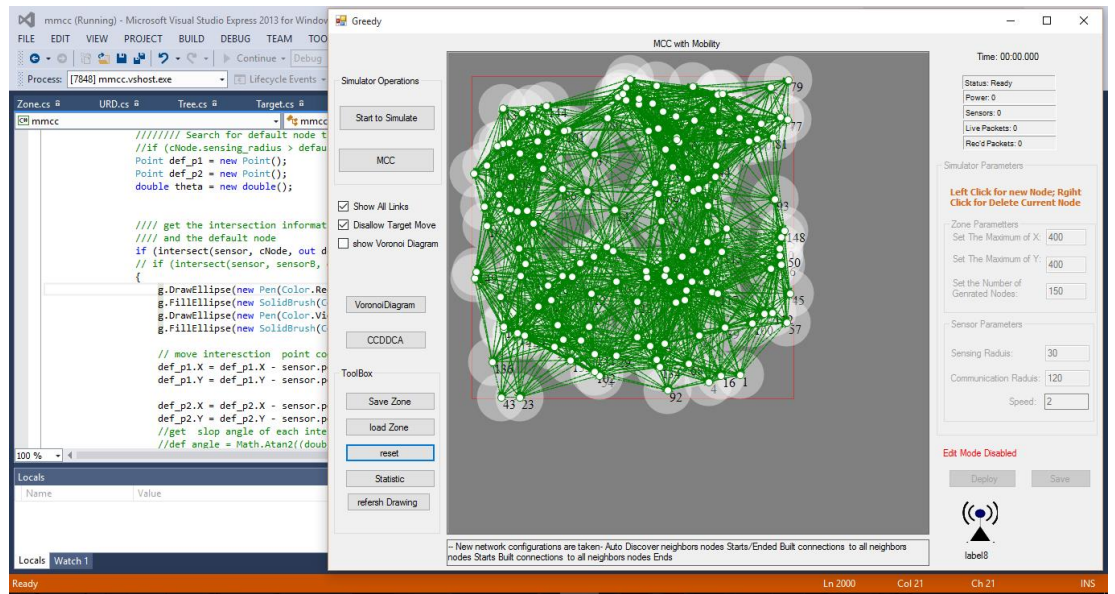
```

```

double A = (double)(areaXcord * areaYcord);
int E = Convert.ToInt32((numOfNodes - 1) * (Math.PI * communicationRaduis * communicationRaduis) / A);
return E;
}
public double getNetworkDensity()
{
    double areaXcord = Convert.ToDouble(maxX);
    double areaYcord = Convert.ToDouble(maxY);
    double n = Convert.ToDouble(numOfNodes);
    double A = (double)(areaXcord * areaYcord);
    return n / A;
}

public Zone_Config()
{
    //
    // TODO: Add constructor logic here

```



ПРИЛОЖЕНИЕ Б. *FT-TEEN*

```

clc;
clear all;
close all;

%Field Dimensions - x and y maximum (in meters)
xm=100;
ym=100;

%x and y Coordinates of the Sink
BS.x=0.5*xm;
BS.y=0.5*ym;
p1 = 0.20;

n=100; %Number of Nodes in the field

% Sn1 = rand(n,2) * xm; %Sensor Nodes positions generated randomly, (array size of n*2) %
% writeToFile(Sn1,'myprotocoll.txt'); %Code That Writes The Sensor Node Positions To
The File.
Sn1 = read From File('myprotocoll.txt');
%Energy Model (all values in Joules)
%Initial Energy given to each node
Eo =0.5;

%Data Aggregation Energy
EDA=5*0.000000001;%5 nJoul
% Transmit Amplifier types
Eelec=70*0.000000001; %70 nJoul
% Eamp=120*0.000000000001; % 120 pJoul
Efs=10*0.000000000001;
Eamp=0.0013*0.000000000001;

do=sqrt(Efs/Eamp);
%maximum number of rounds
rmax=200;
%Optimal Election Probability of a node
%to become cluster head
% p=0.05*n/100;
index11=0;

S(n+1).xd=BS.x;
S(n+1).yd=BS.y;
k=4096; % no of bits in a frame

%Creation of the random Sensor Network.....
figure(1);
xlabel('x dimention');
ylabel('y dimention');
for i=1:1:n
% S(i).xd=rand(1,1)*xm;
% S(i).yd=rand(1,1)*ym;
S(i).xd=Sn1(i,1);
S(i).yd=Sn1(i,2);
S(i).G=0;
S(i).type='N';
S(i).distoBS=abs(sqrt( (S(i).xd-S(n+1).xd)^2 + (S(i).yd-S(n+1).yd)^2 ));

S(i).E=Eo;
S(i).ENERGY=0;
plot(S(i).xd,S(i).yd,'g o');

hold on;
% S(i).distoBS
end

cluster=1;
flag_first_dead1=0;
flag_teenth_dead1=0;
flag_all_dead1=0;
flag_half_dead1=0;

dead=0;
first_dead1=0;
teenth_dead1=0;

```

```

half_deadl=0;
all_deadl=0;
for i=1:1:n
    q(i)=i;
end
Efs=10*0.0000000000001;
do=sqrt(Efs/Eamp);
p=0.05;
h=100; %%%%Hard Thres%%%hold H(t)
s=2; %%%%Soft thres%%%hold S(t)

sv=0;
%%%%%previously Sensed value S(v)
%temperature range
tempi=50;
tempf=200;

for r=0:1:rmax
    cv = tempi + (tempf-tempi).*rand(1,1); %%%%Current sensing value C(v)
    packets_TO_BS=0;
    packets_TO_CH=0;
    packets_TO_BS_nofailure=0;
    packets_TO_CH_nofailure=0;
    r
    if(mod(r, round(1/p) )==0)
        for i=1:1:n
            S(i).G=0;

        end
    end

    hold off;

%Number of dead nodes
dead=0;

figure(2);
for i=1:1:n
    if (S(i).E<=0)

        %checking if there is a dead node or active node or normal node by comparing the energy of
        each node

        plot(S(i).xd,S(i).yd,'red .');
        dead=dead+1;
        %When the first node dies
        if (dead==1)
            if(flag_first_deadl==0)
                first_deadl=r;% first node death FND++++++++++++++++++++
                flag_first_deadl=1;
            end
        end

        if(dead==0.1*n)
            if(flag_teenth_deadl==0)
                teenth_deadl=r;
                flag_teenth_deadl=1;
            end
        end

        if(dead==0.5*n)
            if(flag_half_deadl==0)
                half_deadl=r;
                flag_half_deadl=1;
            end
        end

        if(dead==n)
            if(flag_all_deadl==0)
                all_deadl=r;
                flag_all_deadl=1;
            end
        end

        hold on;
    end
    if(S(i).E>0)
        S(i).type='N';

        plot(S(i).xd,S(i).yd,'g o');
    end
end

```

```

        hold on;
    end
end
plot(S(n+1).xd,S(n+1).yd,'^', 'MarkerSize', 12, 'MarkerFaceColor', 'g');

STATISTICS(r+1).DEAD=dead;
DEAD(r+1)=dead; % first node death FND at rth round
alive(r+1)=n-dead;

%countCHs=0;
cluster=1;
for i=1:1:n
    if(S(i).E>0)
        temp_rand=rand; % radom number generate by node
        if ( (S(i).G)<=0)

            %Election of Cluster Heads using the treshhold equation
            tt=(p/(1-p*mod(r,round(1/p)))));
            % if r<100
            % bb(r+1)=tt;
            % end

            if temp_rand<=tt %desion for node to be CH
                S(i).type='C';
                S(i).G=round(1/p)-1; % set of nodes that have not been CH
                C(cluster).xd=S(i).xd;
                C(cluster).yd=S(i).yd;
                plot(S(i).xd,S(i).yd,'k*');
                distance=sqrt( (S(i).xd-(S(n+1).xd) )^2 + (S(i).yd-(S(n+1).yd) )^2 ); %
distance of CH from BS
                C(cluster).distance=distance;
                C(cluster).id=i;
                cluster=cluster+1;
                X(cluster)=S(i).xd;
                Y(cluster)=S(i).yd;
                line([S(i).xd S(n+1).xd], [S(i).yd S(n+1).yd], 'LineStyle', ':');
                packets_TO_BS_nofailure=packets_TO_BS_nofailure+1;
                ss=rand(1);
                goodl = ss > p1
                if goodl == 1
                    packets_TO_BS=packets_TO_BS+1;
                    C(cluster).status=1;

                    if (distance>do)
                        Etx(i)=(Eelec+Eamp*distance*distance*distance*distance)*k;
                    else
                        Etx(i)=(Eelec+Efs*distance*distance)*k;
                    end
                    S(i).E=S(i).E-Etx(i)-EDA;

                    end
                end
            end
end
for i=1:1:n
    if ( S(i).type=='N' && S(i).E>0 )
        if(cluster-1>=1)
            min_dis=sqrt( (S(i).xd-S(n+1).xd)^2 + (S(i).yd-S(n+1).yd)^2 );
            min_dis_cluster=0;
            for c=1:1:cluster-1
                temp=min(min_dis,sqrt( (S(i).xd-C(c).xd)^2 + (S(i).yd-C(c).yd)^2 ) ); % a
normal node is selection its cluster
                if ( temp<min_dis )
                    min_dis=temp;
                    min_dis_cluster=c;
                end
            end
            min_dis;
            if (cv >= h)
                test = cv-sv;
                if (test >= s)
                    if(min_dis_cluster~=0)
                        packets_TO_CH_nofailure=packets_TO_CH_nofailure+1;
                        if C(min_dis_cluster).status == 1
                            packets_TO_CH=packets_TO_CH+1;
                        end
                    else
                        packets_TO_BS=packets_TO_BS+1;
                    end
                end
            end
        end
    end
end

```

```

%           packets_TO_BS_nofailure=packets_TO_BS_nofailure+1;
%           end
%           end
%           end
%calucating energies
%           if (min_dis>do)
%               Etx(i)=(Eelec+Eamp*min_dis*min_dis*min_dis*min_dis)*k;
%           else
%               Etx(i)=(Eelec+Efs*min_dis*min_dis)*k;
%           end

%           S(i).E=S(i).E-Etx(i);
%           if(min_dis_cluster~=0)
%               Erx=Eelec*k;
%               S(C(min_dis_cluster).id).E = S(C(min_dis_cluster).id).E- (Erx + EDA);
%           end
%           S(i).min_dis=min_dis;
%           S(i).min_dis_cluster=min_dis_cluster;
%       end
%   end
%   hold on;
%   energy=0;
%   for i=1:n
%       if S(i).E>0
%           energy=energy+S(i).E;
%       end
%   end

%   PACKETS_TO_CH(r+1)=packets_TO_CH;
%   PACKETS_TO_BS(r+1)=packets_TO_BS;
%   residue_energy(r+1)=energy;
%   PACKETS_TO_BS_nofailure(r+1)= packets_TO_BS_nofailure;
%   PACKETS_TO_CH_nofailure(r+1)=packets_TO_CH_nofailure;

%   end

%   [vx,vy]=voronoi(X,Y);
%   plot(X,Y,'r*',vx,vy,'b-');
%   hold on;
%   voronoi(X,Y);
%   axis([0 150 0 100]);

%   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   for i=1:l:rmax+1
%       q(i)=i;
%   end
%   figure(3);
%   plot(q,DEAD);
%   title('dead nodes');
%   figure(4);
%   plot(q,residue_energy);
%   title('Residual energy');
%   figure(5);
%   plot(q,alive);
%   title('Alive nodes');
%   figure(6);
%   plot(q,PACKETS_TO_CH);
%   title('PACKETS_TO_CH');
%   figure(7);
%   plot(q,PACKETS_TO_BS);
%   title('PACKETS_TO_BS');

```

ПРИЛОЖЕНИЕ В. 2D и 3D

Script code in Mat LAB

```
%-- 18.11.14 15:00 --%
[x,y]=meshgrid(5:5:50,0.90:0.01:0.99)
z=-log(1-y)./(3.14*x^2)
surf(x,y,z);
3D figure:
[x,y]=meshgrid(5:5:50,0.90:0.01:0.99)
z=-log(1-y)./(4/3*3.14*x^3)*1000
surf(x,y,z);

function createaxes(Parent1, xdata1, ydata1, zdata1)
%CREATEAXES(PARENT1,XDATA1,YDATA1,ZDATA1)
% PARENT1: axes parent
% XDATA1: surface xdata
% YDATA1: surface ydata
% ZDATA1: surface zdata

% Auto-generated by MATLAB on 17-Dec-2014 20:47:20

% Create axes
axes1 = axes('Parent',Parent1,'Position',[0.2058 0.05024 0.6375 0.886],...
    'Color',[1 0.949 0.8667]);
view(axes1,[-37.5 30]);
grid(axes1,'on');
hold(axes1,'all');

% Create surf
surf(xdata1,ydata1,zdata1,'Parent',axes1,'MarkerSize',3,'Marker','*');

% Create xlabel
xlabel('Coverage Range');

% Create ylabel
ylabel('Coverage Fraction(%)' );

% Create zlabel
zlabel('Sensor Density(sensor/m^3)');

[x,y]=meshgrid(5:5:50,0.90:0.01:0.99)
```

```

z1=-log(1-y)./(3.14*x^2)
surf(x,y,z1)
xlabel('x'),ylabel('y'),zlabel('z')
title('3D Plot Example of multiple graphs')
hold on
z2=-log(1-y)./(4/3*3.14*x^3) *1000
surf(x,y,z2)
hold off
surf(xdata1,ydata1,zdata1,'Parent',Parent1,'MarkerSize',3,...
    'MarkerEdgeColor',[1 0.6941 0.3922],...
    'Marker','o',...
    'LineStyle','-',...
    'DisplayName','3D WSNs');
r=30
y=(0.90:0.01:0.99)
x1=-log(1-y)./(3.14*r^2)
x2=-log(1-y)./(4/3*3.14*r^3)
figure
plot(y,x1,'--go',y,x2,':r*')

function createaxes1(Parent1, X1, YMatrix1)
%CREATEAXES1(PARENT1,X1,YMATRIX1)
% PARENT1: axes parent
% X1: vector of x data
% YMATRIX1: matrix of y data

% Auto-generated by MATLAB on 17-Jan-2015 15:48:06
% Create axes
axes1 = axes('Parent',Parent1);
box(axes1,'on');
hold(axes1,'all');
% Create multiple lines using matrix input to plot
plot1 = plot(X1,YMatrix1,'Parent',axes1,'MarkerSize',8,'LineWidth',1.5);
set(plot1(1),'Marker','*','LineStyle','-','Color',[1 0 0],
    'DisplayName','3D WSNs');
set(plot1(2),'Marker','o','LineStyle','-','Color',[0 1 0],
    'DisplayName','2D WSNs');

```

ПРИЛОЖЕНИЕ Г. Акты внедрения результатов работы

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Юридический адрес: набережная реки Мойки,
д. 61, Санкт-Петербург, 191186

Почтовый адрес: пр. Большевиков, д. 22, корп. 1,
Санкт-Петербург, 193232
Тел.(812) 3263156, Факс: (812) 3263159
E-mail: rector@sut.ru
ИНН 7808004760 КПП 784001001
ОГРН 1027809197635 ОКТМО 40909000

30.03.2016 № 416/01
на № _____ от _____



Утверждаю
Первый проректор –
проректор по
учебной работе СПб ГУТ
проф. Г.М.Машков

Акт

о внедрении научных результатов,
полученных Аль-Кадами Нассером Ахмедом Салехом в диссертационной работе
"Исследование алгоритмов кластеризации в беспроводных сенсорных сетях".

Комиссия в составе декана факультета Инфокоммуникационных сетей и систем Л.Б.Бузюкова, заместителя заведующего кафедрой сетей связи и передачи данных Р.В.Киричка и заведующей лабораторией кафедры сетей связи и передачи данных А.Ф.Вороновой составила настоящий акт в том, что научные результаты, полученные Аль-Кадами Нассером Ахмедом Салехом в диссертации "Исследование алгоритмов кластеризации в беспроводных сенсорных сетях", использованы при чтении лекций, проведении практических и лабораторных работ по следующим дисциплинам:

1. Интернет Вещей и самоорганизующиеся сети (Рабочая Программа № 02.12.12/717, утверждена Первым проректором-проректором по учебной работе Г.М.Машковым 18.09.2015), раздел Программы:
 - Ad Нос или самоорганизующиеся сети. Приложения самоорганизующихся сетей. Всепроницающие сенсорные сети как технологическая основа внедрения концепции Интернета Вещей. Кластеризация сенсорных сетей и основные методы кластеризации, включая биоподобные алгоритмы. Особенности сетевой безопасности в сенсорных сетях.

2. Современные проблемы науки в области инфокоммуникаций (Рабочая Программа № 02.12.15/783, утверждена Первым проректором-проректором по учебной работе Г.М.Машковым 21.09.2015), разделы Программы:

- Самоорганизующиеся сети. Примеры самоорганизующихся сетей, услуги и приложения таких сетей,
- Трафик в самоорганизующихся сетях и алгоритмы выбора головного узла кластера.

В указанных дисциплинах используются следующие новые научные результаты, полученные Н.А.С. Аль-Кадами в диссертационной работе:

- адаптивный алгоритм кластеризации для беспроводных сенсорных сетей с мобильными узлами под названием МАСА, отличающийся комплексным применением известных ранее комбинированного критерия прогнозирования и значения пригодности сенсорного узла для выполнения роли головного, который за счет комплексного использования указанных выше величин обеспечивает большее значение длительности жизненного цикла и увеличение длительности периода стабильности по сравнению с известными алгоритмами;
- отказоустойчивый алгоритм кластеризации для беспроводных сенсорных сетей под названием FT-TEEN, отличающийся от известного алгоритма TEEN наличием резервных головных узлов кластера, обнаружением и восстановлением отказов в БСС, что позволяет увеличить число пакетов, успешно полученных как в головных узлах кластеров, так и на базовой станции;
- методика размещения сенсорных узлов для двумерных (2D) и трехмерных (3D) БСС, отличающаяся от известных тем, что обеспечивается по крайней мере 90% покрытие пространства в

зависимости от соотношения плотности размещения и радиуса действия
сенсорного узла R_S .

Декан факультета ИКСС

Л.Б.Бузюков

Зам. заведующего кафедрой сети связи

Р.В.Киричек

Зав. лабораторией кафедры сетей связи

А.Ф.Воронова