УДК 621.311

М.Ю.Зеляновський, О.В. Тимченко

МЕТОДИКА МОДЕЛЮВАННЯ МЕТОДІВ ПОБУДОВИ ПЕРСОНАЛЬНИХ ЛОКАЛЬНИХ МЕРЕЖ БЕЗДРОТОВОГО ДОСТУПУ

Подано спосіб дослідження методу побудови бездротової сенсорної мережі шляхом моделювання за допомогою симулятора реального часу «Сніг». Наведено покрокову методику на прикладі запропонованого методу побудови разом із програмним кодом та поясненнями до нього. Проведено експерименти з методом побудови та показано спосіб покращення методу побудови шляхом зміни комунікаційного радіусу.

Ключові слова: бездротові сенсорні мережі, метод побудови, симулятор реального часу.

The manner of topology construction method analysis of wireless sensory network using the real-time simulation tool called “SNOW”. Step-by-step instruction is given for simplest topology construction method, accompanied with commented program code is given. Executed experiments with newly added construction method and shown method of its improvement using communication radius changing.

Keywords: wireless sensory networks, method of construction, real-time simulator.

Вступ

Моделювання обраного методу побудови бездротових спеціалізованих сенсорних мереж (БСМ) засобами симулятора «Сніг» [1, 2] вимагає попереднього опису поведінки вузлів у класах симулятора з використанням мови програмування C# [3]. Дана стаття розглядає покрокову методику, що допоможе реалізувати такі стимулятори, дослідити та перевірити ефективність методів побудови мереж довільної складності.

Для розгляду методики оберемо початковий метод побудови бездротових спеціалізованих сенсорних мереж, що полягає у обміні привітальних повідомлень і формуванню зв’язків із сусідніми вузлами (далі в програмному коді метод «TC123»). Подамо загальний порядок опису досліджуваного методу побудови та деталізуємо методику, яку проілюстровано рядом експериментів, в яких виконано оптимізацію параметрів досліджуваного методу побудови.

Порядок опису методу побудови БСМ

1. Оголошення підтримки нового контролю топології (КТ) в симуляторі;
2. Додавання класу з описом поведінки досліджуваного КТ;
3. Оголошення типів повідомлень;
4. Опис станів протоколу КТ;

5. Оголошення формату повідомлення;
6. Відокремлення методів та властивостей КТ;
7. Опис властивостей КТ;
8. Опис методів КТ:
   а) створення та ініціалізація;
   б) основні дії та переключення станів;
   в) переходи між станами;
   г) обробка вхідного повідомлення;
   д) реєстрація TC123Methods;
9. Виправлення помилок компіляції.

Покрокова методика

1. Оголошуємо підтримку нового КТ в симуляторі “TC123“. У файлі: \SIM4\NetworkComponents\TopologyControls\AvailableTopologyControls.cs додаємо елемент в перелічуваний тип AvailableTopologyControls:

```
public enum AvailableTopologyControls
{
    Z2a,
    KNEIGH,
    KNEIGH_OPT,
    TC123,
}
```

2. Додаємо клас з описом поведінки досліджуваного КТ (*.cs файл) у відповідну папку. Утворений шлях до файлу: \SIM4\NetworkComponents\ TopologyControls\TC123.cs. Вміст новоствореного файлу:

```
namespace SIM4.NetworkComponents.TopologyControls
{
  public class TC123
  {
  }
}
```

3. У файлі TC123.cs поза класом TC123 але в просторі імен SIM4.Network Components.TopologyControls оголошуємо типи пакетів, що будуть використовуватись як internal enum (внутрішній перелічуваний тип). Зважаючи на вибраний метод побудови, тестовий КТ матиме всього три типи пакетів: HelloReq, HelloResp та Data. Отже:

```
namespace SIM4.NetworkComponents.TopologyControls
{
  internal enum TC123PacketTypes
  {
    HelloReq,
    HelloResp,
    Data,
  }
```

```
    public class TC123
    {
    }
}
```

4. Зауважимо, що тут визначаємо внутрішні імена типів повідомлень. З точки зору симулятора існує всього лише два типи пакетів: “Service” та “Data”, в той час як кожен із КТ може мати визначено довільну кількість різних типів повідомлень.

5. Описуємо стани, в якому може перебувати досліджуваний метод КТ. В даному випадку маємо лише два стани: HelloExchange та Data:

```
internal enum TC123ProtocolState
{
    HelloExchange,
    Data,
}
```

Зауважимо, що тут визначаємо внутрішні імена станів протоколу. З точки зору симулятора існує всього лише два стани протоколу: “Service” та “Data”, в той час як кожен із КТ може мати довільну кількість різних станів протоколу.

6. Оголошуємо формат повідомлення як internal class TC123Message. Спочатку додаємо інформацію про намір використовувати додаткові простори імен:

```
using SIM4.NetworkComponents.Interfaces;
using SIM4.ServiceNetworkComponents;
```

Далі наслідуємо від визначених у стимуляторі інтерфейсів IPacket, ISimulatorPacket і явно реалізуємо методи цих інтерфейсів. Інтерфейс IPacket вимагає реалізації властивості відправник (“Sender”), та властивостей потужності відправки та визначеної потужності при прийомі (“TxPa” та “RxPa” відповідно). Зауважимо, що “RxPa” набуває дійсного значення на стороні приймача. Додаємо властивість, що буде нести інформацію про те, якого саме типу є пакет:

```
public TC123PacketTypes PacketType;
```

Інтерфейс ISimulatorPacket вимагає реалізації методу, що дозволить симулятору під час збору статистики встановити, до якого з двох зрозумілих йому типів належить пакет. За це відповідатиме наступний метод, що повертає SnowEnginePacketType:

```
SnowEnginePacketType ISimulatorPacket.GetPacketType()
```

Клас досліджуваного ТК набуде наступного вигляду:

```
using SIM4.NetworkComponents.Interfaces;
using SIM4.ServiceNetworkComponents;

namespace SIM4.NetworkComponents.TopologyControls
```

```
{
    internal enum TC123PacketTypes
    {
        HelloReq,
        HelloResp,
        Data,
    }
    internal enum TC123ProtocolState
    {
        HelloExchange,
        Data,
    }
            internal class TC123Message: IPacket, ISimulatorPacket
    {
        int IPacket.Sender { get; set; }
        double IPacket.TxPa { get; set; }
        double IPacket.RxPa { get; set; }
        public TC123PacketTypes PacketType;
        public int Recipient;
        SnowEnginePacketType ISimulatorPacket.GetPacketType()
        {
            switch (PacketType)
            {
                case TC123PacketTypes.HelloReq:
                    return SnowEnginePacketType.Service;
                    break;
                case TC123PacketTypes.HelloResp:
                    return SnowEnginePacketType.Service;
                    break;
                case TC123PacketTypes.Data:
                    return SnowEnginePacketType.Data;
                    break;
                default:
                    break;
            }
            return SnowEnginePacketType.Service;
        }
    }
    public class TC123
    {
    }
}
```

7. Перетворюємо наш public class TC123 у два класи – TC123Properties та TC123Methods. Клас TC123 більше не потрібний:

```
 public class TC123Properties : ITopologyControlProperties,
      ISimulatorTopologyControl
{
```

```
}
public class TC123Methods : ITopologyControlMethods
{
}
```

Справа в тому, що концепція симулятора передбачає поділ КТ на дві сутності – методи та властивості. В силу того, що:

а) властивості (TC123Properties) зберігають інформацію для кожного з об'єктів-вузлів (стани протоколу, проміжні змінні)

б) методи (TC123Methods) – статичні і працюють з усіма об'єктами однаково (обробляють стани протоколу, проміжні змінні, здійснюють переходи по станах);

властивості та методи не можуть бути описані в одному класі.

Для полегшення взаємодії із симулятором, кожен із цих класів реалізує певні, визначені симулятором інтерфейси. Розглянемо та опишемо кожен із цих класів.

8. TC123Properties наслідується від наступних інтерфейсів:

а) ITopologyControlProperties - кількість прийнятих та переданих пакетів (визначені явно), поточний та наступний стан протоколу (визначені неявно), список вузлів-сусідів та словник маршрутів (визначені явно);

б) ISimulatorTopologyControl - передбачає для симулятора можливість визначення стану протоколу.

Реалізація класу властивостей КТ TC123Properties:

```
public class TC123Properties : ITopologyControlProperties,
ISimulatorTopologyControl
  {
    PacketCounter ITopologyControlProperties.IncomingPacketsTotal { get; set; }
    PacketCounter ITopologyControlProperties.OutgoingPacketsTotal { get; set; }
     IList<int> ITopologyControlProperties.Neighbors { get; set; }
     Dictionary<int, double> ITopologyControlProperties.Routes { get; set; }
     public object CurrentProtocolState { get; set; }
     public object NextProtocolState { get; set; }
     // custom properties:
     public int numOfHelloRx;
     public int numOfHelloTx;
     SnowEngineProtocolState ISimulatorTopologyControl.GetProtocolState()
     {
        switch ((TC123ProtocolState)CurrentProtocolState)
        {
           case TC123ProtocolState.HelloExchange:
              return SnowEngineProtocolState.Service;
              break;
           case TC123ProtocolState.Data:
              return SnowEngineProtocolState.Data;
              break;
```

```
            default:
                break;
        }
        return SnowEngineProtocolState.Service;
    }
    }
```

Тут введено для додаткових поля для підрахунку прийнятих та переданих пакетів типу “Hello”.

9. TC123Methods наслідується від інтерфейсу ITopologyControlMethods, що в свою чергу визначає наступні методи, кожен з яких приймає як аргумент об'єкт типу "Мережевий Пристрій" (INetworkDevice). Власне наступні методи і виконують усі функції для кожного з вузлів мережі:

    1. ITopologyControlMethods.Initialize - створення та ініціалізація складових TC123Properties. Протокол починає функціонувати зі стану TC123ProtocolState.HelloExchange ("Обмін привітаннями"). Окрім іншого, ініціалізуємо кількість прийнятих та переданих повідомлень в 0;

```
void ITopologyControlMethods.Initialize(INetworkDevice device)
{
    var radioDevice = (IRadioDevice)device;
    device.Properties = new TC123Properties();
    var properties = (TC123Properties)device.Properties;
    device.Properties.Neighbors = new List<int>();
    device.Properties.Routes = new Dictionary<int, double>();
    device.Properties.OutgoingPacketsTotal = new PacketCounter();
    device.Properties.IncomingPacketsTotal = new PacketCounter();
    radioDevice.TxMessageBuffer = new TC123Message();
    device.Properties.CurrentProtocolState = new TC123ProtocolState();
    device.Properties.NextProtocolState = new TC123ProtocolState();
    device.Properties.CurrentProtocolState = TC123ProtocolState.HelloExchange;
    device.Properties.NextProtocolState = TC123ProtocolState.HelloExchange;
    radioDevice.CurrentCommunicationRange =
        radioDevice.MaxCommunicationRange;
    properties.numOfHelloRx = 0;
    properties.numOfHelloTx = 0;
}
```

    2. ITopologyControlMethods.Operate – здійснюємо переключення станів і виконуємо відповідні дії. В даному випадку залишаємось в стані «Обміну привітаннями» доки не вишлемо мінімум три «привітання» і не приймемо мінімум три «привітання». Необхідну кількість «привітань» можемо прив’язати до параметра, що задається з графічного інтерфейсу - ExperimentSettings.Connectivity.TargetNodeDegree. Одного разу

перейшовши в стан «Обміну даних» залишаємось у цьому стані. Метод опрацьовує значення поля CurrentProtocolState і на основі нього визначає NextProtocolState – наступний стан протоколу;

```
void ITopologyControlMethods.Operate(INetworkDevice device)
{
   var radioDevice = (IRadioDevice)device;
   var properties = (TC123Properties)device.Properties;
   var txPacket = (TC123Message)radioDevice.TxMessageBuffer;
   switch ((TC123ProtocolState)device.Properties.CurrentProtocolState)
   {
      case TC123ProtocolState.HelloExchange:
         if ((properties.numOfHelloRx >
         ExperimentSettings.Connectivity.TargetNodeDegree
)&&
            (properties.numOfHelloTx >
         ExperimentSettings.Connectivity.TargetNodeDegree
))
         {
            device.Properties.NextProtocolState = TC123ProtocolState.Data;
         }
         else
         {
            txPacket.PacketType = TC123PacketTypes.HelloReq;
            radioDevice.TxMessageBuffer.Sender = device.NetworkId;
            // Send message
radioDevice.SendMessageToEnvironment(radioDevice.TxMessageBuffer);
                        properties.numOfHelloTx += 1;
         }
         break;
      case TC123ProtocolState.Data:
         txPacket.PacketType = TC123PacketTypes.Data;
         radioDevice.TxMessageBuffer.Sender = device.NetworkId;
         // Send message
       radioDevice.SendMessageToEnvironment(radioDevice.TxMessageBuffer);
         break;
      default:
         break;
   }
}
```

3. ITopologyControlMethods.SwitchState – здійснюємо переходи між станами протоколу. У цій функції задаються правила переходів та описуються додаткові дії, які необхідно здійснювати під час переходів. Методи ITopologyControlMethods.Operate та ITopologyControlMethods.SwitchState – два окремі методи,

оскільки опрацьовуємо масив вузлів, для кожного з елементів якого виконується дія. Таке розділення дозволяє емулювати одночасну одночасну роботу всіх вузлів без застосування окремих потоків для кожного і запобігає ситуаціям, коли один з вузлів закінчив роботу чи змінив стан/вимкнув приймач швидше лише тому, що він отримав молодший індекс у масиві. В нашому випадку код методу доволі простий:

```
void ITopologyControlMethods.SwitchState(INetworkDevice device)
{
   var radioDevice = (IRadioDevice)device;
   radioDevice.ReceiverStateCurrent = radioDevice.ReceiverStateNext;
   device.Properties.CurrentProtocolState =
        device.Properties.NextProtocolState;
}
```

4. ITopologyControlMethods.ProcessRxMessageFromEnvironment – здійснює обробку вхідного повідомлення (приймає також як аргумент саме повідомлення – IPacket). Отримавши «привітання» надсилаємо відповідь. Заповнюємо масиви сусідів та маршрутів. В даному випадку масив маршрутів міститиме лише двосторонні зв'язки. Коли надсилаємо відповідь на повідомлення – вказуємо кому саме ми його надсилаємо.

```
void ITopologyControlMethods.ProcessRxMessageFromEnvironment
(INetworkDevice device, IPacket nodeIncomingMessage)
{
   var radioDevice = (IRadioDevice)device;
   var properties = (TC123Properties)device.Properties;
   var rxPacket = (TC123Message)nodeIncomingMessage;
   var txPacket = (TC123Message)radioDevice.TxMessageBuffer;
   switch (rxPacket.PacketType)
   {
      case TC123PacketTypes.HelloReq:
   if(!device.Properties.Neighbors.Contains(nodeIncomingMessage.Sender))
        {
          device.Properties.Neighbors.Add(nodeIncomingMessage.Sender);
        }
   if (!device.Properties.Routes.ContainsKey(nodeIncomingMessage.Sender))
        {
          device.Properties.Routes.Add(nodeIncomingMessage.Sender, 0);
        }
        txPacket.PacketType = TC123PacketTypes.HelloResp;
        radioDevice.TxMessageBuffer.Sender = device.NetworkId;
        txPacket.Recipient = nodeIncomingMessage.Sender;
```

```
            // Send message
        radioDevice.SendMessageToEnvironment(radioDevice.TxMessageBuffer);
            break;
          case TC123PacketTypes.HelloResp:
            if (rxPacket.Recipient == device.NetworkId)
            {
        if (!device.Properties.Routes.ContainsKey(nodeIncomingMessage.Sender))
              {
                device.Properties.Routes.Add(nodeIncomingMessage.Sender, 0);
              }
              properties.numOfHelloRx += 1;
            }
            break;
          case TC123PacketTypes.Data:
            break;
        }
      }
```

10. Реєструємо TC123Methods. У файлі "\SIM4\SimulationSettings Window.xaml.cs" робимо виклик конструктора при виборі нового контролю топології:

```
  public int CurrentTopologyControl
  {
    get
    {
      return (int)ExperimentSettings.CurrentTopologyControlIndex;
    }
    set
    {
ExperimentSettings.CurrentTopologyControlIndex = (AvailableTopologyControls) value;
      switch (value)
      {
        case 0:
ExperimentSettings.CurrentTopologyControl = new Z2aMethods();
          break;
        case 1:
ExperimentSettings.CurrentTopologyControl = new KneighMethods();
          break;
        case 2:
ExperimentSettings.CurrentTopologyControl = new KneighOptMethods();
          break;
        case 3:
ExperimentSettings.CurrentTopologyControl = new TC123Methods();
          break;
      }
      OnPropertyChanged("CurrentTopologyControl");
    }
  }
```

Перевірка функціональності моделі

1. Генеруємо вузли мережі;

Рис.1. Налаштування симулятора при створенні вузлів

2. Змінюємо налаштування. Обираємо контроль топології TC123

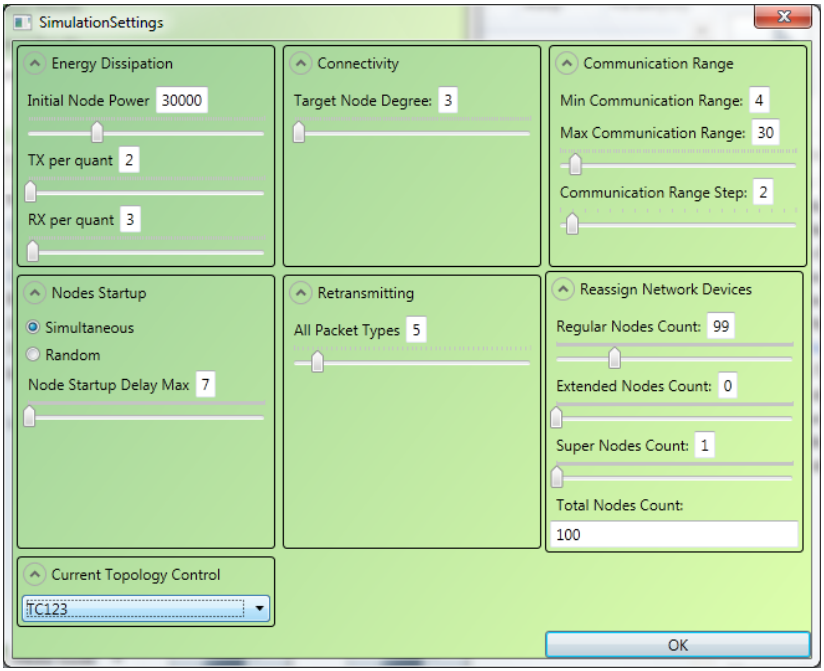

Рис.2. Вікна зміни налаштувань

3. Запускаємо симуляцію

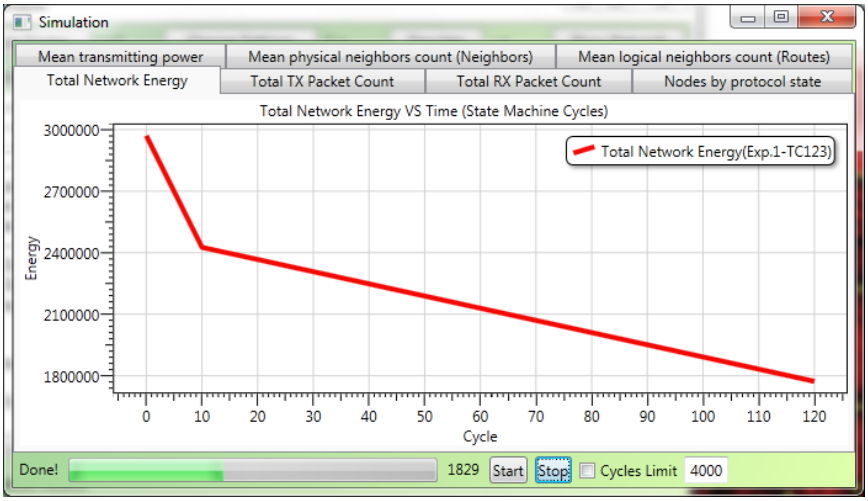

Рис.3. Вікна запуску симуляції; графік сумарного використання енергії

4. Переглядаємо комунікаційний граф, що утворився:

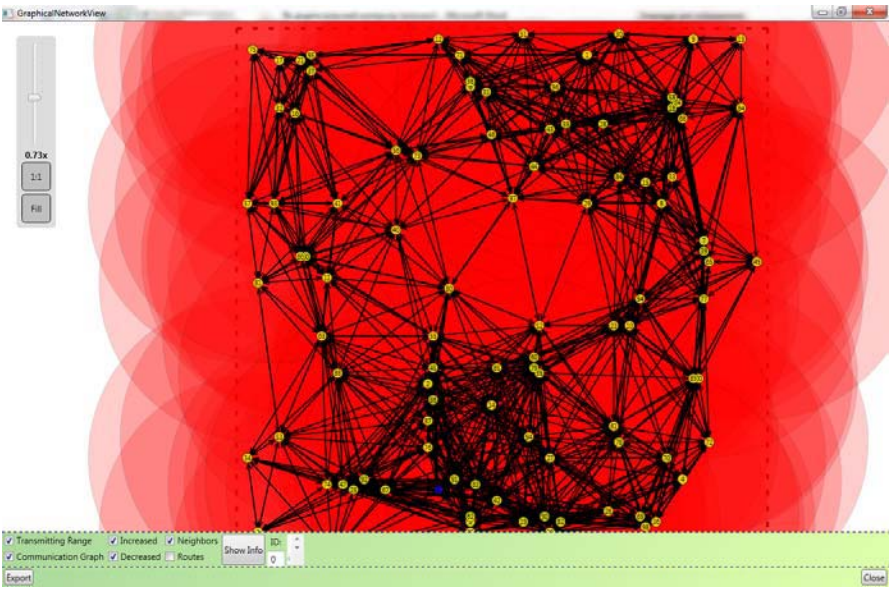

Рис.4. Утворений комунікаційний граф

Очевидно, що комунікаційний граф містить надлишкові ребра. Змінюємо максимальний допустимий комунікаційний радіус. Задаємо його рівним 14, що практично дорівнює критичному радіусу. З графіку на рис. 5 бачимо, що енергоспоживання суттєво знизилось у порівнянні з результатами попереднього експерименту.

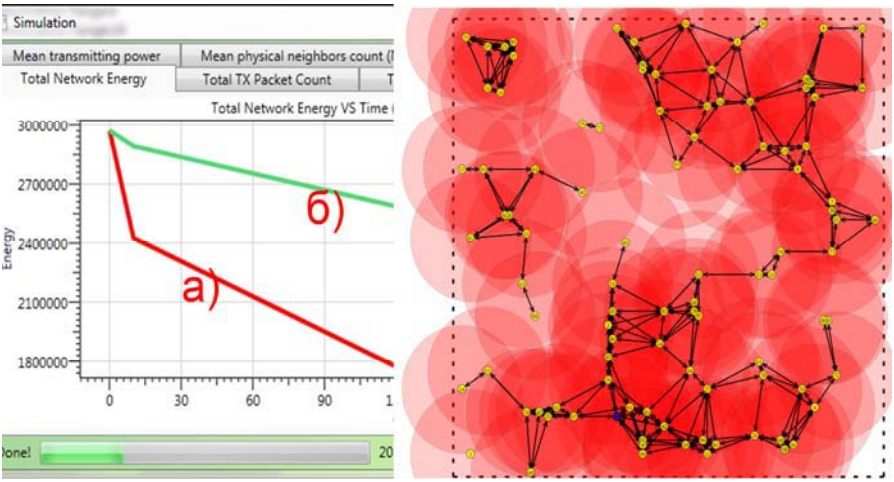

Рис.5. Порівняльні графіки енергоспоживання (зліва) та комунікаційний граф після підстройки допустимого комунікаційного радіусу (справа).
а) сумарна енергія до оптимізації
б) сумарна енергія після оптимізації

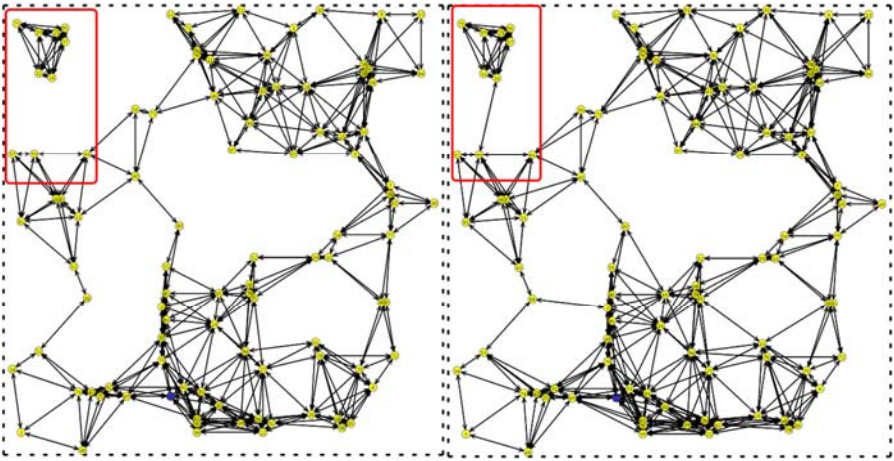

Рис.5. Комунікаційний граф з відокремленою областю (зліва) та зв'язний комунікаційний граф (справа)

5. Оскільки у графі на рис. 5 існують відокремлені компоненти то продовжуємо вдосконалення – збільшуємо дальність передачі «привітальних» повідомлень. Як результат, залишилась лише одна відокремлена компонента (відображення комунікаційних зон покриття вимкнуто) при максимальному радіусі рівному 17 (рис. 6, зліва) та повну зв'язність комунікаційного графа на рис.7 (справа) при його значенні рівному 18.

Висновки

Подано спосіб дослідження методу побудови бездротової сенсорної мережі шляхом моделювання у симуляторі реального часу «Сніг».

Наведено покрокову інструкцію на прикладі найпростішого методу побудови разом із програмним кодом та поясненнями до нього. Загалом опис методи побудови проходить у 9 етапів і вимагає знання мови програмування C#, з використанням якої написаний симулятор.

Проведено експерименти з доданим методом побудови та показано спосіб покращення методу побудови шляхом зміни параметрів комунікаційного радіусу. Показано, що прив'язка границь допустимого комунікаційного радіусу до обчисленого критичного дозволяє знизити сумарне витрати енергоспоживання на побудову мережі до 30% і таким чином дозволяє збільшити час життя мережі

1. *Зеляновський М.Ю., Алхіхі Мухамад, Альбдур Нашат, Самі Аскар.* Засоби для моделювання спеціалізованих та сенсорних мереж бездротового доступу: симулятори роботи комп'ютерних мереж NS-2 та NS-3 // Зб. наук. пр. ІПМЕ НАН України. - Вип.51. - К.: 2009. - С. 203-210.
2. *Зеляновський М.Ю., Тимченко О.В.* Засоби для моделювання спеціалізованих та сенсорних мереж бездротового доступу: симулятор SHAWN // Моделювання та інформаційні технології. Зб. наук. пр. ІПМЕ НАН України. - Вип.54. - К.: 2009. - С. 52-62.
3. *Зеляновський М.Ю., Тимченко О.В.* Інтелектуальна система для локальних мереж бездротового доступу: моделювання методів побудови мережі // Зб. наук. пр. ІПМЕ НАН України. – Вип.53. – К.: 2009. – С. 185-196.
4. http://msdn.microsoft.com/en-us/library/ms752059.aspx
5. http://msdn.microsoft.com/en-us/library/ms754130.aspx
6. http://wpfextensions.codeplex.com/
7. http://dynamicdatadisplay.codeplex.com/
8. http://msdn.microsoft.com/en-us/vcsharp/aa904594
9. http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx

Поступила 31.01.2011р.