Cover

Federal Agency and Organization Element to Which Report is Submitted:
4900
Federal Grant or Other Identifying Number Assigned by Agency:
1816197
Project Title:
NeTS: Small: RUI: Bulldog Mote- Low Power Sensor Node and design Methodologies for Wireless Sensor Networks
PD/PI Name:

- Nan Wang, Principal Investigator
- Woonki Na, Co-Principal Investigator

Recipient Organization:
California State University-Fresno Foundation
Project/Grant Period:
10/01/2018 - 09/30/2021
Reporting Period:
10/01/2019 - 09/30/2020

Prepared by: Russell S. Schellenberg
Daniel Wright

National Science Foundation
Bulldog Mote Project-Protocol Development
Progress as of September 21, 2020

By: Russell Skaggs-Schellenberg and Daniel Wright

Department of Electrical and Computer Engineering
California State University, Fresno
NSF Bulldog Mote Team

Table of Contents

List of Figures and Tables

1. Purpose

The National Science Foundation (NSF) Bulldog Mote Team is part of California State University, Fresno Department of Electrical and Computer Engineering. The purpose of this group is to discover, research, and develop wireless communication protocols and hardware used in Mobile Ad-Hoc Networks (MANETs). This team researches current wireless technologies and algorithms and create new hardware implementations to support current and future protocols for both MANETs and Wireless Sensor Networks (WSNs).

To accomplish this goal, the team has researched new developing wireless schemes, hardware to support the processing of data within these mobile networks and various routing algorithms used to direct data through MANETs.

The remainder of the paper is broken up into three main sections each focusing on a unique project that was conducted over this past year. Each section goes into the background of the project, the development behind it, and the results that it produced. The paper concludes summarizing the accomplishments and touches on how the work performed will assist in future development.

2. Protocol Speed Comparison

2.1 Background

MANET protocols have been evaluated and analyzed previously by the research team to determine the best choice for an application. Considering that the main attraction in using a MANET protocol is that the nodes are mobile. It became apparent that the gap from the previously obtained data was factoring in the node's movement speed.

Determining that one MANET protocol is better than the other without considering this factor may hinder performance once implemented within its application. For example, a protocol may excel in an application that requires lower speeds such as a person walking or a static position such as a parking meter. Though the same protocol may underperform in an application that requires higher speeds such as a bicyclist or a vehicle. The data obtained from this project will help the team in the development of application-specific protocols and others as well.

2.2 Development

The simulation software that was used is NS-3, a robust network simulator. NS-3 was chosen since it supports mobility and WiFi models as well as MANET routing protocols. It has excellent

documentation and allows for simple network analysis. NS-3 can also utilize other tools such as NetAnim. NetAnim visualizes the simulation, allowing for the movement and communication of the nodes to be observed. An example of a simulation being run through NetAnim is shown in Fig. 1.

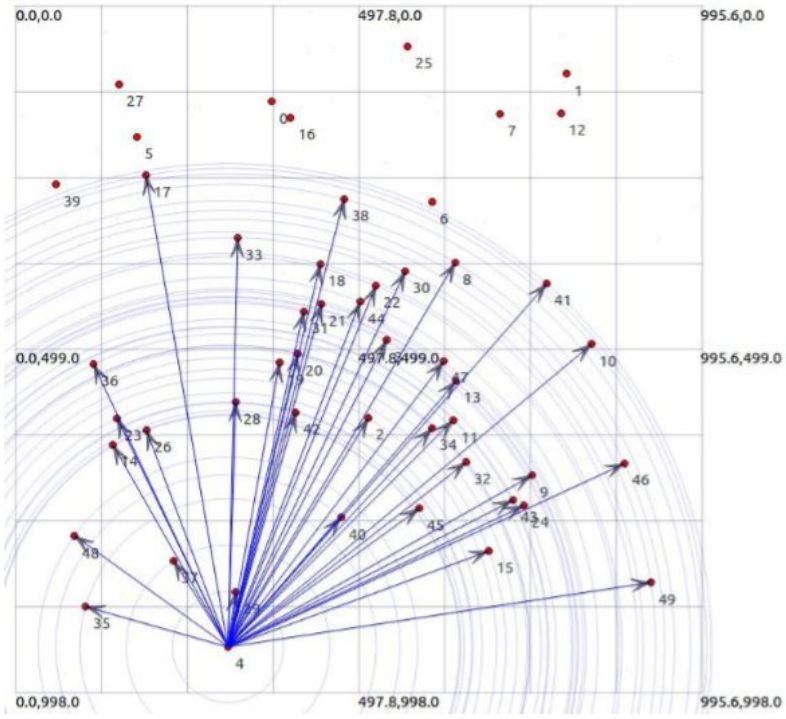

Figure 1: NetAnim visualizing a simulation with a 1000m 2 area.

There are a total of 50 nodes with 10 of them acting as sinks and 10 sending application data. The data is sent at a constant rate of 256 bytes per second in the form of four 64byte UDP packets. Each simulation runs over a period of 200 seconds and the application packets start being sent between the 100 and 101-second marks. The nodes use WiFi 802.11b in ad hoc mode and the transmit power is fixed at 7.5 dBm. Simulations were run for each protocol (AODV, DSDV, DSR,OLSR) across several node speeds: 0, 5, 10, 15, 20, 25, and 30m/s. Each set of simulations were run for areas of 500 m^2 ,750 m^2 , and 1000 m^2 . The complete configuration of the simulation is tabulated below in Table 1.

Table 1: System and Simulation Configuration.

Parameter	Value
Simulator	Network Simulator v3.29
Operating System	Ubuntu 16.04 LTS
Protocols	AODV, DSDV, DSR, OLSR
Simulation Area (m^2)	500, 750, 1000
Total amount of nodes	50
Amount of sink nodes	10
Node Speed (m/s)	0, 5, 10, 15, 20, 25, 30
Simulation Time	200 seconds
Channel Type	Wireless Channel
Mac Protocol	802.11b
Data Packet Size	64 bit
Data Packet Type	UDP
Transmit Power	7.5 dBm

Two metrics were obtained from this experiment. The first was the node's Packet Delivery Ratio (PDR) which is the ratio in which a successful packet is sent then received by its destination. The equation used for this metric is shown below.

$$PDR = \frac{Packets\ Received}{Packets\ Sent}$$

The second metric was the Average End to End Delay (AETED). This was obtained by measuring and then averaging the time it took a packet from one node to another. This equation is shown below.

$$AETED = \frac{1}{n}\sum_{i=1}^{n} Time\ Received_i - Time\ Sent_i$$

These two metrics were chosen, after considering reliability and timing are essential when developing a Wireless Sensor Network (WSN). The development code for the simulation experiment is shown in Appendix A1.

2.3 Results

After obtaining the results from the simulation, the data was tabulated and graphed to be analyzed visually. The first set of graphs consist of the PDR data from the three Simulation Area sizes 500m2, 750m2, and 1000m2 in Figure 2-4 respectively.

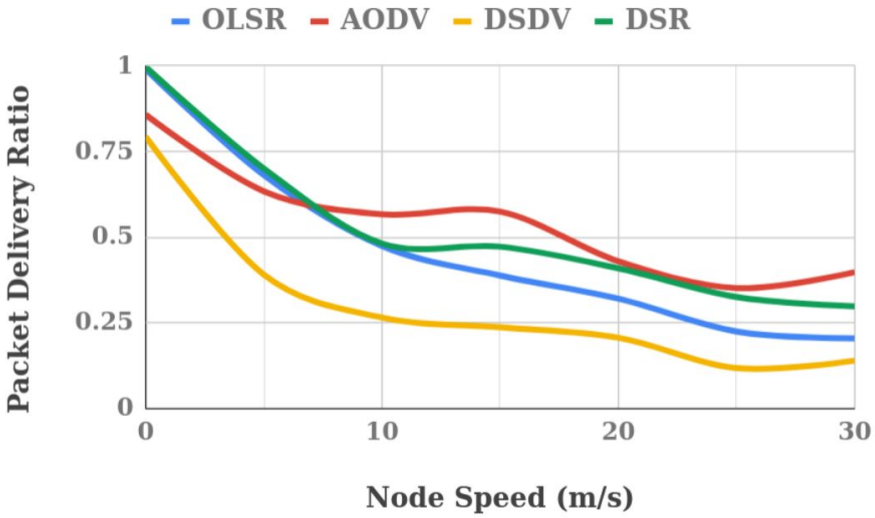

Figure 2: PDR vs node speed in a 500 m^2 area.

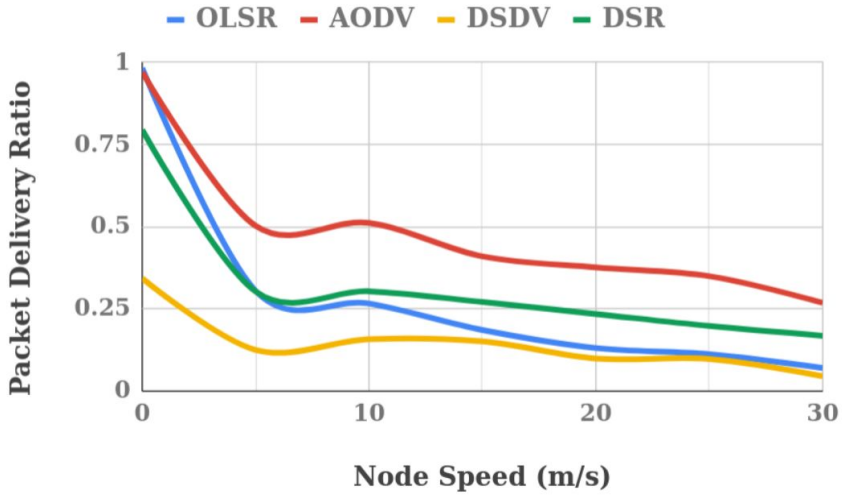

Figure 3: PDR vs node speed in a 750 m^2 area.

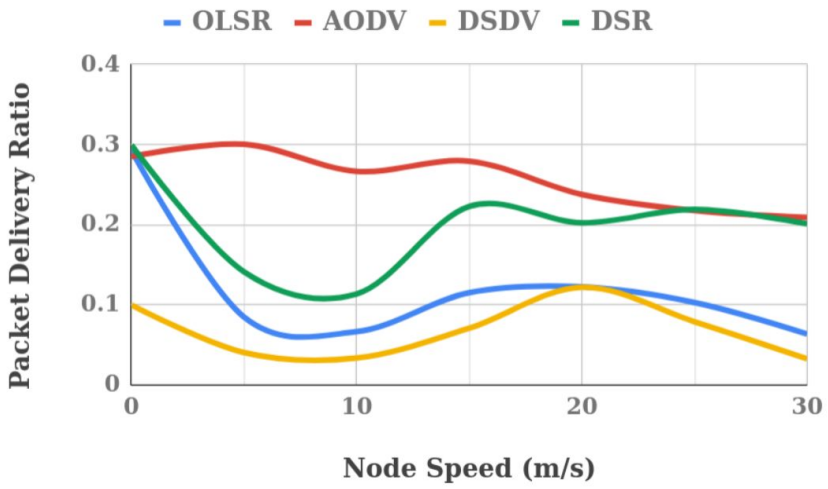

Figure 4: PDR vs node speed in a 1000 m^2 area.

A closer look at the obtained PDR data is shown in Table 2 below.

Table 2: PDR Results.

Area Size (m^2)	Protocols	Speeds (m/s)						
		0	***5***	***10***	***15***	***20***	***25***	***30***
500	AODV	.86	.63	.57	.58	.43	.35	.40
	DSDV	.79	.39	.27	.24	.27	.12	.14
	DSR	.99	.70	.48	.47	.41	.33	.30
	OLSR	.99	.68	.48	.39	.32	.23	.21
750	AODV	.97	.50	.51	.41	.38	.35	.27
	DSDV	.34	.13	.16	.15	.10	.10	.05
	DSR	.80	.30	.3	.27	.23	.20	.17
	OLSR	.98	.31	.27	.19	.13	.11	.07
1000	AODV	.29	.30	.27	.28	.24	.22	.21
	DSDV	.10	.05	.03	.07	.12	.08	.03
	DSR	.30	.14	.11	.22	.20	.22	.20
	OLSR	.29	.08	.07	.12	.12	.10	.06

The second set of graphs consist of the AETED data from the three Simulation Area sizes 500 m^2 , 750 m^2 and 1000 m^2 in Figure 5-7 respectively.

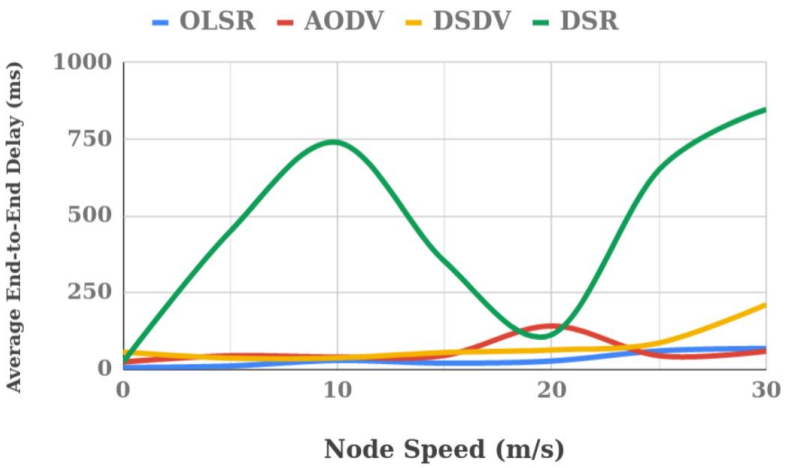

Figure 5: AETED vs node speed in a 500 m^2 area.

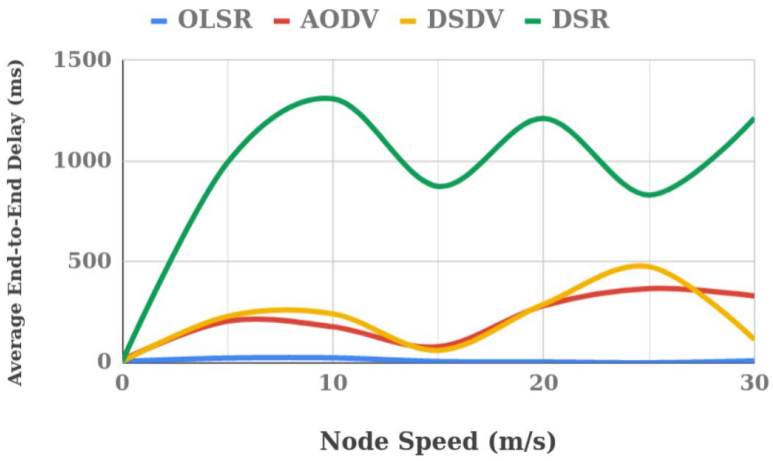

Figure 6: AETED vs node speed in a 750 m^2 area.

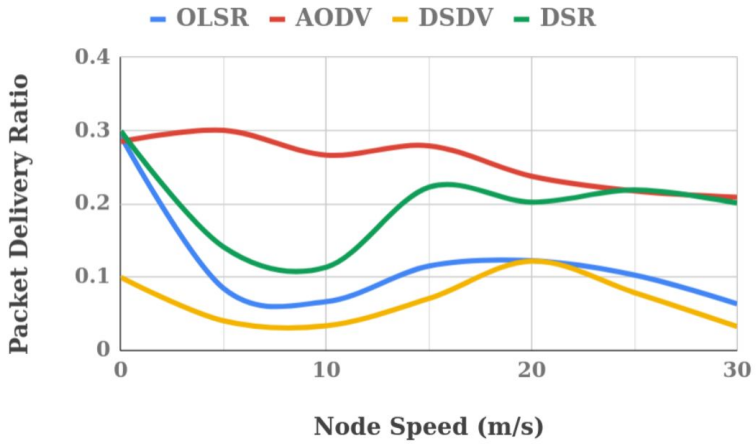

Figure 7: AETED vs node speed in a 1000 m^2 area.

A more detailed view of the obtained AETED data is shown in Table 3 below.

Table 3: AETED Results.

Area Size (m^2)	Protocols	Speeds (m/s)						
		0	***5***	***10***	***15***	***20***	***25***	***30***
500	AODV	25	45	41	46	142	45	60
	DSDV	57	37	38	56	64	87	212
	DSR	24	450	740	352	114	651	847
	OLSR	7	12	30	21	28	61	69
750	AODV	15	207	179	80	284	368	332
	DSDV	8	231	243	62	291	447	115
	DSR	9	995	1310	875	1212	832	1215
	OLSR	7	24	25	7	5	1	10
1000	AODV	13	204	189	376	572	990	838
	DSDV	0	17	113	456	580	22	516
	DSR	5	1221	2109	2199	1476	3411	3670
	OLSR	5	9	9	1	0	0	1

3. Custom Virtual OS Image

3.1 Background

When developing a project, every team member needs to have the same software and tools, including the same version. Having worked with Contiki OS and Contiki-NG OS previously, it became apparent that it was quite difficult to install correctly and efficiently. It was decided that having everyone run the same virtual machine would allow the team to develop more efficiently. Searching online there was only one source that contained Contiki OS, though it was using an outdated version of its operating system as well as not containing the necessary Contiki-NG OS. Therefore the solution arose that having a custom virtual os image with all of the software and tools installed would allow the team a consistent foundation for development.

3.2 Development

It was not necessary to create our base operating system (OS), considering the support that Contiki and Contiki-NG offered it became apparent that the Linux distro Ubuntu would be a viable choice. To extend the support of our developed image, the latest stable version of Ubuntu was used 20.04 which has support for at least the next five years.

This base OS was loading into VirtualBox which is a free virtual machine manager. This software was chosen since it is capable of running on most computers running Windows, Linux, or Mac, as well as being free. Although this custom image will run on other virtual machine managers, it benefited everyone new to virtual machines to have a guide on how to use it on a

single platform. Starting up the Ubuntu image, the necessary libraries and tools were downloaded, installed, and updated.

3.3 Results

The custom virtual OS image was completed and uploaded to a location that is accessible to all team members. This allowed the team to work on the next project more efficiently as well as introducing new team members to the tools. A screenshot of the OS opened within VirtualBox is shown in Figure 8.

Figure 8: Custom virtual OS image running in VirtualBox.

4. Custom S-MAC RPL Protocol

4.1 Background

Having several TelosB motes on hand and other teams developing the Bulldog mote which would run on the same Contiki-NG OS, it became apparent that implementation on that platform would be something the Bulldog Mote team has not done yet. Considering that TelosB motes would be static or mobile at low speeds, based on the data gathered from the speed comparison previously done, a proactive protocol was pursued. Looking at popular proactive protocols being used today, the Routing Protocol for Low-Power and Lossy Networks (RPL) was decided as a viable choice. It was a desirable protocol for low power WSN and ones that did not need high reliability. It was our plan that we would modify the RPL protocol to increase its energy efficiency even further.

4.2 Development

The initial development started using the simulator Cooja which is a tool within Contiki-NG. Using an already created RPL protocol, it was altered so the TelosB would be capable of running it. A three-way handshake was established which allowed the client nodes a connection with the server node, with the limited amount of memory of the TelosB. A communication diagram of this three-way handshake is shown in Figure 9.

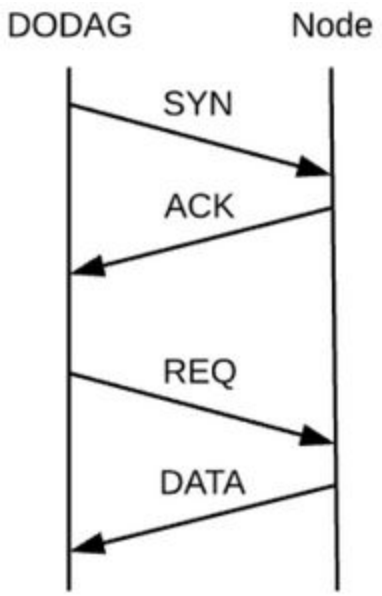

Figure 9: Communication between the server and client.

Once the original RPL protocol was established, the S-MAC protocol was implemented. Normally a protocol would activate its transceiver and receiver regardless if a packet is in transit. What the S-MAC protocol offers is scheduled time where both transmit and receiver are powered down as shown in Figure 10. Although this comes with the great benefit of conserving energy, if a packet was sent to a node that was in its sleeping state, the packet would not be received.

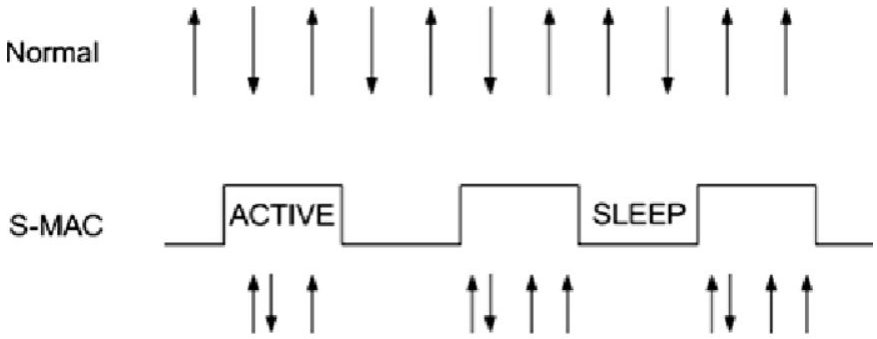

Figure 10: Normal protocol and S-MAC transmission times.

Upon completion of the protocol, the system was first tested within Cooja, first with a single client and server WSN shown in Figure 11. Followed by a WSN that contained 3 clients and required the packet to travel using multiple hops to reach its destination.

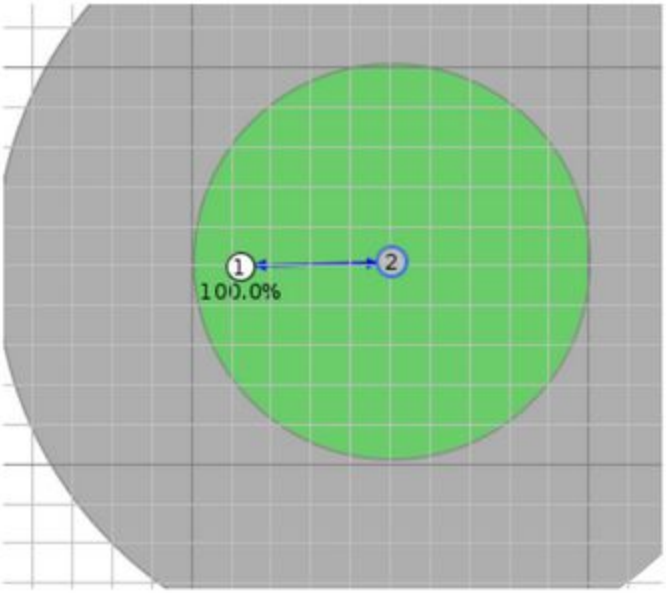

Figure 11: Server and single client WSN simulated within Cooja.

After a successful simulation, the proposed protocol was implemented on the TelosB motes. The first test was performed on a single client and server. Which the energy usage was gathered from the client using a battery charger analyzer after a 24-hour test. This was performed four times, each time using a different transmission cycle: 10 seconds, 100 seconds, 1,000 seconds, and 10,000 seconds. The transmission cycle is how often the server sends out a request to the client nodes to transmit their sensor data. Each client node would transmit their gathered temperature value as well as their battery sensor value. The developed protocol's code can be found in Appendix A2 for the client and Appendix A3 for the server node.

The second test was performed in which the WSN contained 8 clients and a server mote. This test was performed over 12 hours, again with the four different transmission cycles. The clients surrounded the server mote in a star topology as shown in Figure 12.

Figure 12: 8 clients and single server mote.

4.1 Results

To analyze the performance of the proposed system multiple methods were used to get a clear understanding. The first method was using the battery sensor on the TelosB mote. These results were gathered after a 24-hour test, in which the same single server and client were used with the same rechargeable batteries. The gathered data was organized into four graphs, with the transmission cycle being the varying factor. The 10 seconds, 100 seconds, 1,000 seconds, and 10,000 seconds graphs are shown in Figure 13 through Figure 16 respectively.

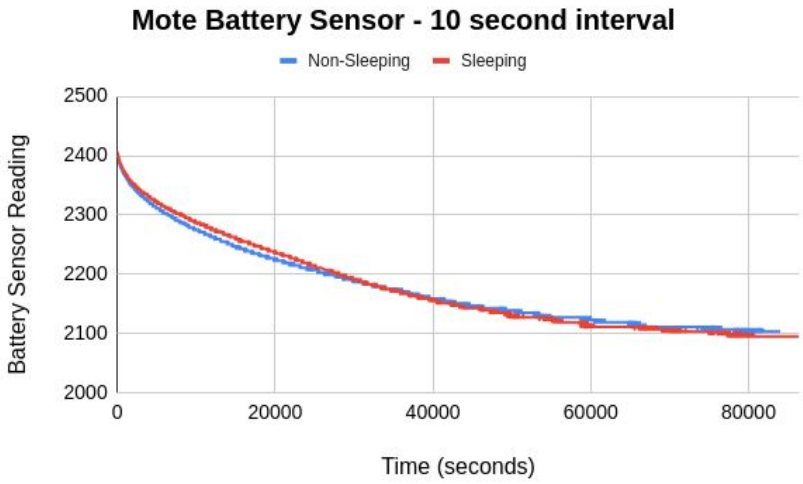

Figure 13: Motes battery sensor test, 10 second transmission cycle.

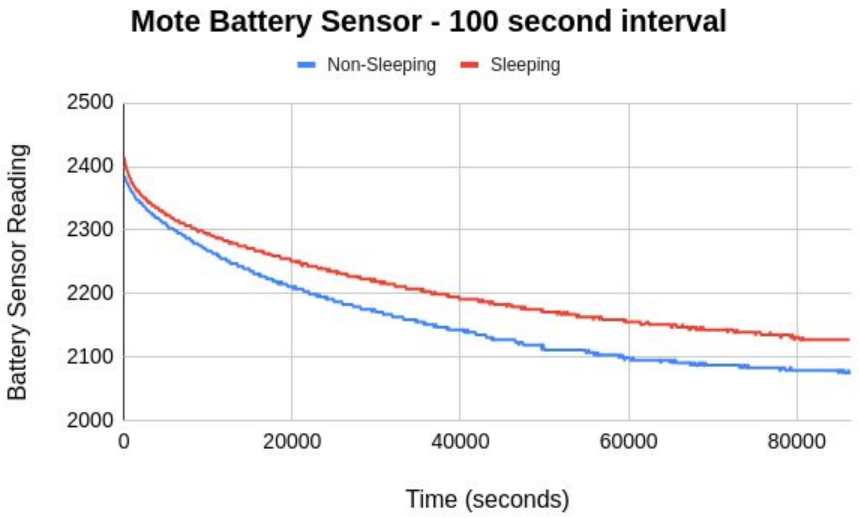

Figure 14: Motes battery sensor test, 100 second transmission cycle.

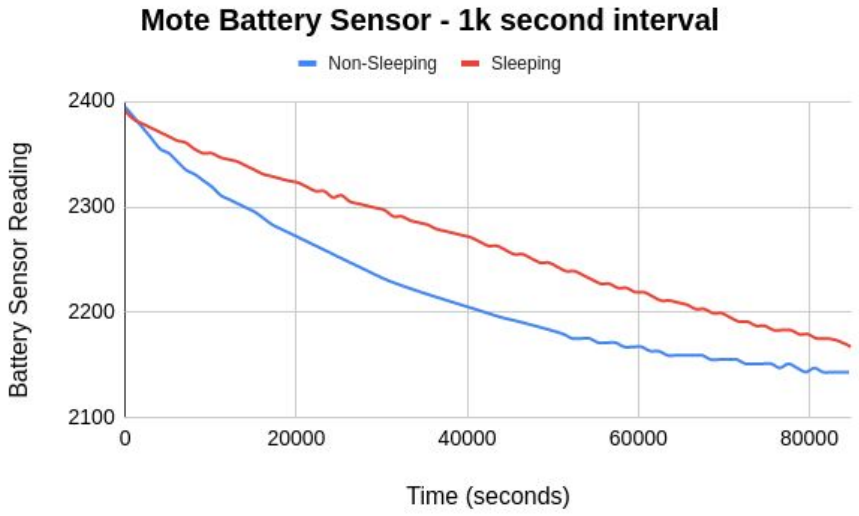

Figure 15: Motes battery sensor test, 1,000 second transmission cycle.

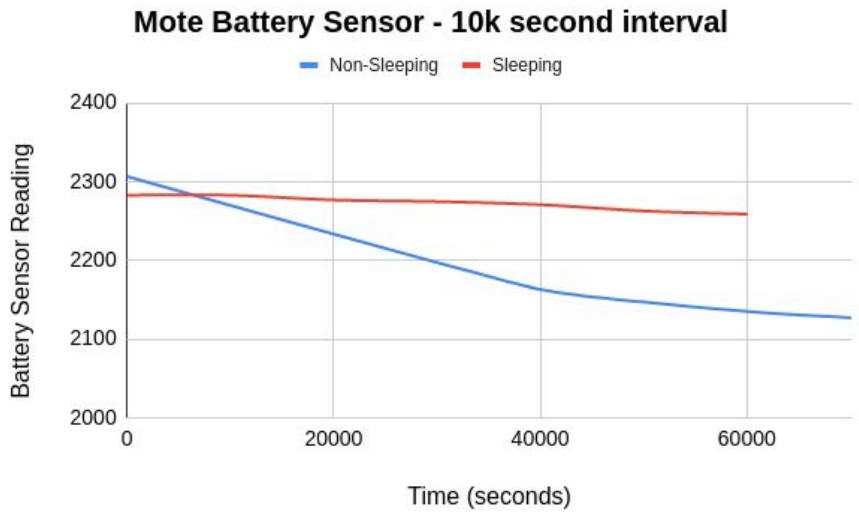

Figure 16: Motes battery sensor test, 10,000 second transmission cycle.

This gave us a rough idea of how efficient our proposed system was, though was not near the accuracy that was desired. Searching for other solutions led to the idea of using a State of Charge (SoC) graph. In which a battery's voltage would determine how much charge was left. Searching the manufacturer's website led to the conclusion that one was not released for the brand of batteries being used. So using a battery charger/analyzer, an SoC graph was created. This graph is shown below in Figure 17.

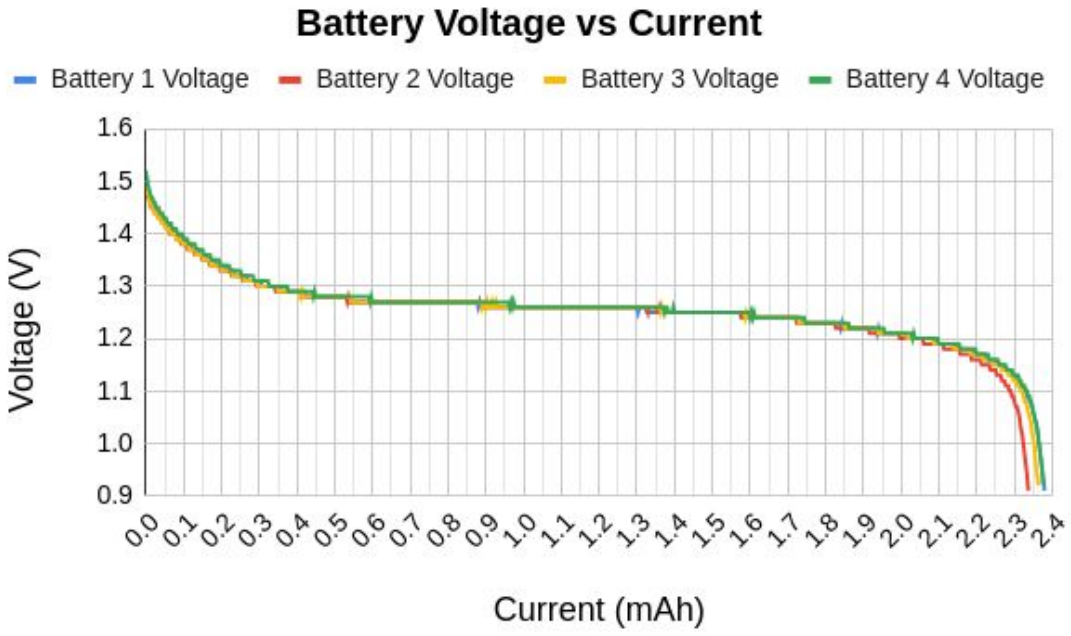

Figure 17: SoC graph created by the team.

This provided a better idea of how much energy was used when correlating the voltage of the battery after the experiment to this graph. Though still did not provide us with the accuracy that was desired. Using the same battery/analyzer, the charge of the batteries was documented after each test. This provided a much better view of the energy used from each transmission cycle. The results gathered are shown in Figure 18.

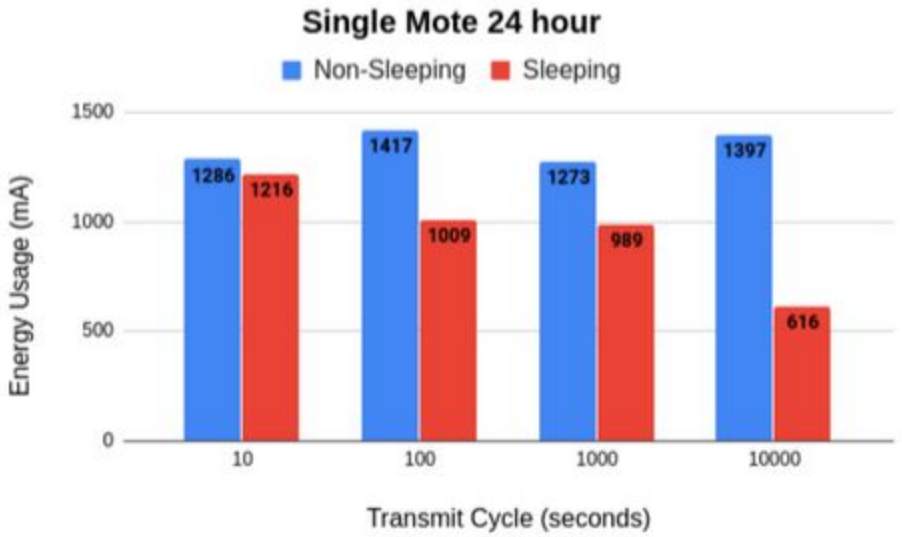

Figure 18: Single mote 24 hour test, analyzing the batteries remaining charge.

The collected data is graphed to display the efficiency of each transmitted cycle as shown in Figure 19.

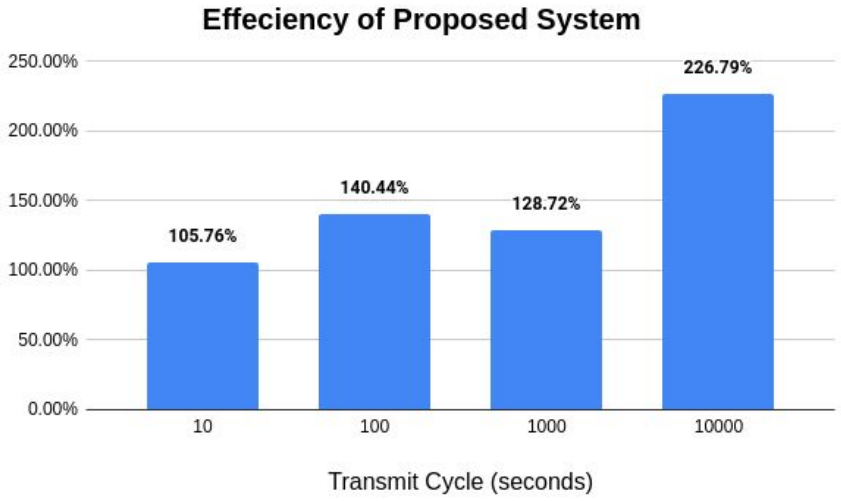

Figure 19: Efficiency of proposed system.

Figure 20 contains the gathered data from a 12 hour experiment, in which the WSN contained 8 clients and a server.

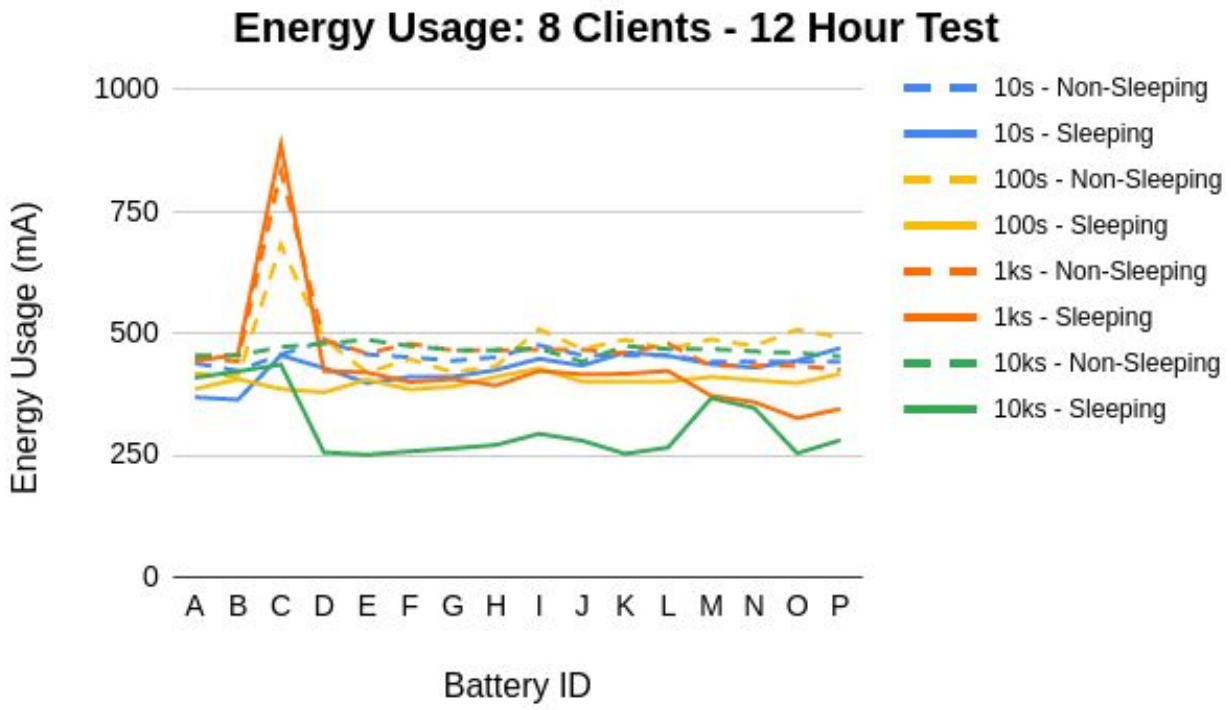

Figure 20: Energy usage of the mote's batteries after each 12 hour test.

The other metric that was analyzed from these experiments was the motes PDR. The single client's PDR is shown in Figure 21, with the PDR from the 8 client test in Figure 22.

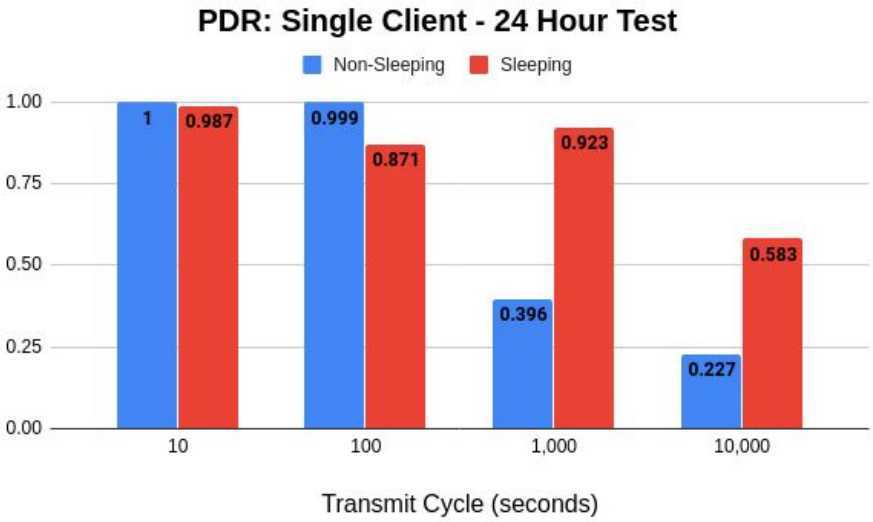

Figure 21: PDR of single mote test over 24-hour period.

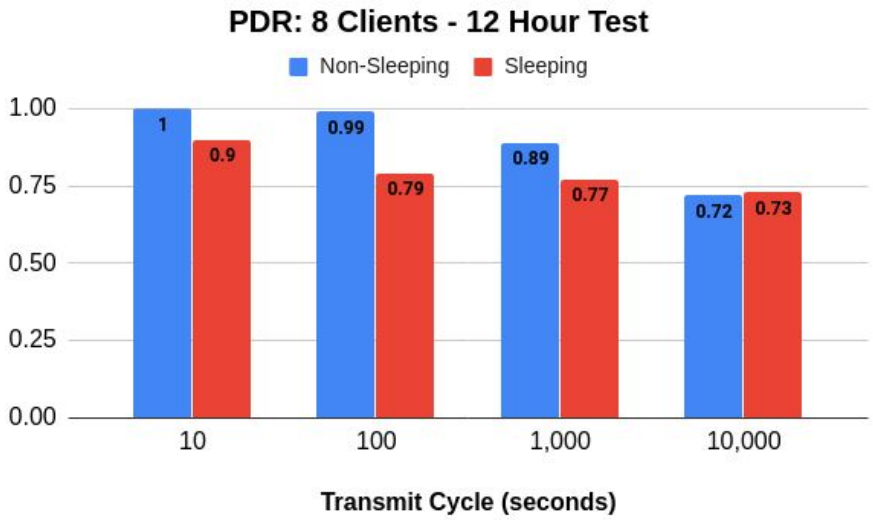

Figure 22: PDR of the eight mote over a 12-hour test period and each transmission cycle.

The PDR of each individual mote can be further analyzed from Table 4, which was used to generate the previous graph.

Table 4: Individual mote's PDR for 12 hour test.

Mote	10s		100s		1,000s		10,000s	
	NON	SLP	NON	SLP	NON	SLP	NON	SLP
c492	1.0	1.0	1.0	0.93	1.0	0.88	1.0	1.0
d080	1.0	1.0	1.0	0.86	1.0	0.88	0.67	1.0
e346	1.0	0.22	0.98	0.05	0.14	0.13	0.06	0
79ff	1.0	1.0	0.99	0.96	0.95	0.74	1.0	1.0
6775	1.0	1.0	0.96	1.0	1.0	1.0	0	0.38
d6ed	1.0	1.0	1.0	0.93	1.0	0.79	1.0	1.0
e862	1.0	1.0	1.0	0.69	1.0	1.0	1.0	0.5
e42b	1.0	1.0	1.0	0.94	1.0	0.76	1.0	1.0

5. Conclusion

Over the past year, three main projects were completed by this team. Speed comparison of MANET protocols was conducted, filling the gap of the speed factor when developing a WSN. This work was then presented at CCWC 202 in Las Vegas, Nevada, and published. A custom virtual OS image was developed as an efficient way for the team to develop with the Contiki/Contiki-NG OS. This was used in the last project as well as being available for future researchers to utilize in their projects. Lastly, a custom RPL protocol was developed using the S-MAC algorithm to increase energy efficiency. The proposed protocol was implemented and tested on the TelosB motes. Analyzing the results, the proposed system was up to 226.79% more energy efficient than its counterpart. This work was submitted for through IEEE UEMCON 2020, and currently pending for publication. The proposed protocol will be implemented on the

Bulldog Motes once completed. These projects would not have been possible without the support of the National Science Foundation's grant # 1816197.

Appendix A1

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2011 University of Kansas
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Justin Rohrer <rohrej@ittc.ku.edu>
 *
 * James P.G. Sterbenz <jpgs@ittc.ku.edu>, director
 * ResiliNets Research Group  http://wiki.ittc.ku.edu/resilinets
 * Information and Telecommunication Technology Center (ITTC)
 * and Department of Electrical Engineering and Computer Science
 * The University of Kansas Lawrence, KS USA.
 *
 * Work supported in part by NSF FIND (Future Internet Design) Program
 * under grant CNS-0626918 (Postmodern Internet Architecture),
 * NSF grant CNS-1050226 (Multilayer Network Resilience Analysis and Experimentation on
GENI),
 * US Department of Defense (DoD), and ITTC at The University of Kansas.
 */

/*
 * This example program allows one to run ns-3 DSDV, AODV, or OLSR under
 * a typical random waypoint mobility model.
 *
 * By default, the simulation runs for 200 simulated seconds, of which
 * the first 50 are used for start-up time.  The number of nodes is 50.
 * Nodes move according to RandomWaypointMobilityModel with a speed of
 * 20 m/s and no pause time within a 300x1500 m region.  The WiFi is
 * in ad hoc mode with a 2 Mb/s rate (802.11b) and a Friis loss model.
 * The transmit power is set to 7.5 dBm.
```

```
*
* It is possible to change the mobility and density of the network by
* directly modifying the speed and the number of nodes.  It is also
* possible to change the characteristics of the network by changing
* the transmit power (as power increases, the impact of mobility
* decreases and the effective density increases).
*
* By default, OLSR is used, but specifying a value of 2 for the protocol
* will cause AODV to be used, and specifying a value of 3 will cause
* DSDV to be used.
*
* By default, there are 10 source/sink data pairs sending UDP data
* at an application rate of 2.048 Kb/s each.    This is typically done
* at a rate of 4 64-byte packets per second.  Application data is
* started at a random time between 50 and 51 seconds and continues
* to the end of the simulation.
*
* The program outputs a few items:
* - packet receptions are notified to stdout such as:
*   <timestamp> <node-id> received one packet from <src-address>
* - each second, the data reception statistics are tabulated and output
*   to a comma-separated value (csv) file
* - some tracing and flow monitor configuration that used to work is
*   left commented inline in the program
*/

#include <fstream>
#include <iostream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"
// Added these last headers
#include "ns3/netanim-module.h"
#include <stdlib.h>
#include <ctime>
#include "ns3/flow-monitor-helper.h"
```

```
using namespace ns3;
using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("manet-routing-compare");

//--------Set Parameters here-----------------
double SimTime = 200.0; // Time to run the simulation, simulation starts at 100
int ProtocolUsed = 2; // 1=OLSR;2=AODV;3=DSDV;4=DSR
int areaSizeX = 500; // Size of the area in meters
int areaSizeY = 500;
int nodeSpeed = 20; // The speed(m/s) of the nodes
int nodeAmount = 20; // The number of nodes in the simulation (too few will give error)




//------------------------------------------
class RoutingExperiment
{
public:
  RoutingExperiment ();
  void Run (int nSinks, double txp, std::string CSVfileName);
  //static void SetMACParam (ns3::NetDeviceContainer & devices,
  //                         int slotDistance);
  std::string CommandSetup (int argc, char **argv);

private:
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node> node);
  void ReceivePacket (Ptr<Socket> socket);
  void CheckThroughput ();

  uint32_t port;
  uint32_t bytesTotal;
  uint32_t packetsReceived;

  std::string m_CSVfileName;
  int m_nSinks;
  std::string m_protocolName;
  double m_txp;
  bool m_traceMobility;
  uint32_t m_protocol;
};
```

```
RoutingExperiment::RoutingExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
    m_CSVfileName ("manet-routing.output.csv"),
    m_traceMobility (false),
    m_protocol (ProtocolUsed)
{
}

static inline std::string
PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet, Address senderAddress)
{
  std::ostringstream oss;

  oss << Simulator::Now ().GetSeconds () << " " << socket->GetNode ()->GetId ();

  if (InetSocketAddress::IsMatchingType (senderAddress))
    {
      InetSocketAddress addr = InetSocketAddress::ConvertFrom (senderAddress);
      oss << " received one packet from " << addr.GetIpv4 ();
    }
  else
    {
      oss << " received one packet!";
    }
  return oss.str ();
}

void
RoutingExperiment::ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  Address senderAddress;
  while ((packet = socket->RecvFrom (senderAddress)))
    {
      bytesTotal += packet->GetSize ();
      packetsReceived += 1;
      NS_LOG_UNCOND (PrintReceivedPacket (socket, packet, senderAddress));
    }
}

void
RoutingExperiment::CheckThroughput ()
```

```
{
  double kbs = (bytesTotal * 8.0) / 1000;
  bytesTotal = 0;

  std::ofstream out (m_CSVfileName.c_str (), std::ios::app);

  out << (Simulator::Now ()).GetSeconds () << ","
      << kbs << ","
      << packetsReceived << ","
      << m_nSinks << ","
      << m_protocolName << ","
      << m_txp << ""
      << std::endl;

  out.close ();
  packetsReceived = 0;
  Simulator::Schedule (Seconds (1.0), &RoutingExperiment::CheckThroughput, this);
}

Ptr<Socket>
RoutingExperiment::SetupPacketReceive (Ipv4Address addr, Ptr<Node> node)
{
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (node, tid);
  InetSocketAddress local = InetSocketAddress (addr, port);
  sink->Bind (local);
  sink->SetRecvCallback (MakeCallback (&RoutingExperiment::ReceivePacket, this));

  return sink;
}

std::string
RoutingExperiment::CommandSetup (int argc, char **argv)
{
  CommandLine cmd;
  cmd.AddValue ("CSVfileName", "The name of the CSV output file name", m_CSVfileName);
  cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
  cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
  cmd.Parse (argc, argv);
  return m_CSVfileName;
}

int
main (int argc, char *argv[])
```

```
{
  RoutingExperiment experiment;
  std::string CSVfileName = experiment.CommandSetup (argc,argv);

  //blank out the last output file and write the column headers
  std::ofstream out (CSVfileName.c_str ());
  out << "SimulationSecond," <<
  "ReceiveRate," <<
  "PacketsReceived," <<
  "NumberOfSinks," <<
  "RoutingProtocol," <<
  "TransmissionPower" <<
  std::endl;
  out.close ();

  int nSinks = 10;      // Number of sinks
  double txp = 7.5;     //Transmission power

  experiment.Run (nSinks, txp, CSVfileName);
}

void
RoutingExperiment::Run (int nSinks, double txp, std::string CSVfileName)
{
  Packet::EnablePrinting ();
  m_nSinks = nSinks;
  m_txp = txp;
  m_CSVfileName = CSVfileName;

  int nWifis = nodeAmount; // Number of Nodes

  double TotalTime = SimTime;
  std::string rate ("2048bps");
  std::string phyMode ("DsssRate11Mbps");
  std::string tr_name ("manet-routing-compare");
  int nodePause = 0; //in s
  m_protocolName = "protocol";

  Config::SetDefault  ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
  Config::SetDefault ("ns3::OnOffApplication::DataRate",  StringValue (rate));

  //Set Non-unicastMode rate to unicast mode
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue
(phyMode));
```

```
  NodeContainer adhocNodes;
  adhocNodes.Create (nWifis); // These are the nodes we will be using, only one type of nodes

  // setting up wifi phy and channel using helpers
  WifiHelper wifi;
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));

  wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
  wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));

  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac, adhocNodes);

  MobilityHelper mobilityAdhoc;
  int64_t streamIndex = 0; // used to get consistent mobility across scenarios

  ObjectFactory pos;
  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
  pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=" +
std::to_string(areaSizeX) + "]"));
  pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=" +
std::to_string(areaSizeY) + "]"));

  Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();
  streamIndex += taPositionAlloc->AssignStreams (streamIndex);

  std::stringstream ssSpeed;
       // Removed the "Min=0", so that each node will move at the designated speed
  ssSpeed << "ns3::UniformRandomVariable[Min=" << nodeSpeed << "|Max=" << nodeSpeed
<< "]";
  std::stringstream ssPause;
```

```
ssPause << "ns3::ConstantRandomVariable[Constant=" << nodePause << "]";
mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                                "Speed", StringValue (ssSpeed.str ()),
                                "Pause", StringValue (ssPause.str ()),
                                "PositionAllocator", PointerValue (taPositionAlloc));
mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
mobilityAdhoc.Install (adhocNodes);
streamIndex += mobilityAdhoc.AssignStreams (adhocNodes, streamIndex);
NS_UNUSED (streamIndex); // From this point, streamIndex is unused

AodvHelper aodv;
OlsrHelper olsr;
DsdvHelper dsdv;
DsrHelper dsr;
DsrMainHelper dsrMain;
Ipv4ListRoutingHelper list;
InternetStackHelper internet;

switch (m_protocol)
  {
  case 1:
    list.Add (olsr, 100);
    m_protocolName = "OLSR";
    break;
  case 2:
    list.Add (aodv, 100);
    m_protocolName = "AODV";
    break;
  case 3:
    list.Add (dsdv, 100);
    m_protocolName = "DSDV";
    break;
  case 4:
    m_protocolName = "DSR";
    break;
  default:
    NS_FATAL_ERROR ("No such protocol:" << m_protocol);
  }

if (m_protocol < 4)
  {
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
  }
```

```
  else if (m_protocol == 4)
    {
      internet.Install (adhocNodes);
      dsrMain.Install (dsr, adhocNodes);
    }

  NS_LOG_INFO ("assigning ip address");

  Ipv4AddressHelper addressAdhoc;
  addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer adhocInterfaces;
  adhocInterfaces = addressAdhoc.Assign (adhocDevices);

  OnOffHelper onoff1 ("ns3::UdpSocketFactory",Address ());
  onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
  onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));

  for (int i = 0; i < nSinks; i++)
    {
      Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i), adhocNodes.Get
(i));

      AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (i), port));
      onoff1.SetAttribute ("Remote", remoteAddress);

      Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();
      ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i + nSinks));
      temp.Start (Seconds (var->GetValue (100.0,101.0)));
      temp.Stop (Seconds (TotalTime));
    }

  std::stringstream ss;
  ss << nWifis;
  std::string nodes = ss.str ();

  std::stringstream ss2;
  ss2 << nodeSpeed;
  std::string sNodeSpeed = ss2.str ();

  std::stringstream ss3;
  ss3 << nodePause;
  std::string sNodePause = ss3.str ();

  std::stringstream ss4;
```

```
  ss4 << rate;
  std::string sRate = ss4.str ();

  //NS_LOG_INFO ("Configure Tracing.");
  //tr_name = tr_name + "_" + m_protocolName +"_" + nodes + "nodes_" + sNodeSpeed +
"speed_" + sNodePause + "pause_" + sRate + "rate";

  //AsciiTraceHelper ascii;
  //Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream ( (tr_name + ".tr").c_str());
  //wifiPhy.EnableAsciiAll (osw);
  AsciiTraceHelper ascii;
  MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name + ".mob"));

       // flowmon is used to view the systems data, need to run the python script
  Ptr<FlowMonitor> flowmon;
  FlowMonitorHelper flowmonHelper;
  flowmon = flowmonHelper.InstallAll ();


  NS_LOG_INFO ("Run Simulation.");

  CheckThroughput ();


  //AnimationInterface anim("manet_tester9000.xml"); // Name of simulation file, load this into
NetAnim



  Simulator::Stop (Seconds (TotalTime));
  Simulator::Run ();

  flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(), false, false);

  Simulator::Destroy ();
}
```

Appendix A2

```
#include "contiki.h"
#include
"net/routing/routing.h"

#include
"random.h"

#include
"net/netstack.h"

#include
"net/ipv6/simple-udp.h"

#include
<msp430.h>

#include
"dev/lpm.h"

#include
"sys/log.h"

#include
"dev/battery-sensor.h"
```

```
#include
<stdio.h>



#include
<stdlib.h>



#include
"dev/sht11/sht11-sensor.h"









// --- START Code Block for Simulation
---



#include
"sys/energest.h"



#include
"sys/timer.h"
```

```
#include
"dev/watchdog.h"



#include
"dev/msp430-lpm-override.h"



extern int
msp430_lpm4_required;









volatile long start,
end;









unsigned long LSPC,
DSPC;



unsigned long voltage = 3.0; // Units =
Volts



unsigned long ActiveCurrent =
330;
```

```
unsigned long LPM1Current =
75;



unsigned long LPM4Current_DIV =
50;



unsigned long ActiveTime =
0;



unsigned long LPM1Time =
0;



unsigned long LPM4Time =
0;



unsigned long TotalTime =
0;









//#define ENERGEST_CONF_ON 1 // This was manually changed in the file for
the Sim
```

```
// --- END Code Block for Simulation ---




#define LOG_MODULE "App"

#define LOG_LEVEL LOG_LEVEL_INFO




#define WITH_SERVER_REPLY
1

#define UDP_CLIENT_PORT
        8800

#define UDP_SERVER_PORT
        5700
```

```
#define SEND_INTERVAL (5 * CLOCK_SECOND)



enum p_type{SYN, DATA};



int
get_temperature(){


return
((sht11_sensor.value(SHT11_SENSOR_TEMP)/10)-396)/10;


}



static uint32_t
get_battery(){
```

```
return
battery_sensor.value(0);



}







static unsigned long to_seconds(uint64_t time){ //
Sim



return (unsigned long)(time /
ENERGEST_SECOND);



}









static struct simple_udp_connection
udp_conn;



uint8_t
p_data[32];
```

```
/*---------------------------------------------------------------------------*/


PROCESS(udp_client_process, "UDP
client");


PROCESS(udp_client_sleep, "UDP client
data");


AUTOSTART_PROCESSES(&udp_client_proces
s);


/*---------------------------------------------------------------------------*/


static void encode_packet(uint8_t* dest, uint16_t datalen, uint8_t* data, enum p_type
type){


int
i;
```

```
(*dest) =
type;



for(i = 0;i < datalen;
i++){



dest[i+2] =
data[i];



}



}
```

```
static
void



udp_rx_callback(struct simple_udp_connection
*c,
```

```
const uip_ipaddr_t
*sender_addr,


uint16_t
sender_port,


const uip_ipaddr_t
*receiver_addr,


uint16_t
receiver_port,


const uint8_t
*data,


uint16_t
datalen)


{




end = clock_seconds(); // For Deep
sleep
```

```
watchdog_stop(
);



msp430_lpm4_required =
0;



ENERGEST_OFF(ENERGEST_TYPE_DEEP_LPM
);



ENERGEST_SWITCH(ENERGEST_TYPE_CPU,
ENERGEST_TYPE_LPM);



LPM1;



watchdog_start(
);









//P5OUT &= ~(1<<5);
```

```
//P5OUT |= (1<<4);



static char
str[32];









LOG_INFO("Received '%.*s' from ", datalen, (char *)
data);



LOG_INFO_6ADDR(sender_addr)
;



LOG_INFO_("\n");



LOG_INFO("Sending Battery: %lu\n",
get_battery());



snprintf(str, sizeof(str), "%lu",
get_battery());



simple_udp_sendto(&udp_conn, str, strlen(str),
sender_addr);
```

```
// --- START Code Block for Simulation
---



/* Update all energest times.
*/



energest_flush
();



printf("\nEnergest:\n
");









ActiveTime =
(to_seconds(energest_type_time(ENERGEST_TYPE_CPU)));



LPM1Time =
(to_seconds(energest_type_time(ENERGEST_TYPE_LPM)));
```

```
LPM4Time = (to_seconds(energest_type_time(ENERGEST_TYPE_DEEP_LPM))); //
Difference in time between going to sleep and waking up



TotalTime =
(to_seconds(ENERGEST_GET_TOTAL_TIME()));




LSPC = voltage *((ActiveTime * ActiveCurrent) + ((LPM1Time + LPM4Time) *
LPM1Current));



printf("Light Sleep
System:\n");



printf(" Active: %5lus, LPM1: %5lus, LPM4: 0s, Total Time: %5lus\n", ActiveTime,
(LPM1Time + LPM4Time), TotalTime);



printf(" Energy Used: %10luuW\n\n",
LSPC);



printf("Simulation_Data: Light %5lu %5lu 0 %5lu %10lu\n", ActiveTime, (LPM1Time +
LPM4Time), TotalTime, LSPC);
```

```
DSPC = voltage *((ActiveTime * ActiveCurrent) + (LPM1Time * LPM1Current) + (LPM4Time /
LPM4Current_DIV));



printf("Deep Sleep
System:\n");



printf(" Active: %5lus, LPM1: %5lus, LPM4: %5lus, Total Time: %5lus\n\n", ActiveTime,
LPM1Time, LPM4Time, TotalTime);



printf(" Energy Used: %10luuW\n\n",
DSPC);



printf("Simulation_Data: Deep %5lu %5lu %5lu %5lu %10lu\n", ActiveTime, LPM1Time,
LPM4Time, TotalTime, DSPC);



//printf("Active: %5lus\n",
ActiveTime);



// Trying to get the Deep LPM
working



/*
```

```
printf("Active: %lus, LPM: %gs, DEEP LPM: %gs, Total time:
%lus\n",



to_seconds(energest_type_time(ENERGEST_TYPE_CP
U)),



(double)()(to_seconds(energest_type_time(ENERGEST_TYPE_LPM))-DeepSleepTi
me)),



DeepSleepTim
e,



to_seconds(ENERGEST_GET_TOTAL_TIME())
);



*/









//LSPC = voltage *(
((ActiveCurrent)*(to_seconds(energest_type_time(ENERGEST_TYPE_CPU)))) +
LPM1Current*(to_seconds(energest_type_time(ENERGEST_TYPE_LPM))));
```

```
// LSPC =
LPM1Current*(to_seconds(energest_type_time(ENERGEST_TYPE_LPM)));



// printf("Light Sleep Power Consumption: %g W\n",
(double)LSPC);









// --- END Code Block for Simulation ---









process_exit(&udp_client_slee
p);



process_start(&udp_client_sleep,
NULL);









#if LLSEC802154_CONF_ENABLED
```

```
LOG_INFO_(" LLSEC LV:%d",
uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));



#endi
f









}



/*---------------------------------------------------------------------------*/



PROCESS_THREAD(udp_client_process, ev,
data)



{



void clock_init
(void);



static struct etimer
periodic_timer;
```

```
static char
str[32];



uip_ipaddr_t
dest_ipaddr;









P5DIR |=
0x70;









PROCESS_BEGIN()
;



SENSORS_ACTIVATE(battery_sensor
);



//P5OUT &= ~(1<<5);
```

```
/* Initialize UDP connection
*/



simple_udp_register(&udp_conn, UDP_CLIENT_PORT,
NULL,



UDP_SERVER_PORT,
udp_rx_callback);









etimer_set(&periodic_timer, random_rand() %
SEND_INTERVAL);



lpm_on()
;



while(1) {



PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer)
);
```

```
if(NETSTACK_ROUTING.node_is_reachable() &&
NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {



/* Send to DAG root */



LOG_INFO("Sending SYN to ");



LOG_INFO_6ADDR(&dest_ipaddr);



LOG_INFO_("\n");









snprintf(str, sizeof(str),
"SYN");



encode_packet(p_data, 0, (uint8_t*)NULL,
SYN);



simple_udp_sendto(&udp_conn, p_data, 2,
&dest_ipaddr);
```

```
process_exit(&udp_client_slee
p);



process_start(&udp_client_sleep,
NULL);



break
;



} else {



LOG_INFO("Not reachable
yet\n");



}









/* Add some jitter */



etimer_set(&periodic_timer,
SEND_INTERVAL
```

```
- CLOCK_SECOND + (random_rand() % (2 * CLOCK_SECOND)));

}

PROCESS_END()
;

}

/*---------------------------------------------------------------------------*/

PROCESS_THREAD(udp_client_sleep, ev,
data)

{

static struct etimer
periodic_timer;
```

```
etimer_set(&periodic_timer, 5 *
CLOCK_SECOND);



PROCESS_BEGIN()
;



LOG_INFO("Going to
sleep\n");









//P5OUT &= ~(1<<4);



//P5OUT |= (1<<5);









// Comment out for simulation
only:



PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer)
);
```

```
// For Simulation: Start timer to see how long it
sleeps.



watchdog_stop(
);



start =
clock_seconds();



//msp430_lpm4_required = 1; //Comment out to stay
awake



ENERGEST_OFF(ENERGEST_TYPE_CPU)
;



ENERGEST_SWITCH(ENERGEST_TYPE_LPM,
ENERGEST_TYPE_DEEP_LPM);



//LPM4; //Comment out to stay
awake
```

```
  watchdog_start(
  );









  //printf("Start_2: %4lu\n", start); // Used for
  Debugging









  PROCESS_END()
  ;


}
```

Appendix A3

```
/*
 * Redistribution and use in source and binary forms, with or
without


 * modification, are permitted provided that the following
conditions


 * are
met:


 * 1. Redistributions of source code must retain the above
copyright


 * notice, this list of conditions and the following
disclaimer.


 * 2. Redistributions in binary form must reproduce the above
copyright


 * notice, this list of conditions and the following disclaimer
in the


 * documentation and/or other materials provided with the
distribution.
```

* 3. Neither the name of the Institute nor the names of its
contributors

* may be used to endorse or promote products derived from this
software

* without specific prior written
permission.

*

* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS''
AND

* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE

* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE
LIABLE

* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS

* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

* SUCH DAMAGE.

*

* This file is part of the Contiki operating
system.

```
*

*/

#include
"contiki.h"

#include
"net/routing/routing.h"

#include
"net/netstack.h"

#include
"net/ipv6/simple-udp.h"

#include
"string.h"
```

```
#include
"random.h"



#include
"dev/sht11/sht11-sensor.h"



#include
<msp430.h>



#include
<stdlib.h>









#include
"dev/lpm.h"









#include
"sys/log.h"



#define LOG_MODULE "App"
```

```
#define LOG_LEVEL LOG_LEVEL_INFO

#define UDP_CLIENT_PORT          8800
        // 8765

#define UDP_SERVER_PORT          5700
        // 5678

#define SEND_INTERVAL            (5 *
CLOCK_SECOND)

#define ATTEMPTS
5

enum p_type{SYN, DATA};
```

```
static struct simple_udp_connection
udp_conn;



uip_ipaddr_t*
sink_addrs;



uint8_t sink_addrs_len =
0;



uint16_t rec_data =
0;



uint8_t
a_i;



uint8_t
s_i;



char syn[] =
"SYN";
```

```
PROCESS(udp_server_process, "UDP
server");



AUTOSTART_PROCESSES(&udp_server_proces
s);









void
init_sinks(){



int j;



for(j = 0;j < 16;j++){



uip_ip6addr(&sink_addrs[j], 0, 0, 0, 0, 0, 0, 0,
0);



}
```

```
}




void add_sink(const uip_ipaddr_t*
sink){



int j;



for(j = 0;j <
sink_addrs_len;j++){



if(uip_ipaddr_cmp(&sink_addrs[j],
sink)){



break
;



}



}
```

```
if(j ==
sink_addrs_len){



sink_addrs[sink_addrs_len] =
*sink;



sink_addrs_len+
+;



}



}









void set_rec_data(const uip_ipaddr_t* sink, uint8_t
data){



int j;



for(j = 0;j <
sink_addrs_len;j++){
```

```
if(uip_ipaddr_cmp(&sink_addrs[j],
sink)){



rec_data &= ~(1<<j);



rec_data |= data <<
j;



}



}



}









/*---------------------------------------------------------------------------*/



static
void
```

```
udp_rx_callback(struct simple_udp_connection
*c,



const uip_ipaddr_t
*sender_addr,



uint16_t
sender_port,



const uip_ipaddr_t
*receiver_addr,



uint16_t
receiver_port,



const uint8_t
*data,



uint16_t
datalen)



{
```

```
if(data[1] == SYN){



LOG_INFO("Received 'SYN' from ");



LOG_INFO_6ADDR(sender_addr)
;



LOG_INFO_("\n\n");



add_sink(sender_addr
);



}else{



LOG_INFO("Received DATA '%.*s' from ", (datalen), (char *)
(data));



LOG_INFO_6ADDR(sender_addr)
;
```

```
LOG_INFO_("\n\n");



set_rec_data(sender_addr,
1);



}



}



/*---------------------------------------------------------------------------*/



PROCESS_THREAD(udp_server_process, ev,
data)



{



// P5DIR |=
0x32;



char req[] =
"REQ";
```

```
static struct etimer
periodic_timer;



uip_ipaddr_t
zero_addr;



uip_ip6addr(&zero_addr, 0, 0, 0, 0, 0, 0, 0,
0);









PROCESS_BEGIN()
;



sink_addrs = (uip_ipaddr_t*) malloc(16 *
sizeof(uip_ipaddr_t));



init_sinks(
);



lpm_on()
;
```

```
/* Initialize DAG root */


NETSTACK_ROUTING.root_start();




/* Initialize UDP connection
*/


simple_udp_register(&udp_conn, UDP_SERVER_PORT,
NULL,


UDP_CLIENT_PORT,
udp_rx_callback);




etimer_set(&periodic_timer, random_rand() %
SEND_INTERVAL);


while(1){
```

```
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer)
);



for(s_i = 0; s_i < sink_addrs_len;
s_i++){



if(!uip_ipaddr_cmp(&sink_addrs[s_i],
&zero_addr)){



for(a_i = 0; a_i < ATTEMPTS;
a_i++){



if((rec_data & (1<<s_i)) > 0)
break;



LOG_INFO("Sending '%.*s' to ", strlen(req), (char *)
req);



LOG_INFO_6ADDR(&sink_addrs[s_i]);



LOG_INFO_("\n");



simple_udp_sendto(&udp_conn, (char *) req, strlen(req),
```

```
&sink_addrs[s_i]);



etimer_set(&periodic_timer, SEND_INTERVAL + (random_rand() % (5 *
CLOCK_SECOND)));



PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer)
);



}



if((rec_data & (1<<s_i)) == 0){



LOG_INFO("Sink unresponsive:
");



LOG_INFO_6ADDR(&sink_addrs[s_i]);



LOG_INFO_("\n\n");



}
```

```
}


}


rec_data =
0;


etimer_set(&periodic_timer, 20 * CLOCK_SECOND + (random_rand() % (1 *
CLOCK_SECOND)));


}


PROCESS_END()
;


}

/*---------------------------------------------------------------------------*/
```