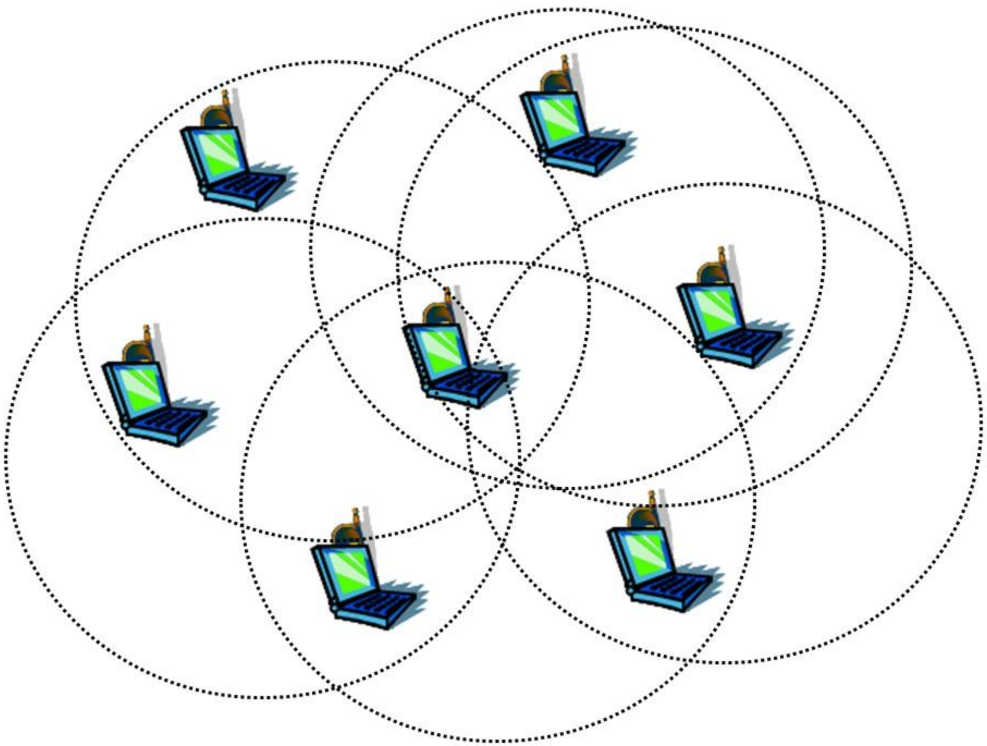
Practical Broadcast Tree Construction with Potential Game for Energy-Efficient Data Dissemination in Ad-Hoc Network

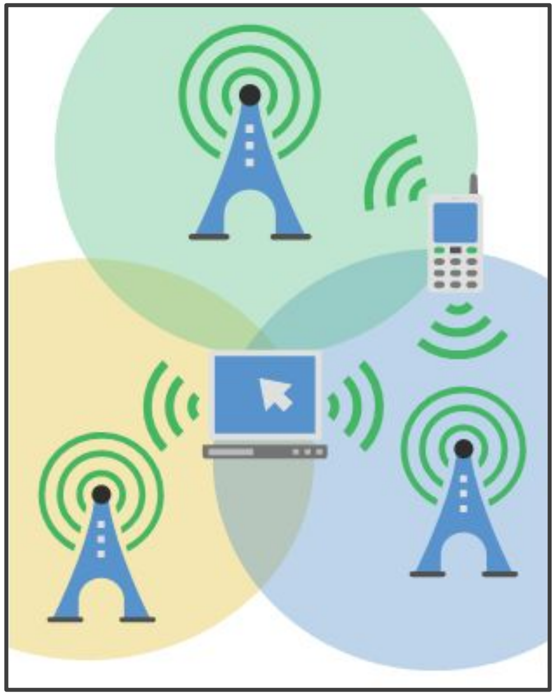


09/02/2018 | Sergio Domínguez | sdominguez@seemoo.tu-darmstadt.de

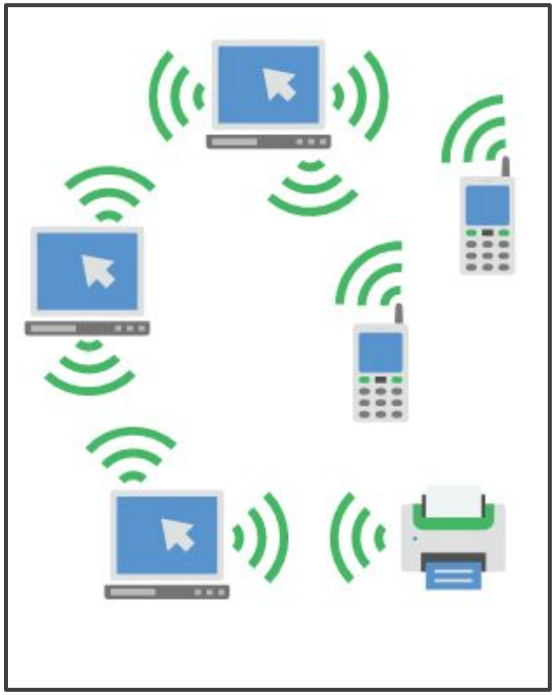
Wireless ad hoc networks

SECURE MOBILE NETWORKING

Infrastructure



Ad Hoc



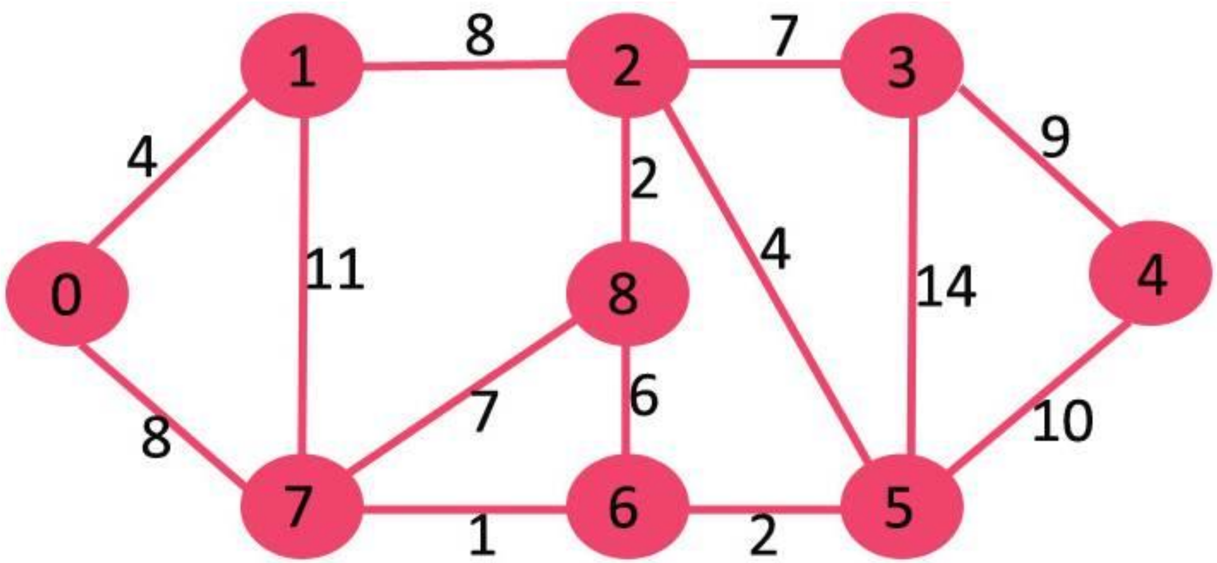
Motivation

What is the existing problem?

Our focus will be set in how packets are transmitted during a broadcast session.

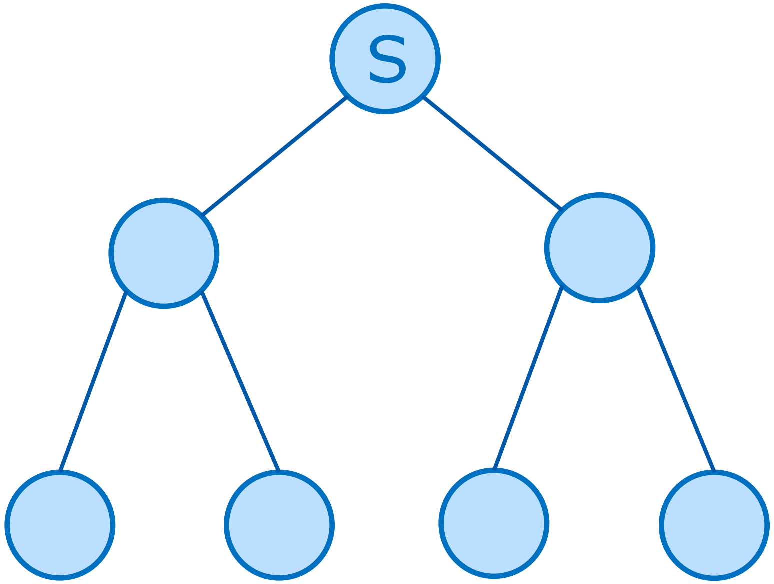
Other approaches

Dijkstra algorithm

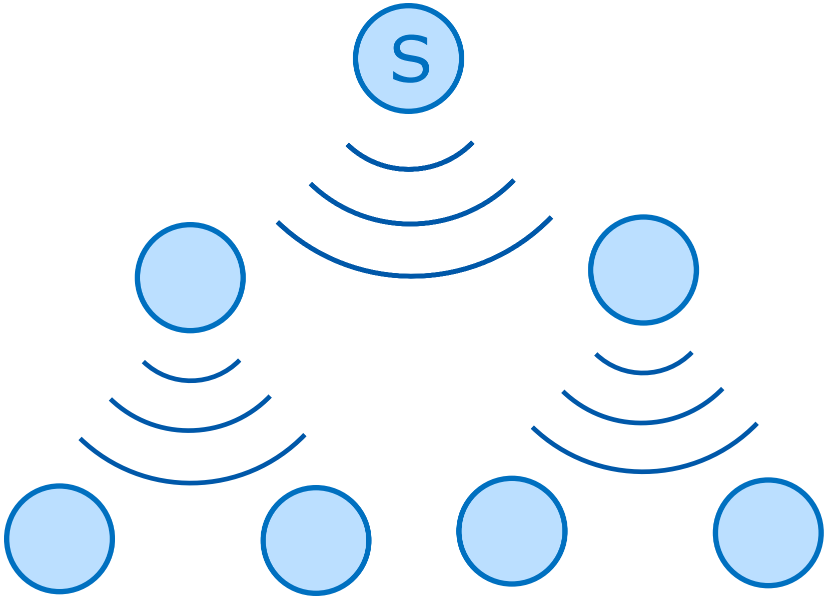


The broadcast nature of wireless networks

Wired



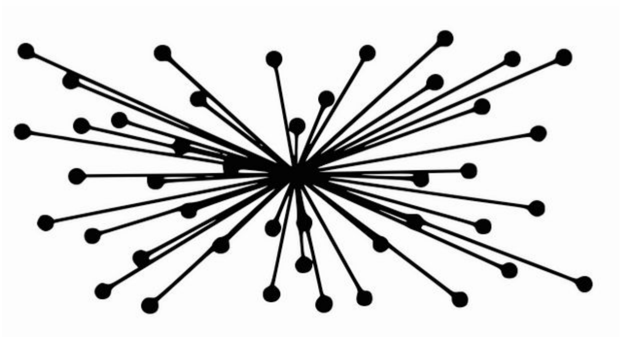
Wireless



Other approaches

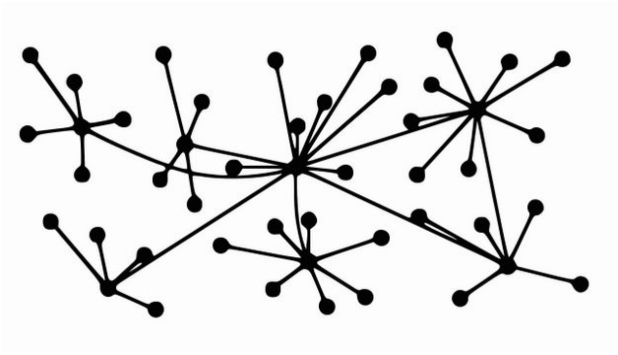
Centralized

BIP, BIPSW



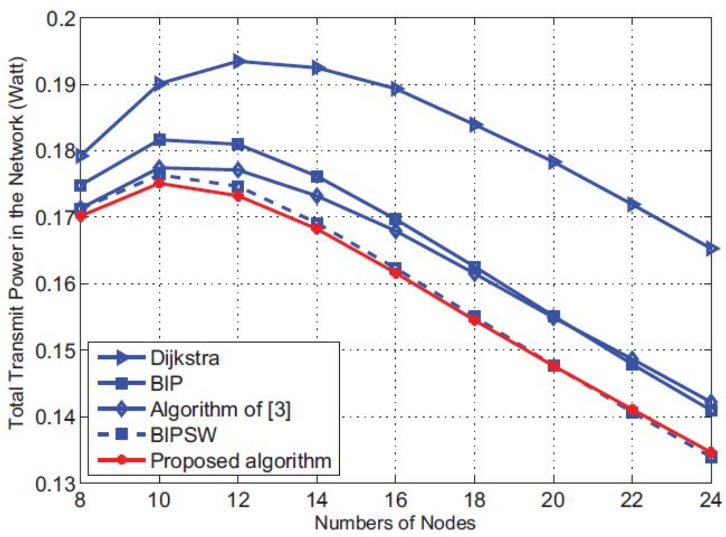
Decentralized

BDP, DynaBIP



Foundation

Why another implementation?



[1]

[1] M. Mousavi, H. Al-Shatri, M. Wichtlhuber, D. Hausheer, and A. Klein, "Energy-efficient data dissemination in Ad Hoc networks: Mechanism design with potential game," in *Wireless Communication Systems (ISWCS), 2015 International Symposium on*, 2015, pp. 616–620.

Energy-efficient data dissemination in Ad Hoc networks

SECURE MOBILE NETWORKING

Key points about this work[1]:

A theoretical design based on the principles of game theory is proposed.

- **Game theory and Nash Equilibrium (NE)**
- **Minimum transmit power and cost definition**
- **Cycle avoidance in the broadcast tree**
- **Weakly dominant strategy**

[1] M. Mousavi, H. Al-Shatri, M. Wichtlhuber, D. Hausheer, and A. Klein, "Energy-efficient data dissemination in Ad Hoc networks: Mechanism design with potential game," in *Wireless Communication Systems (ISWCS), 2015 International Symposium on*, 2015, pp. 616–620.

Idealized assumptions

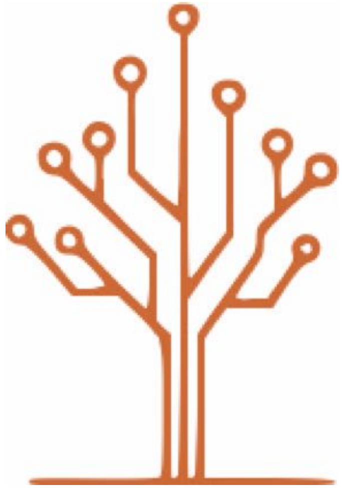
What elements in the original paper [1] were undefined, unrealistic or impractical?

- Knowledge of the neighbors of a node
- Knowledge of the minimum transmit power to reach a node
- Message scheduling mechanism
- Knowledge of the actions of other nodes
- Knowledge of the end of game

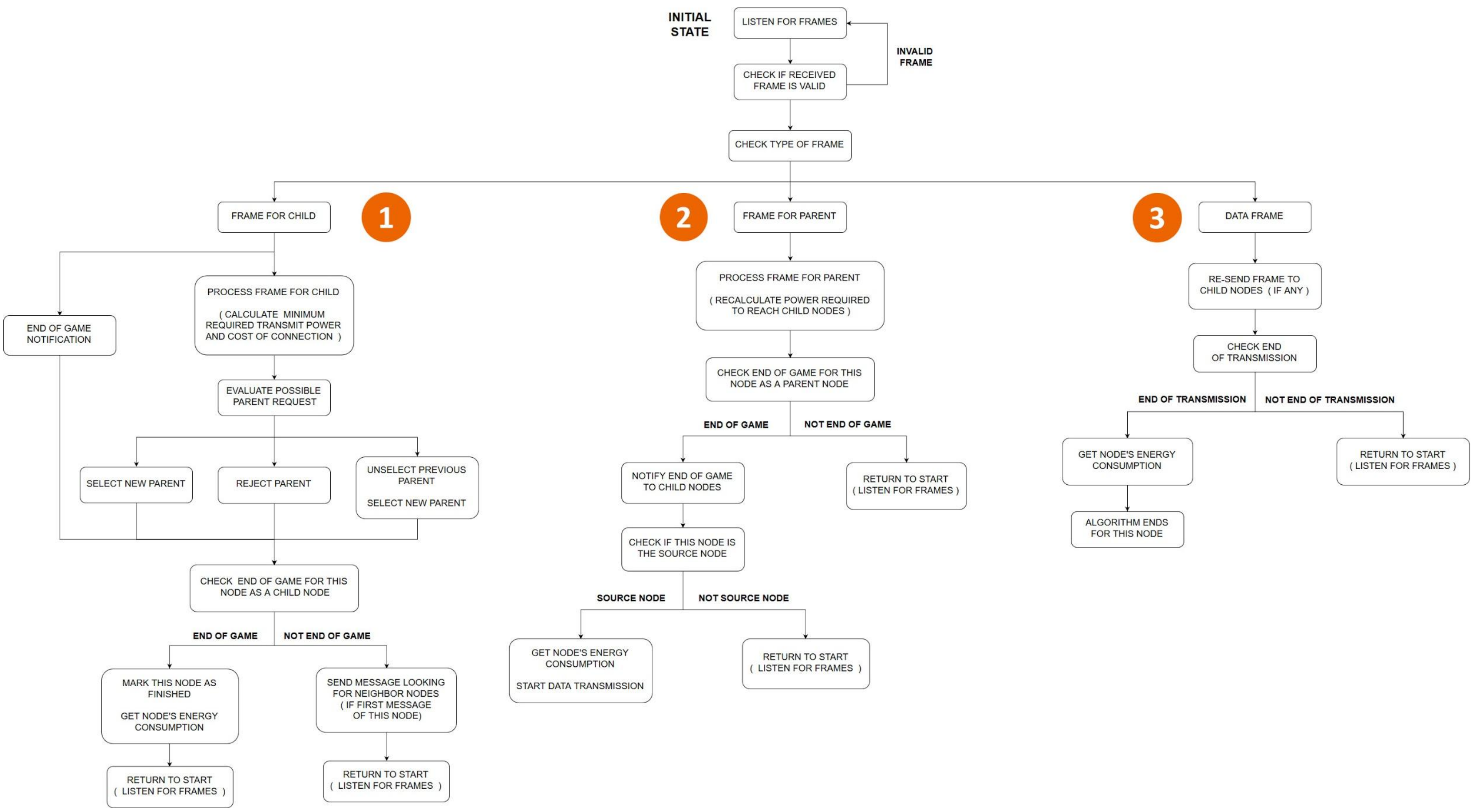
Phases of the protocol

1. Broadcast tree construction phase

2. Application data transmission phase



Practical algorithm design



Practical design

SECURE MOBILE NETWORKING

Generic frame

MAC Header	LLC Header	SNAP Extension	Broadcast Tree Protocol Header	CRC

Broadcast Tree Protocol header

Fixed header part							Optional header part		
4 (bytes)	2	2	2	2	2	2	32	Variable	Variable
Sequence number	Source flag	TTL	Frame type	Finished flag	Power flag	Route length	Power data	Route data	Application data

Practical design

Broadcast Tree Protocol header:

Fixed header part							Optional header part		
4 (bytes)	2	2	2	2	2	2	32	Variable	Variable
Sequence number	Source flag	TTL	Frame type	Finished flag	Power flag	Route length	Power data	Route data	Application data

Optional power data header:

8 (bytes)	8	8	8
Used Tx power	Highest Tx power	2^{nd} highest Tx power	Minimum sender SNR

Optional route data header:

6 (bytes)	6	6
MAC Address	…	MAC Address

Practical design

Types of frames

Neighbour discovery frame
Child request frame

Sequence number	Source flag	TTL	Frame type	Finished flag	Power flag	Route length	Power data	Route data

1

Local end game frame

Sequence number	Source flag	TTL	Frame type	Finished flag	Power flag	Route length

Parent confirmation frame
Parent rejection frame
Parent revocation frame

Sequence number	Source flag	TTL	Frame type	Power flag	Route length	Power data

2

Application data frame

Sequence number	Source flag	TTL	Frame type	Finished flag	Power flag	Route length	Application data

3

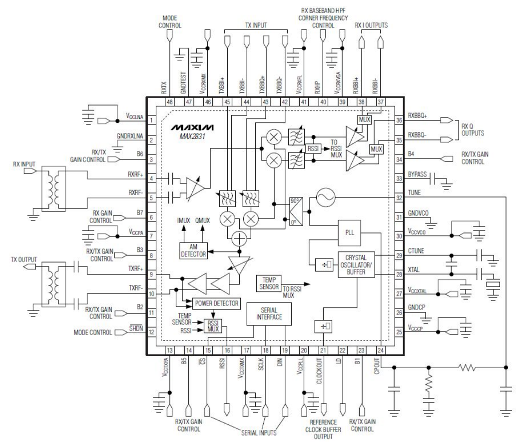
Technologies used

ns-3
NETWORK SIMULATOR

C++

Physical aspects of the model (The scenario)

The standard selection and additional configuration



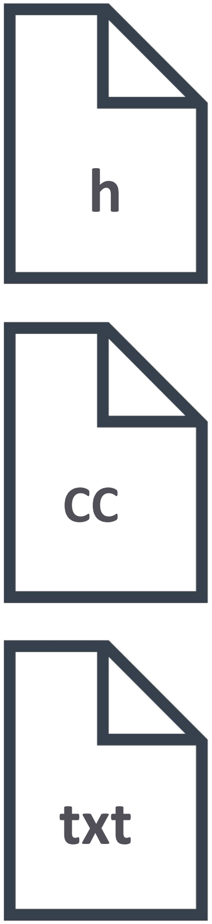
The radio energy model

The battery model

Practical implementation

Files and classes

- BroadcastTreeProtocolScenario.cc
- broadcast-tree-protocol.h/cc
- broadcast-tree-protocol-helper.h/cc
- broadcast-tree-protocol-header.h/cc
- broadcast-tree-protocol-point.h/cc
- rx-power-tag.h/cc & noise-tag.h/cc
- wscript.txt



BroadcastTreeProtocolScenario.cc

This file:

- Contains main function
- Defines the physical aspects of the simulation commented previously

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

  NS_LOG_UNCOND ("START MAIN FUNCTION");

  Packet::EnablePrinting ();

//VARIABLE INITIALIZATION-START-----------------------------------------------

  std::string phyMode ("ErpOfdmRate12Mbps");
  uint32_t packetSize = 1000;       // bytes
  uint32_t numPackets = 1000;
  uint32_t numNodes = 10;
  double interval = 0.01;                     // seconds
  const uint16_t protNumber = 0x0101;    // Our protocol Ethertype
  bool verbose = false;


//COMMAND LINE INITIALIZATION-START-------------------------------------------

  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
  cmd.AddValue ("numPackets", "number of packets generated", numPackets);
  cmd.AddValue ("interval", "interval (seconds) between packets", interval);
  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);

  cmd.Parse (argc, argv);


//NETWORK INFRASTRUCTURE SET-UP-----------------------------------------------

  // Convert to time object
  Time interPacketInterval = Seconds (interval);

  // disable fragmentation for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue ("2200"));
  // turn off RTS/CTS for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2200"));
  // Fix non-unicast data rate to be the same as that of unicast
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue (phyMode));

⋮
```

Practical implementation

broadcast-tree-protocol.h/cc

These files:

- **Contain BroadcastTreeProtocol class**
- **The algorithm defined in the explained diagram is defined here.**

```cpp

NS_OBJECT_ENSURE_REGISTERED (BroadcastTreeProtocol);

TypeId
BroadcastTreeProtocol::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::BroadcastTreeProtocol")
    .SetParent<Object> ()
    .SetGroupName ("Wifi")
    .AddConstructor<BroadcastTreeProtocol> ()
  ;
  return tid;
}

/********************* BROADCAST TREE PROTOCOL ************************/

BroadcastTreeProtocol::BroadcastTreeProtocol ()
{
  NS_LOG_UNCOND ( "BROADCAST TREE PROTOCOL CONSTRUCTOR" );

  m_parentEndOfGame = false;
  m_childEndOfGame = false;
  m_isSourceNode = false;
  m_finishedBranch = false;
  m_notFirstMessage = false;
  m_selectedParent = Mac48Address();

  m_powerToReachChilds = 0;
  m_iterationsUnchanged = 0;
  m_iterationsWithoutImproving = 0;
  m_iterationsWithoutNeighbor = 0;
  m_costOfCurrentConnection = -1.0;
  m_pktCount = 1000;

  m_sequenceNumbers = {0};
  m_neighbors = std::vector< Ptr<BroadcastTreeProtocolPoint> >();
  m_routeToSource = std::vector<Mac48Address>();
}


BroadcastTreeProtocol::~BroadcastTreeProtocol ()
{
```

⋮

Practical implementation

broadcast-tree-protocol-helper.h/cc

These files:

- **Contain BroadcastTreeProtocolHelper class**
- **Install our protocol on the nodes**

```
NS_LOG_COMPONENT_DEFINE ("BroadcastTreeProtocolHelper");

NS_OBJECT_ENSURE_REGISTERED (BroadcastTreeProtocolHelper);


TypeId
BroadcastTreeProtocolHelper::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::BroadcastTreeProtocolHelper")
    .SetParent<Object> ()
    .SetGroupName ("Wifi")
    .AddConstructor<BroadcastTreeProtocolHelper> ()
  ;
  return tid;
}


BroadcastTreeProtocolHelper::BroadcastTreeProtocolHelper ()
{
  NS_LOG_UNCOND ( "Created BroadcastTreeProtocolHelper" );
}

BroadcastTreeProtocolHelper::~BroadcastTreeProtocolHelper ()
{
}

void
BroadcastTreeProtocolHelper::DoDispose ()
{
}

void
BroadcastTreeProtocolHelper::Install (Ptr<Node> node, bool setAsSource)
{
  ObjectFactory factory;
  factory.SetTypeId ("ns3::BroadcastTreeProtocol");
  Ptr<BroadcastTreeProtocol> protocol = DynamicCast<BroadcastTreeProtocol>( factory.Create<Object>() );

  if ( setAsSource )
  {
    protocol->SetSourceNode();
```

⋮

Practical implementation

broadcast-tree-protocol-header.h/cc

These files:

- **Contain BroadcastTreeProtocolHeader class**
- **Define frames and methods for their modification as well as de/serialization**

```
NS_LOG_COMPONENT_DEFINE ("BroadcastTreeProtocolHeader");

NS_OBJECT_ENSURE_REGISTERED (BroadcastTreeProtocolHeader);

// static counter inizialization
uint32_t BroadcastTreeProtocolHeader::counter = 1;


BroadcastTreeProtocolHeader::BroadcastTreeProtocolHeader ()
{
  m_sequenceNumber = counter;
}


BroadcastTreeProtocolHeader::~BroadcastTreeProtocolHeader ()
{
}


void BroadcastTreeProtocolHeader::NewSequenceNumber (void)
{
  NS_LOG_FUNCTION (this);
  m_sequenceNumber = ++counter;
}

void
BroadcastTreeProtocolHeader::SetSequenceNumber (uint32_t sequenceNumber)
{
  m_sequenceNumber = sequenceNumber;
}

uint32_t
BroadcastTreeProtocolHeader::GetSequenceNumber (void)
{
  return m_sequenceNumber;
}

void
BroadcastTreeProtocolHeader::SetSourceFlag (uint16_t sourceFlag)
{
  m_sourceFlag = sourceFlag;
```

⋮

Practical implementation

broadcast-tree-protocol-point.h/cc

These files:

- **Contain BroadcastTreeProtocolPoint class**
- **Define a structure for storing the information of neighboring nodes**

```
NS_OBJECT_ENSURE_REGISTERED (BroadcastTreeProtocolPoint);

TypeId
BroadcastTreeProtocolPoint::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::BroadcastTreeProtocolPoint")
    .SetParent<Object> ()
    .SetGroupName ("Wifi")
    .AddConstructor<BroadcastTreeProtocolPoint> ()
  ;
  return tid;
}


BroadcastTreeProtocolPoint::BroadcastTreeProtocolPoint () : m_routeToSource(), m_address(),
                                                            , m_powerRequiredToReach(0)
                                                            , m_isFinished(false), m_isSourceNode(false)
                                                            , m_isChild(false)

{
}

double BroadcastTreeProtocolPoint::GetPowerRequiredToReach (void)
{
  return m_powerRequiredToReach;
}

void BroadcastTreeProtocolPoint::SetPowerRequiredToReach ( double powerRequiredToReach )
{
  m_powerRequiredToReach = powerRequiredToReach;
}

Mac48Address BroadcastTreeProtocolPoint::GetMacAddress (void)
{
  return m_address;
}

void BroadcastTreeProtocolPoint::SetMacAddress (Mac48Address address)
{
  m_address = address;
}
```

⋮

Practical implementation

rx-power-tag.h/cc & noise-tag.h/cc

These files:

- **Contain RxPowerTag and NoiseTag classes**
- **Are used for transferring physical layer information to our protocol**

```cpp
NS_OBJECT_ENSURE_REGISTERED (RxPowerTag);

TypeId
RxPowerTag::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::RxPowerTag")
    .SetParent<Tag> ()
    .SetGroupName ("Wifi")
    .AddConstructor<RxPowerTag> ()
    .AddAttribute ("RxPower", "The received signal power of the last packet received",
                   DoubleValue (0.0),
                   MakeDoubleAccessor (&RxPowerTag::Get),
                   MakeDoubleChecker<double> ())
  ;
  return tid;
}

TypeId
RxPowerTag::GetInstanceTypeId (void) const
{
  return GetTypeId ();
}

RxPowerTag::RxPowerTag () : m_rxPower (0)
{
}

RxPowerTag::RxPowerTag (double rxPower)
  : m_rxPower (rxPower)
{
}

uint32_t
RxPowerTag::GetSerializedSize (void) const
{
  return sizeof (double);
}

void
RxPowerTag::Serialize (TagBuffer i) const
{
```

⋮

Practical implementation

wscript.txt

This file:

- **Is used to build the wifi module of the ns-3 simulator (where our implementation lies)**

```python
def build(bld):
    obj = bld.create_ns3_module('wifi', ['network', 'propagation', 'energy', 'spectrum', 'antenna', 'mobility'])
    obj.source = [
        'model/wifi-information-element.cc',
        'model/wifi-information-element-vector.cc',
        'model/wifi-channel.cc',
        'model/wifi-mode.cc',
        'model/ssid.cc',
        'model/wifi-phy.cc',
        'model/wifi-phy-state-helper.cc',
        'model/error-rate-model.cc',
        'model/yans-error-rate-model.cc',
        'model/nist-error-rate-model.cc',
        'model/dsss-error-rate-model.cc',
        'model/interference-helper.cc',
        'model/yans-wifi-phy.cc',
        'model/yans-wifi-channel.cc',
        'model/spectrum-wifi-phy.cc',
        'model/wifi-phy-tag.cc',
        'model/wifi-spectrum-phy-interface.cc',
        'model/wifi-spectrum-signal-parameters.cc',
        'model/wifi-mac-header.cc',


        'model/broadcast-tree-protocol-header.cc',
        'model/broadcast-tree-protocol.cc',
        'model/broadcast-tree-protocol-point.cc',
        'model/rx-power-tag.cc',
        'model/noise-tag.cc',
        'model/wifi-utils.cc',


        'model/broadcast-tree-protocol-test.cc',


        'model/wifi-mac-trailer.cc',
        'model/mac-low.cc',
        'model/wifi-mac-queue.cc',
        'model/mac-tx-middle.cc',
        'model/mac-rx-middle.cc',
        'model/dca-txop.cc',
```

⋮

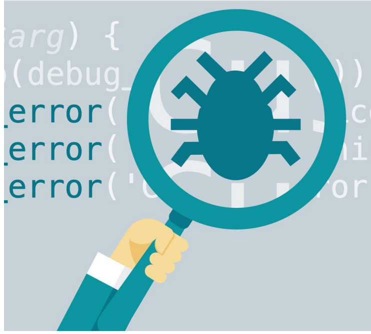
Evaluation issues

SECURE MOBILE NETWORKING

Due to memory related issues the acquisition of simulation data has not been possible yet.

<u>What has been done to solve the issue:</u>

1. Using GDB
2. Opening a discussion on the ns-3 forum
3. Using Valgrind
4. Creating a reduced version of the protocol



Further possibilities/Future work

SECURE MOBILE NETWORKING

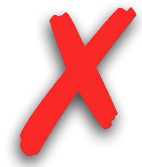
1. Solving the simulation issues

2. Modifying some aspects of the protocol
   - Rank based parent validation
   - End of game simplification
   - Delay dependent on cost

3. Implementing the protocol in a testbed

Conclusions

Impossibility of acquiring simulation results



However...

- Discussion of state of the current solutions
- Transcription of the original theoretical model into a practical design
- Implementation code has been completed

You deserve it

Thank you

Questions

SECURE MOBILE NETWORKING

Questions

?

Doubts

Comments

Contact

Prof. Dr.-Ing. Matthias Hollick
matthias.hollick@seemoo.tu-darmstadt.de

Technische Universität Darmstadt
Secure Mobile Networking Lab – SEEMOO
Department of Computer Science
Mornewegstr. 32
D-64293 Darmstadt

Phone: +49 6151 16-25472
Fax: +49 6151 16-25471
Web: https://seemoo.de

Copyright Notice