[How To] Implement a New Unicast Routing Protocol of MANETs* in NS-2.34

Hi Folks,

In this post I would tell u about how to successfully perform the steps to Implement a New Unicast Routing Protocol for MANETs by F. J. Ros and P. M. Ruiz While performing this task I faced a no. of problems. The tutorial by F. J. Ros and P. M. Ruiz is complete for a beginner to start working with. However it has certain anomalies (not in the code) such as typographical errors or case sensitivity problems. The tutorial is also designed for earlier versions of ns2. So I tried working on this tutorial which is quite excellent and and after facing a lot of problems and issues I successfully implemented it in ns-2.34.

If u perform the steps according to this post. U shall be able to implement the protocol succesfully. I shall not go into explaining the source code as it has already been beautifully explained in the original document. The source of the document is given at the end of this post. Thus, for successfull implementation u should read this post very carefully.

NOTE: Beware of Case Sensitiveness and perform as I shall tell below.

Step 1) First of all create a directory in your ns-2.34 directory with the following files.

protoname_pkt.h
protoname_rtable.cc
protoname_rtable.h
Protoname.cc \\ Note the capital P in the beginning (it should be capital P)
protoname.h

Step 2) The code for protoname_pkt.h is given below (u can copy paste)

```
#ifndef PROTONAME_PKT_H_
#define PROTONAME_PKT_H_

#include <packet.h>
#include <config.h>
#include <random.h>
#include <timer-handler.h>
#include <trace.h>
#include <classifier-port.h>

#define HDR_PROTONAME_PKT(p) hdr_protoname_pkt::access(p)

struct hdr_protoname_pkt {

      nsaddr_t pkt_src_;         //Node which originated this packet
      u_int16_t pkt_len_;    //Packet length (in bytes)
      u_int8_t pkt_seq_num_; //Packet Sequence Number

      inline nsaddr_t& pkt_src() {return pkt_src_;}
      inline u_int16_t& pkt_len() {return pkt_len_;}
      inline u_int8_t& pkt_seq_num() {return pkt_seq_num_;}

      static int offset_;
      inline static int& offset() {return offset_;}

      // To access the hdr_protoname_pkt packet header in a packet pointed by p,
we use the access fn
      inline static hdr_protoname_pkt* access(const Packet* p) {
```

```
            return (hdr_protoname_pkt*)p->access(offset_);
      }
};

#endif /* PROTONAME_PKT_H_ */
```

Step 3) The code for protoname_rtable.cc is given below

```
#include "protoname_rtable.h"
#include "ip.h"

protoname_rtable::protoname_rtable() { }

void protoname_rtable::print(Trace* out) {
      sprintf(out->pt_->buffer(), "p\tdest\tnext");
      out->pt_->dump();
      for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
            sprintf(out->pt_->buffer(), "P\t%d\t%d", (*it).first, (*it).second);
            out->pt_->dump();
      }
}

void
protoname_rtable::clear() {
      rt_.clear();
}

void
protoname_rtable::rm_entry(nsaddr_t dest) {
      rt_.erase(dest);
}

void
protoname_rtable::add_entry(nsaddr_t dest, nsaddr_t next) {
      rt_[dest] = next;
}

nsaddr_t
protoname_rtable::lookup(nsaddr_t dest) {
      rtable_t::iterator it = rt_.find(dest);
      if (it == rt_.end())
            return IP_BROADCAST;
      else
            return (*it).second;
}

u_int32_t
protoname_rtable::size() {
      return rt_.size();
}
```

Step 4) The code for protoname_rtable.h is given below

```
#ifndef PROTONAME_RTABLE_H_
#define PROTONAME_RTABLE_H_

#include <trace.h>
#include <map>
#include <ip.h>

typedef std::map<nsaddr_t, nsaddr_t> rtable_t;
```

```
class protoname_rtable {
      rtable_t rt_;

public:
      protoname_rtable();
      void print(Trace*);
      void clear();
      void rm_entry(nsaddr_t);
      void add_entry(nsaddr_t, nsaddr_t);
      nsaddr_t lookup(nsaddr_t);
      u_int32_t size();
};

#endif /* PROTONAME_RTABLE_H_ */
```

Step 5) The code for Protoname.cc is given below

```
#include "protoname.h"
#include "protoname_pkt.h"
#include "protoname_rtable.h"
#include <timer-handler.h>
#include <node.h>
#include <random.h>
#include <cmu-trace.h>
#include <iostream>
#include <classifier-port.h>
#include <packet.h>
#include <address.h>

// To Bind our packet in OTcl Interface
int hdr_protoname_pkt::offset_;
static class ProtonameHeaderClass : public PacketHeaderClass {
public:
      ProtonameHeaderClass() : PacketHeaderClass("PacketHeader/PROTONAME",
sizeof(hdr_protoname_pkt)) {
            bind_offset(&hdr_protoname_pkt::offset_);
      }
}class_rtProtoProtoname_hdr;

static class ProtonameClass : public TclClass
{
public:
      ProtonameClass() : TclClass("Agent/PROTONAME") {}
      TclObject* create(int argc, const char*const* argv) {
            assert(argc==5);
            return(new
Protoname((nsaddr_t)Address::instance().str2addr(argv[4])));
      }
}class_rtProtoProtoname;

void
Protoname_PktTimer::expire(Event* e) {
      agent_->send_protoname_pkt();
      agent_->reset_protoname_pkt_timer();
}

Protoname::Protoname(nsaddr_t id) : Agent(PT_PROTONAME), pkt_timer_(this) {
      bind_bool("accessible_var_", &accessible_var_);
      ra_addr_ = id;
}
```

```
int
Protoname::command(int argc, const char*const* argv) {
      if(argc == 2) {
            if (strcasecmp(argv[1], "start") == 0) {
                  pkt_timer_.resched(0.0);
                  return TCL_OK;
            }
            else if (strcasecmp(argv[1],"print_rtable") == 0) {
            if(logtarget_ != 0) {
                  sprintf(logtarget_->pt_->buffer(), "P %f _%d_ Routing Table",
CURRENT_TIME, ra_addr());
                  logtarget_->pt_->dump();
                  rtable_.print(logtarget_);
            }
            else {
                  fprintf(stdout, "%f _%d_ If you want to print this routing
table ""you must create a trace file in your TCL Script",CURRENT_TIME,
ra_addr());
            }
            return TCL_OK;
            }
      }
      else if (argc == 3) {
            //Obtains corresponding dmux to carry packets to upper layer
            if (strcmp(argv[1], "port-dmux") == 0) {
                  dmux_ = (PortClassifier*)TclObject::lookup(argv[2]);
                  if (dmux_ == 0) {
                        fprintf(stderr, "%s: %s lookup of %s failed \n",
__FILE__,argv[1],argv[2]);
                        return TCL_ERROR;
                  }
                  return TCL_OK;
            }
            //Obtains corresponding tracer
            else if (strcmp(argv[1], "log-target") == 0 || strcmp(argv[1],
"tracetarget") == 0) {
                  logtarget_ = (Trace*)TclObject::lookup(argv[2]);
                  if (logtarget_ == 0)
                        return TCL_ERROR;
                  return TCL_OK;
            }
      }
      //Pass the command to the base class
      return Agent::command(argc, argv);
}

void
Protoname::recv(Packet* p, Handler* h) {
      struct hdr_cmn* ch      = HDR_CMN(p);
      struct hdr_ip* ih       = HDR_IP(p);

      if (ih->saddr() == ra_addr()){
            //If there exists a routing loop, drop the packet
            if(ch->num_forwards() > 0) {
                  drop(p, DROP_RTR_ROUTE_LOOP);
                  return;
            }

            //else if this is a packet I am originating, must add IP header
            else if(ch->num_forwards() == 0)
                  ch->size() += IP_HDR_LEN;
```

```
	}

	// If it is a protoname packet, must process it
	if(ch->ptype() == PT_PROTONAME)
		recv_protoname_pkt(p);

	//Otherwise, must forward the packet (unless TTL reaches zero
	else {
		ih->ttl_--;
		if(ih->ttl_ == 0) {
			drop(p, DROP_RTR_TTL);
			return;
		}
		forward_data(p);
	}
}

void
Protoname::recv_protoname_pkt(Packet* p) {
	struct hdr_ip* ih		= HDR_IP(p);
	struct hdr_protoname_pkt* ph  = HDR_PROTONAME_PKT(p);

	// All routing messages are sent from and to port RT_PORT, so we shall
check it
	assert(ih->sport() == RT_PORT);
	assert(ih->dport() == RT_PORT);

	/* processing of protoname packet */

	// Release resources
	Packet::free(p);
}

void
Protoname::send_protoname_pkt() {
	Packet* p				= allocpkt();
	struct hdr_cmn* ch			= HDR_CMN(p);
	struct hdr_ip* ih			= HDR_IP(p);
	struct hdr_protoname_pkt* ph  = HDR_PROTONAME_PKT(p);

	ph->pkt_src()				= ra_addr();
	ph->pkt_len()				= 7;
	ph->pkt_seq_num()			= seq_num_++;

	ch->ptype()				= PT_PROTONAME;
	ch->direction()			= hdr_cmn::DOWN;
	ch->size()				= IP_HDR_LEN + ph->pkt_len();
	ch->error()				= 0;
	ch->next_hop()			= IP_BROADCAST;
	ch->addr_type()			= NS_AF_INET;

	ih->saddr()				= ra_addr();
	ih->daddr()				= IP_BROADCAST;
	ih->sport()				= RT_PORT;
	ih->dport()				= RT_PORT;
	ih->ttl()				= IP_DEF_TTL;

	Scheduler::instance().schedule(target_, p, JITTER);
}

void
Protoname::reset_protoname_pkt_timer() {
```

```
	pkt_timer_.resched((double)5.0);
}

void
Protoname::forward_data(Packet* p) {
	struct hdr_cmn* ch = HDR_CMN(p);
	struct hdr_ip* ih = HDR_IP(p);

	if(ch->direction() == hdr_cmn::UP && ((u_int32_t)ih->daddr() ==
IP_BROADCAST || ih->daddr() == ra_addr())) {
		dmux_->recv(p, NULL);
		return;
	}
	else {
		ch->direction() = hdr_cmn::DOWN;
		ch->addr_type() = NS_AF_INET;
		if ((u_int32_t)ih->daddr() == IP_BROADCAST)
			ch->next_hop() = IP_BROADCAST;
		else {
			nsaddr_t next_hop = rtable_.lookup(ih->daddr());
			if(next_hop == IP_BROADCAST) {
				debug("%f: Agent %d can not forward a packet destined to
%d \n", CURRENT_TIME,ra_addr(),ih->daddr());
				drop(p, DROP_RTR_NO_ROUTE);
				return;
			}
			else
				ch->next_hop() =  next_hop;
		}
		Scheduler::instance().schedule(target_, p, 0.0);
	}
}
```

Step 6) The code for protoname.h is given below

```
#ifndef PROTONAME_H_
#define PROTONAME_H_

#include "protoname_pkt.h"
#include "protoname_rtable.h"
#include <agent.h>
#include <packet.h>
#include <trace.h>
#include <timer-handler.h>
#include <random.h>
#include <classifier-port.h>
#include "arp.h"
#include "ll.h"
#include "mac.h"
#include "ip.h"
#include "delay.h"

#define CURRENT_TIME Scheduler::instance().clock()
#define JITTER (Random::uniform()*0.5)

class Protoname;  //Forward Declaration

/* TIMERS */

class Protoname_PktTimer : public TimerHandler {
public :
```

```
      Protoname_PktTimer(Protoname* agent) : TimerHandler() {
      agent_ = agent;
      }
protected:
      Protoname* agent_;
      virtual void expire(Event* e);
};

/* Agent */
class Protoname : public Agent {
      /* Friends */
      friend class Protoname_PktTimer;

      /*Private Members*/
      nsaddr_t               ra_addr_;
      //protoname_state      state_;
      protoname_rtable  rtable_;
      int                    accessible_var_;
      u_int8_t               seq_num_;

protected :

      PortClassifier*   dmux_;                 //For Passing Packets Up To
Agents
      Trace*                      logtarget_;       //For Logging
      Protoname_PktTimer     pkt_timer_;       //Timer for sending packets

      inline nsaddr_t&       ra_addr()              {return ra_addr_; }
      //inline protoname_state& state()                    {return state_;}
      inline int&                  accessible_var()  {return accessible_var_;}

      void forward_data(Packet*);
      void recv_protoname_pkt(Packet*);
      void send_protoname_pkt();

      void reset_protoname_pkt_timer();

public:
      Protoname(nsaddr_t);
      int command(int, const char*const*);
      void recv(Packet*, Handler*);

};

#endif /* PROTONAME_H_ */
```

**Note: Now we have to make changes to certain ns-2.34 files. The steps given below are most important for successful implementation so perform them very carefully.**

Step 7) Open file packet.h under ns-2.34/common/packet.h. Do the changes given below.

Under `typedef unsigned int packet_t;` lots of constants would have been defined.

Go down and at Line 182 u will find

```
      // AOMDV packet
static const packet_t PT_AOMDV = 61;
```

After it insert the line as follows

```
// insert new packet types here
static const packet_t   PT_PROTONAME = 62;
```

and change PT_NTYPE value from 62 (earlier) to 63 now.

```
static packet_t PT_NTYPE = 63; // This MUST be the LAST one
```

Then in the same file move down till u find **p_info**(). In it search for the function
`static packetClass classify(packet_t type) {`

and perform the following changes in it by entering the 2nd line below.

```
type == PT_AODV ||
type == PT_PROTONAME)
return ROUTING;
```

In the same file, go below and search for the function `static void initName().` Now at the end of this function u shall find this

```
// AOMDV patch
            name_[PT_AOMDV]= "AOMDV";
```

After this enter the following line.

```
name_[PT_PROTONAME] = "PROTONAME";

name_[PT_NTYPE]= "undefined"; // let it remain as it is
```

Step 8) Open the file cmu-trace.h under ns-2.34/trace/cmu-trace.h

At line 162 u shall find

```
void    format_aomdv(Packet *p, int offset);
```

After it enter the following line

```
void format_protoname(Packet *p, int offset);
```

Step 9) Now open the file cmu-trace.cc in the same folder ns-2.34/trace/cmu-trace.h

Include the following header file at the beginning.

```
#include <protoname/protoname_pkt.h>
```

Now at the end write the following code ( u can always copy-paste) :)

```
void
CMUTrace::format_protoname(Packet* p, int offset)
{
      struct hdr_protoname_pkt* ph = HDR_PROTONAME_PKT(p);

      if(pt_->tagged()) {
            sprintf(pt_->buffer() + offset, "-PROTONAME:o %d -PROTONAME:s %d
-PROTONAME:l %d", ph->pkt_src(), ph->pkt_seq_num(), ph->pkt_len());
      }
      else if (newtrace_) {
            sprintf(pt_->buffer() + offset, "-P PROTONAME -Po %d -Ps %d -Pl %d
", ph->pkt_src(), ph->pkt_seq_num(), ph->pkt_len());
```

```
	}
	else {
		sprintf(pt_->buffer() + offset, "[PROTONAME %d %d %d] ", ph-
>pkt_src(), ph->pkt_seq_num(), ph->pkt_len());
	}
}
```

After the above steps search for the function (in the same file) at Line 1305 (most probably)

```
void CMUTrace::format(Packet* p, const char *why)
```

move down and after

```
case PT_PING:
	break;
```

```
Enter the following lines
```

```
case PT_PROTONAME:
	format_protoname(p, offset);
	break;
```

Step 10) Open the file ns-packet.tcl under ns-2.34/tcl/lib/ns-packet.tcl

Search at Line 113 for
```
foreach prot {
```

```
Enter the following after to it to make it look like this
```

```
foreach prot {
	PROTONAME
# Common:
	Common
	Flags
	IP	# IP
# Routing Protocols:
	NV	# NixVector classifier for stateless routing
....
```

Step 11) Open the file ns-default.tcl under ns-2.34/tcl/lib/ns-default.tcl

In the end of the file enter the following line

```
Agent/PROTONAME set accessible_var_ true
```

Step 12) Open the file ns-lib.tcl under ns-2.34/tcl/lib/ns-lib.tcl

Search at Line 604 for the function
```
Simulator instproc create-wireless-node args {
```

Go below and after

```
switch -exact $routingAgent_ {
```

Enter the following lines

```
PROTONAME {
    set ragent [$self create-protoname-agent $node]
```

```
}
```
To make it look like this
```
switch -exact $routingAgent_ {
                PROTONAME {
                set ragent [$self create-protoname-agent $node]
                }
                DSDV {
                      set ragent [$self create-dsdv-agent $node]
                }
```

Now in the same file go at last or end and enter the following lines

```
Simulator instproc create-protoname-agent { node } {
            # Create Protoname Routing Agent
            set ragent [new Agent/PROTONAME [$node node-addr]]
            $self at 0.0 "$ragent start"
            $node set ragent_ $ragent
            return $ragent
}
```

Step 13) Open file priqueue.cc under ns-2.34/queue/priqueue.cc

Search for the function at Line 82.

```
PriQueue::recv(Packet *p, Handler *h)
```

After `case PT_AOMDV: enter the following line`

```
case PT_PROTONAME:
```

```
It should look like this
```

```
case PT_AOMDV:
            case PT_PROTONAME:
                  recvHighPriority(p, h);
                        break;
```

Step 14) Now changes to be done in Makefile.in

Under INCLUDES add the following directory

```
-I./wpan \
-I./protoname
```

Under OBJ_CC do the following changes at the end.

```
protoname/Protoname.o protoname/protoname_rtable.o \
@V_STLOBJ@
```

```
Step 15) Now save your project and move to the command line Ctrl + Alt + T.
```

```
Browse to directory ns-2.34 under ns-allinone-2.34 directory.
```

```
Perform the following commands in order.
```

```
./configure
```

```
make clean
```

```
make depend // Optional
```

```
make

sudo make install
```

U shall get no errors. If you are getting any errors ask in the comments section.

Step 16) Now we have to make a tcl file to test the protocol.

In home directory. Create a tcl file with the name exproto.tcl. The contents of the tcl file are as follows (I have defined 6 nodes)

```
# Define options
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) CMUPriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 6 ;# number of mobilenodes
set val(rp) PROTONAME ;# routing protocol
set val(x) 1000 ;# X dimension of topography
set val(y) 1000 ;# Y dimension of topography
set val(stop) 150 ;# time of simulation end

set ns [new Simulator]
set tracefd [open simple.tr w]
set namtrace [open simwrls.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

# configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set n($i) [$ns node]
}
```

```
# Provide initial location of mobilenodes
$n(0) set X_ 100.0
$n(0) set Y_ 200.0
$n(0) set Z_ 0.0

$n(1) set X_ 200.0
$n(1) set Y_ 400.0
$n(1) set Z_ 0.0

$n(2) set X_ 500.0
$n(2) set Y_ 600.0
$n(2) set Z_ 0.0

$n(3) set X_ 400.0
$n(3) set Y_ 500.0
$n(3) set Z_ 0.0

$n(4) set X_ 700.0
$n(4) set Y_ 400.0
$n(4) set Z_ 0.0

$n(5) set X_ 500.0
$n(5) set Y_ 800.0
$n(5) set Z_ 0.0



# Set a TCP connection between n(1) and n(3)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(1) $tcp
$ns attach-agent $n(3) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

# Set a TCP connection between n(2) and n(4)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(2) $tcp
$ns attach-agent $n(4) $sink
$ns connect $tcp $sink

#defining heads
$ns at 0.0 "$n(0) label CH"
$ns at 0.0 "$n(1) label Source"
#$ns at 0.0 "$n(2) label N2"



$ns at 10.0 "$n(2) setdest 785.0 228.0 5.0"
$ns at 13.0 "$n(4) setdest 700.0 20.0 5.0"
$ns at 15.0 "$n(3) setdest 115.0 85.0 5.0"

#Color change while moving from one group to another
$ns at 73.0 "$n(2) delete-mark N2"
$ns at 73.0 "$n(2) add-mark N2 pink circle"
$ns at 124.0 "$n(1) delete-mark N11"
$ns at 124.0 "$n(1) add-mark N11 purple circle"
```

```
$ns at 103.0 "$n(5) delete-mark N5"
$ns at 103.0 "$n(5) add-mark N5 white circle"
$ns at 87.0 "$n(3) delete-mark N26"
$ns at 87.0 "$n(3) add-mark N26 yellow circle"
$ns at 92.0 "$n(0) delete-mark N14"
$ns at 92.0 "$n(0) add-mark N14 green circle"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 20 defines the node size for nam
$ns initial_node_pos $n($i) 20
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$n($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
global ns tracefd namtrace
$ns flush-trace
close $tracefd
close $namtrace
exec nam simwrls.nam &
}

$ns run
```

Save and close it.

Step 17) Now open terminal and type ns exproto.tcl

It shall run perfectly and nam should open.

The trace file should look as follows.

```
s 0.007102330 _0_ RTR  --- 0 PROTONAME 27 [0 0 0 0] ------- [0:255 -1:255 32 0]
[PROTONAME 0 0 7]
r 0.008123076 _1_ RTR  --- 0 PROTONAME 27 [0 ffffffff 0 800] ------- [0:255
-1:255 32 0] [PROTONAME 0 0 7]
s 0.044061336 _1_ RTR  --- 1 PROTONAME 27 [0 0 0 0] ------- [1:255 -1:255 32 0]
[PROTONAME 1 0 7]
r 0.045102081 _0_ RTR  --- 1 PROTONAME 27 [0 ffffffff 1 800] ------- [1:255
-1:255 32 0] [PROTONAME 1 0 7]
r 0.045102081 _3_ RTR  --- 1 PROTONAME 27 [0 ffffffff 1 800] ------- [1:255
-1:255 32 0] [PROTONAME 1 0 7]
...........
...........
```

Step 18) Now u have successfully implemented the unicast routing protocol for MANETs. U should not face any problems if you follow the above steps. However I have used Eclipse Galileo for the developing environment and prefer u to also use the same. Search my blog for configuring ns-2.34 with eclipse.

* Source – Implementing a New Manet Unicast Routing Protocol in NS2 by Francisco J. Ros, and Pedro M. Ruiz <download tutorial>