MANET Implementation and Test

KRnet2004 Workshop

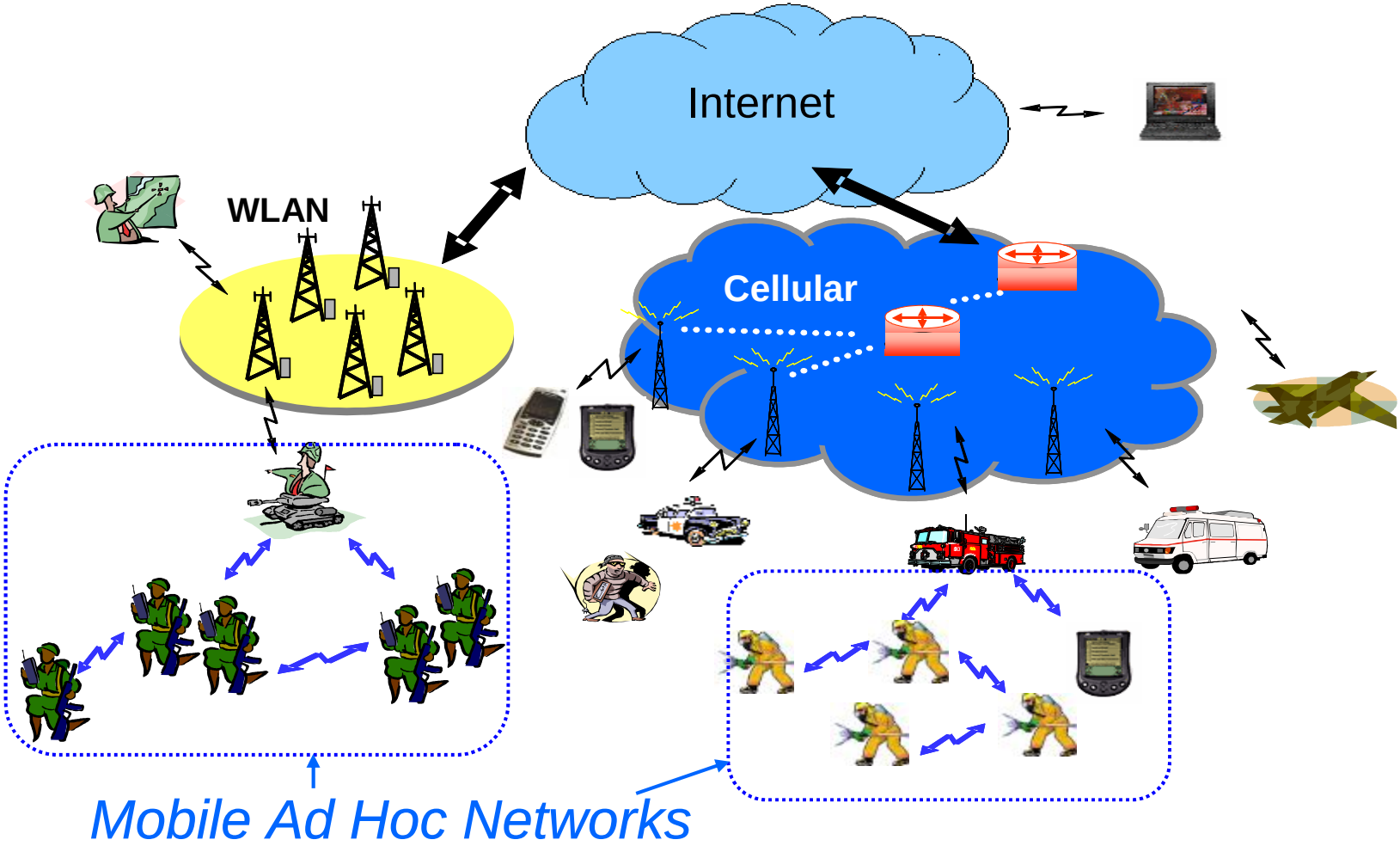
Jaehoon Jeong, ETRI
paul@etri.re.kr
http://www.adhoc.6ants.net/~paul

Contents

Introduction - 1/3

- Wireless Networks
  - Infrastructured Network
    - Cellular Network (3GPP or 3GPP2)
    - Wireless LAN (IEEE 802.11)
  - Infrastructureless Network
    - Ad Hoc Network
- Ad Hoc Network
  - Definition
    - Temporary network composed of mobile nodes without preexisting communication infrastructure, such as Access Point (AP) and Base Station (BS).
      - Each node plays the role of router for multi-hop routing.
  - Application
    - War-field communication, Emergency recovery, Sensor networking, Home-networking, Video-conferencing etc.

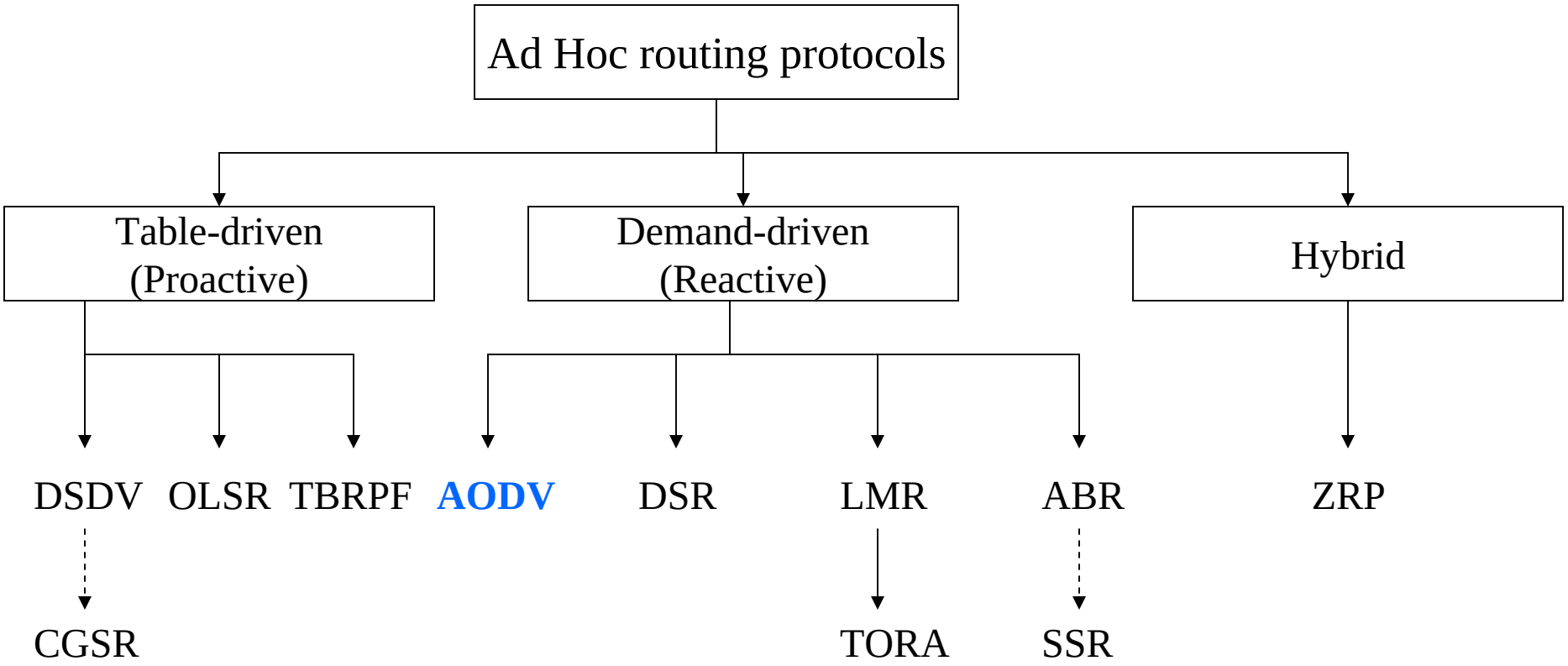
Wireless Networks



Introduction – 2/3

- Issues in MANET
  - Ad Hoc Unicast Routing
  - Ad Hoc Multicast/Broadcast Routing
  - Power Saving
  - Global Connectivity for MANET
  - Addressing & DNS Service
  - Automatic Support of Networking in MANET
    - MANET Autoconfiguration

Ad Hoc Unicast Routing Protocols

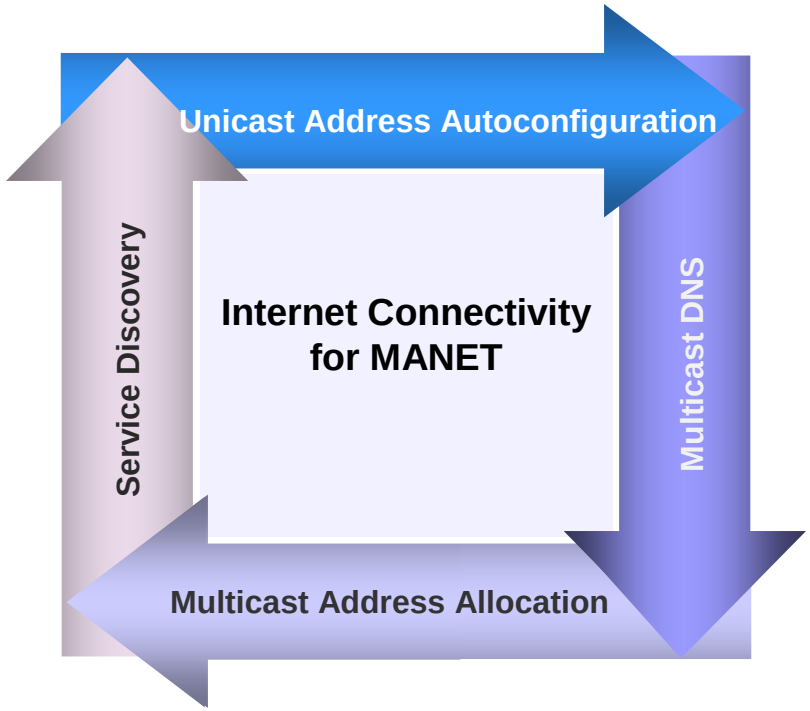


Introduction – 3/3

- MANET Autoconfiguration
  - MANET has dynamically changing network topology.
    - MANET partition and mergence may happen.
      - In MANET, there are many points to consider unlike the Internet.
  - There is no network administrator.
    - The current Internet services, such as address autoconfigation and DNS, are difficult to adopt.
    - Autoconfiguration is necessary in MANET.

MANET Auto-Configuration

- Unicast Address Autoconfiguration
- Multicast Address Allocation
- Multicast DNS
- Service Discovery
- Internet Connectivity



MANET Routing Protocol :
Ad Hoc On-Demand Distance Vector (AODV) Routing

C. Perkins, E. Belding-Royer and S. Das,
RFC 3561,
July 2003.

Contents

- Overview of AODV
- Route Discovery
- Route Maintenance
- Optimizations
- AODV Message Formats
- IPv6 AODV

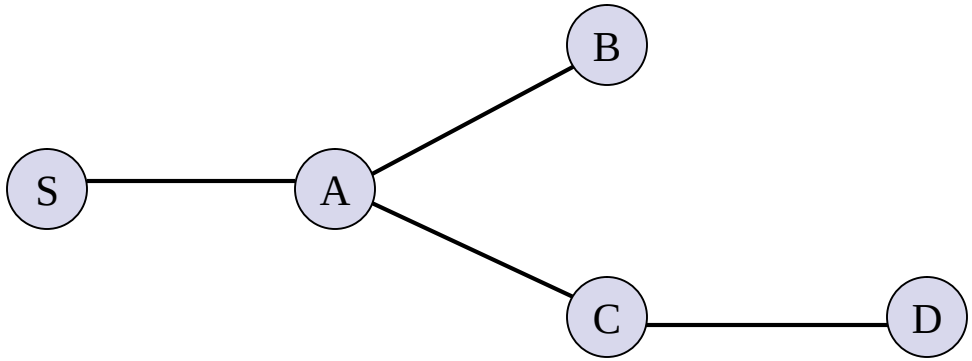
Overview of AODV - 1/2

- AODV is **improved DSDV algorithm**
  - Reactive or On-demand
  - Provides unicast and multicast communication (MAODV)
  - AODV doesn’t maintain a complete list of routes as in DSDV algorithm.
    - Nodes that are not on a selected path don’t maintain routing information or participate in routing table exchanges.
  - Loop-free
    - AODV utilizes **destination sequence numbers** to ensure all routes are loop-free.

Overview of AODV - 2/2

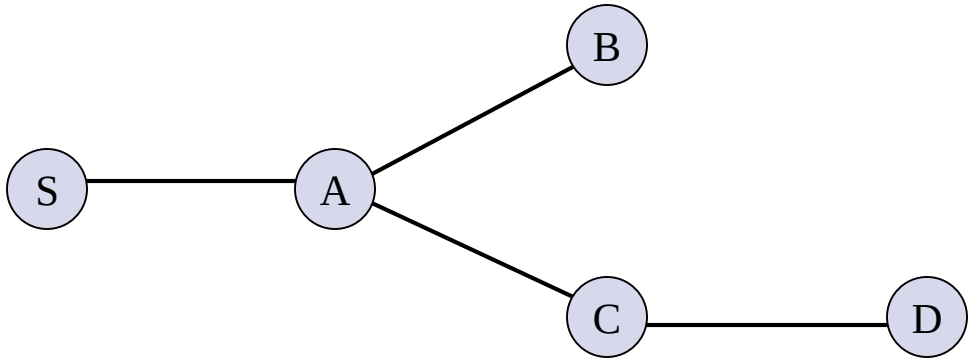
- Route Discovery process
  - When does it happen?
    - When source node doesn't have **a valid route to a destination** yet.
  - Route Discovery
    - It broadcasts a **route request (RREQ)** packet to its neighbors.
    - Neighbors forward the request to their neighbors, and so on until either the **destination** or an **intermediate node with a "fresh enough" route to the destination** is located.

Route Discovery - 1/10



1. Node S needs a route to D in order to send data packets to D

Route Discovery - 2/10



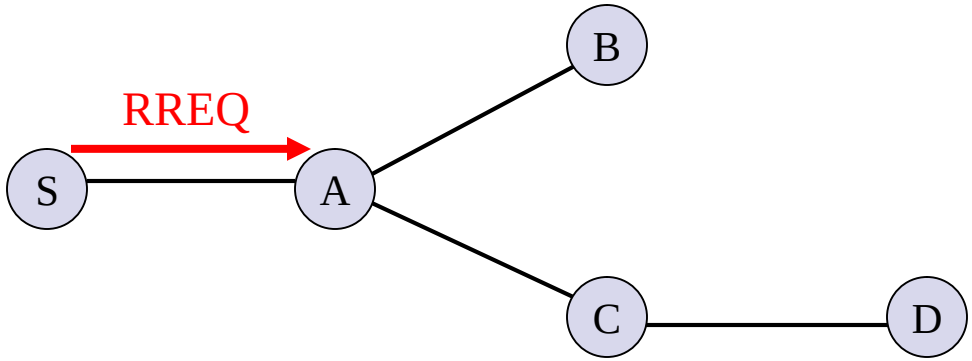
1. Node S needs a route to D in order to send data packet to D
2. Creates a Route Request (RREQ)

   Enters D's IP addr, seq#,

   S's IP addr, seq#

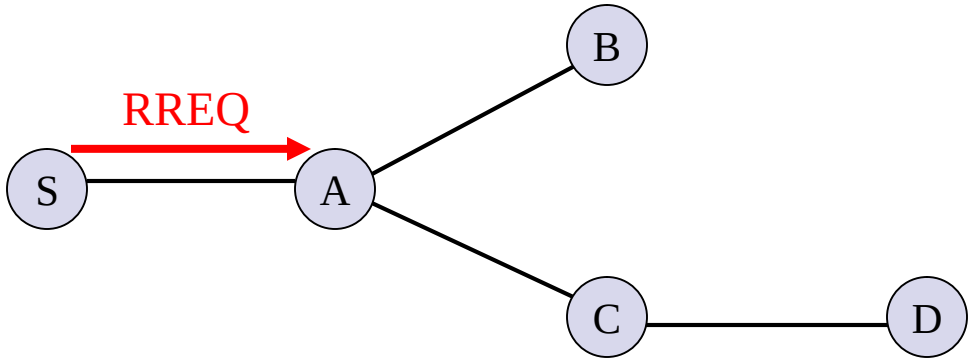
   hopcount (=0)

Route Discovery - 3/10



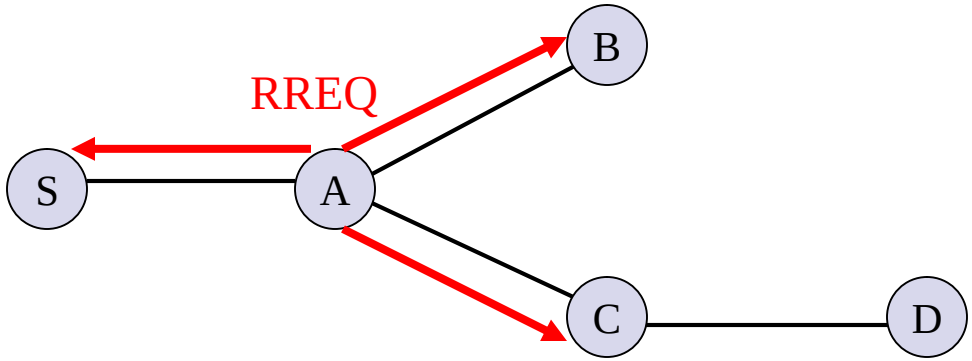
2. Creates a Route Request (RREQ)
   Enters D's IP addr, seq#,
   S's IP addr, seq#
   hopcount (=0)
3. Node S broadcasts RREQ to neighbors

Route Discovery - 4/10



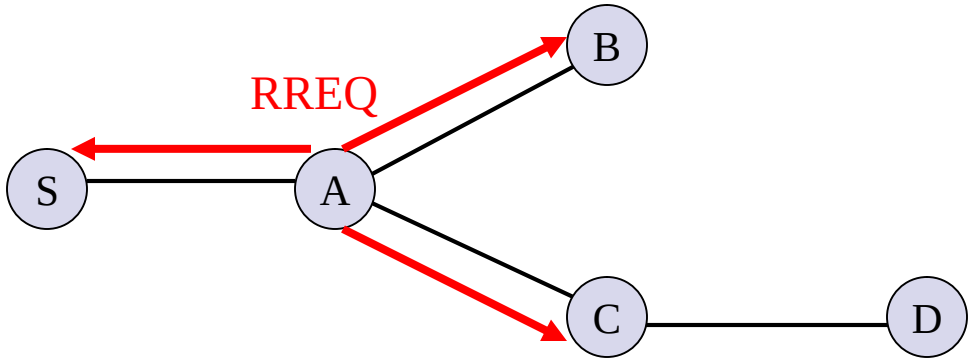
4. Node A receives RREQ
   - Makes reverse route entry for S
     dest = S, nexthop = S, hopcnt = 1

Route Discovery - 5/10



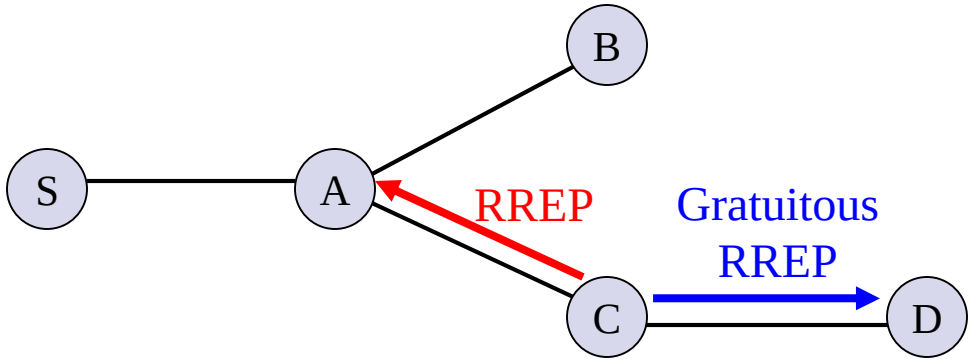
4. Node A receives RREQ
   - Makes reverse route entry for S
     dest = S, nexthop = S, hopcnt = 1
   - It has no route to D, so it rebroadcasts RREQ

Route Discovery - 6/10



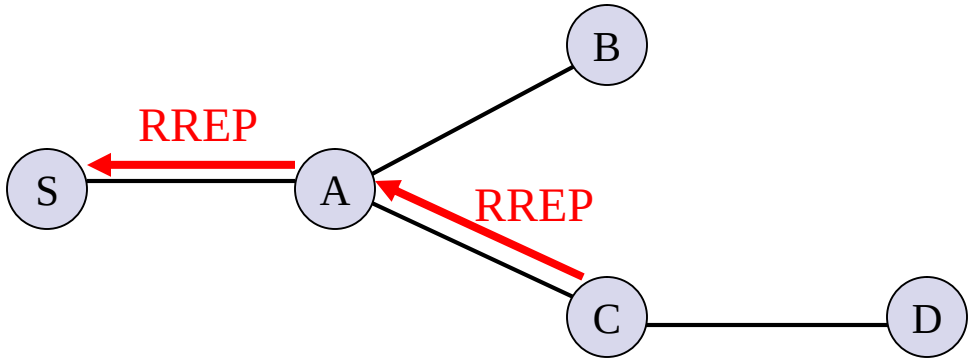
5. Node C receives RREQ
   - Makes reverse route entry for S
     dest = S, nexthop = A, hopcnt = 2
   - It has a route to D, and
     the seq# for route for D is >=D's seq# in RREQ

Route Discovery - 7/10



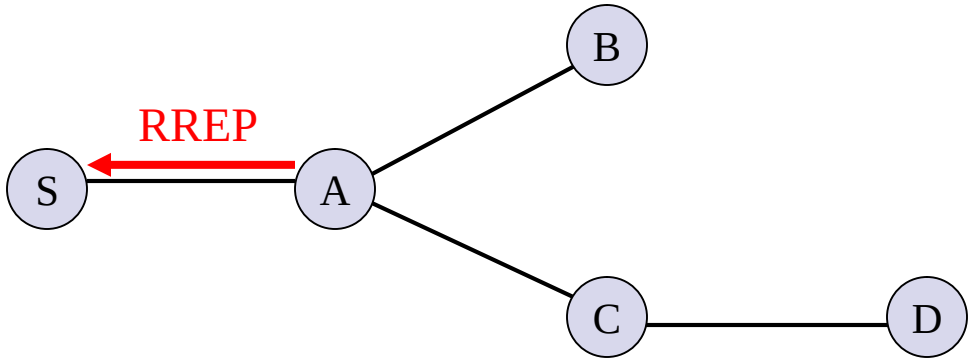
6. Node C sends RREP
   - C creates a Route Reply (RREP)
     Enters D's IP addr, seq #
     S's IP addr, hopcount to D (=1)
     Lifetime
   - Unicasts RREP towards A
   - C can allow D to set up a path towards S with **Gratuitous RREP**.

Route Discovery - 8/10



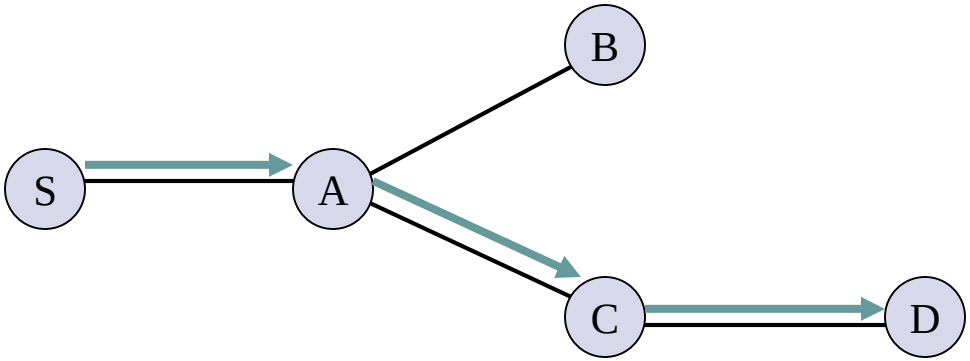
7. Node A receives RREP
   - Makes forward route entry to D
     dest = D, nexthop = C, hopcount = 2, Lifetime
   - Unicasts RREP to S

Route Discovery - 9/10



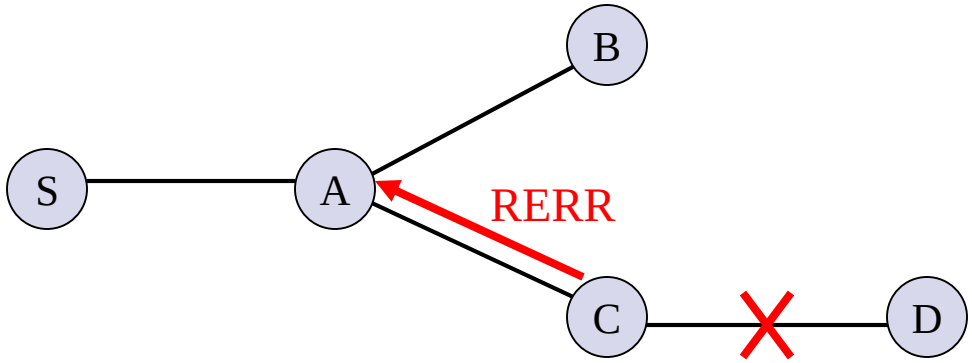
8. Node S receives RREP
   - Makes forward route entry to D
     dest = D, nexthop = C, hopcount = 3, Lifetime

Route Discovery - 10/10



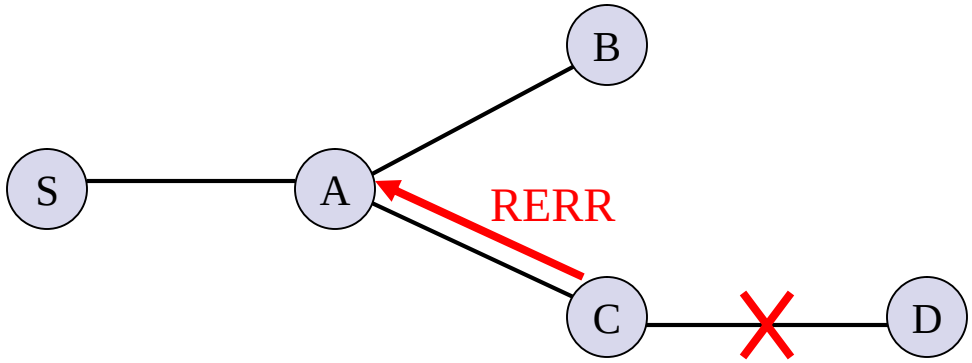
9. Node S sends data packets on route to D

Route Maintenance - 1/5



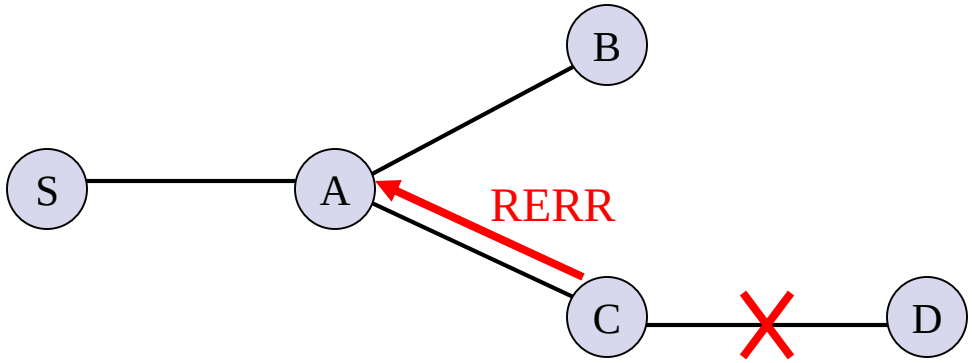
1. Link between C and D breaks down
   - C can perform local repair for the route to D
   - Methods to detect link breakage
     ① Hello Message, RREP whose TTL is one
     ② L2 Trigger

Route Maintenance - 2/5



1. Link between C and D breaks down
   - C can perform local repair for the route to D
2. Node C invalidates route to D in route table

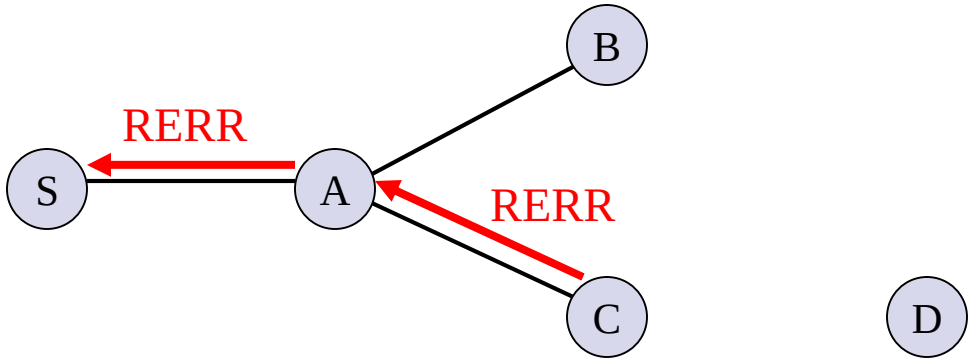
Route Maintenance - 3/5



3. Node C creates Route Error (RERR) message

- C creates a Route Error (RERR)
  Enters DestCount (=1), D's IP addr, seq #
- Lists all destinations which are now unreachable
  - "DestCount" field indicates the number of unreachable destinations included in the RERR message.
- Unicasts RERR to upstream neighbors in precursor list

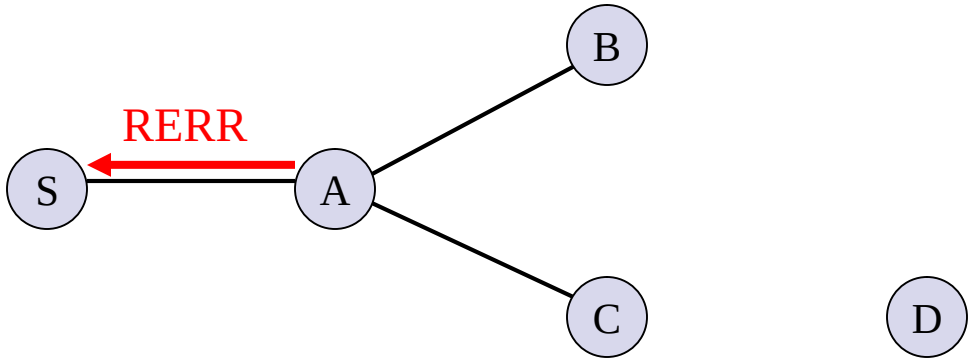
Route Maintenance - 4/5



4. Node A receives RERR

- Checks whether C is its next hop on route to D
- Deletes route to D
  or invalidates the route to D according to N flag
  (No delete flag)
  - When N flag is set, A does not send RERR to S and may reinitiate route discovery for D.
- Forwards RERR to S

Route Maintenance - 5/5



5. Node S receives RERR
   - Checks whether A is its next hop on route to D
   - Deletes route to D
   - Rediscovers route if still needed

Optimizations - 1/2

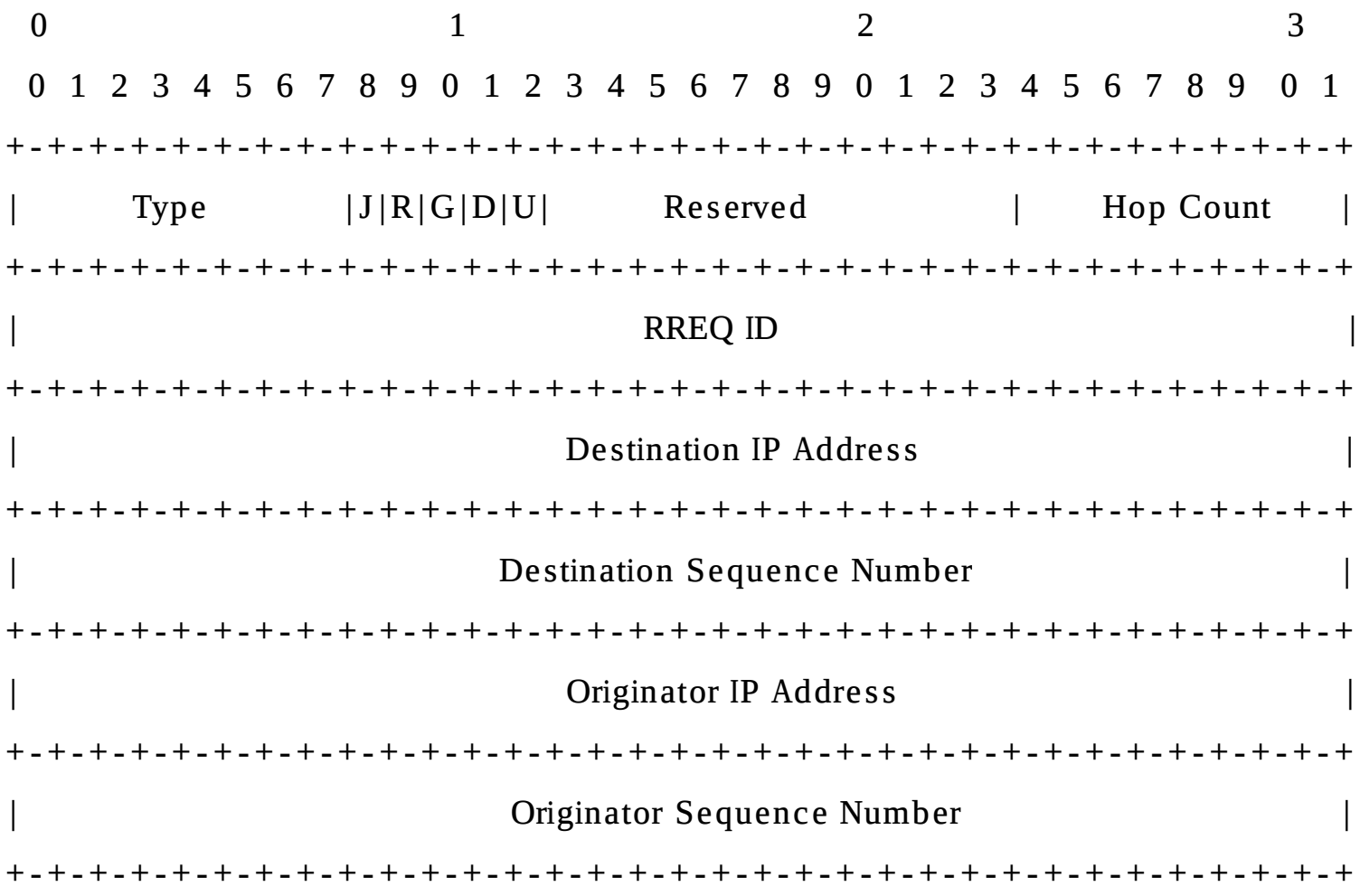
- Expanding Ring Search
  - It prevents flooding of network during route discovery
  - Control Time To Live (TTL) of RREQ to search incrementally larger areas of network
  - Advantage
    - Less overhead when successful
  - Disadvantage
    - Longer delay if route not found immediately

Optimizations - 2/2

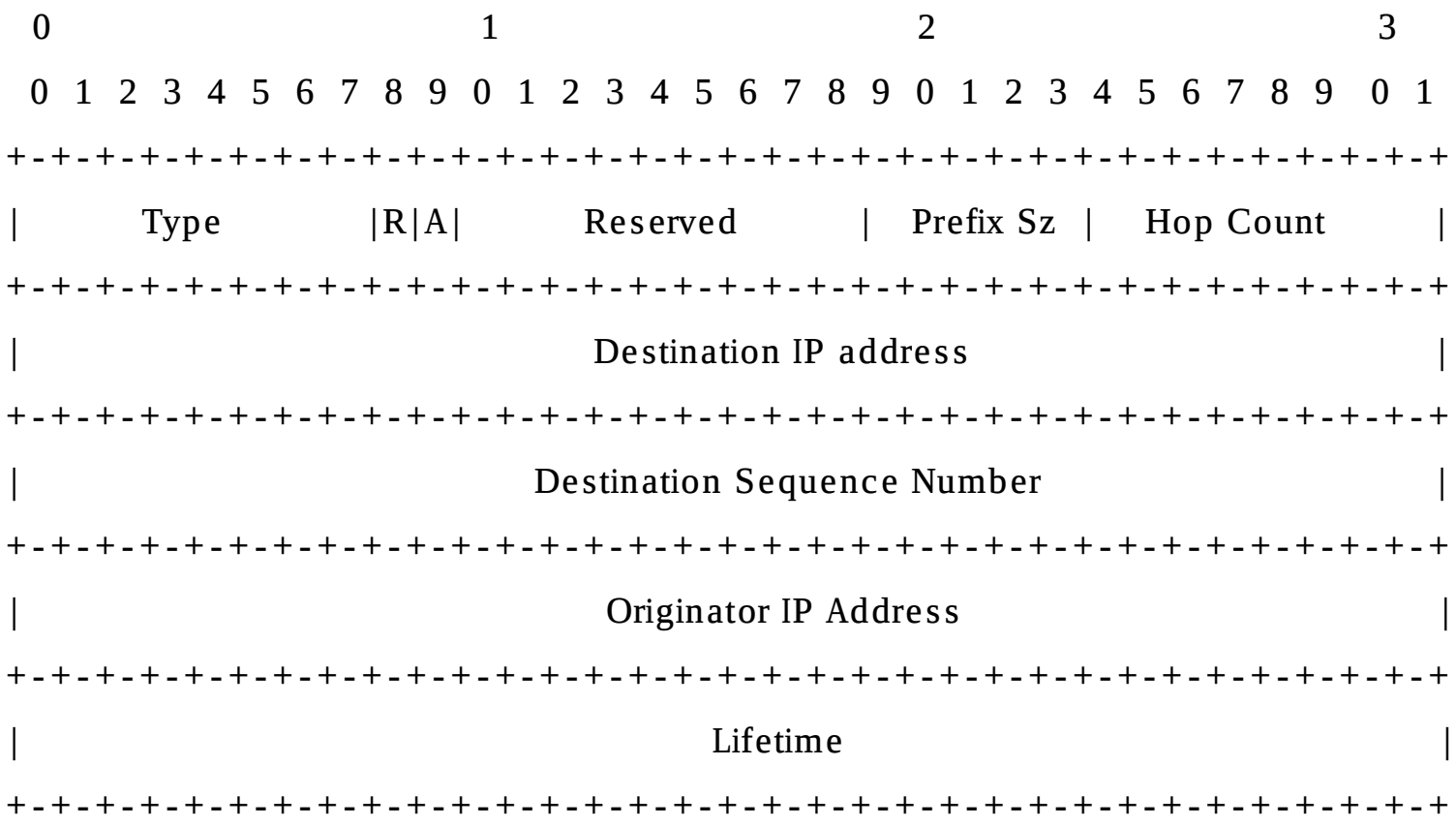
- Local Repair
  - It repairs breaks in active routes locally instead of notifying source.
    - If the first repair attempt is unsuccessful, it sends RERR to source.
  - Advantage
    - Link repair with less overhead, delay and packet loss.
  - Disadvantage
    - Longer delay and greater packet loss when unsuccessful

AODV Message Formats

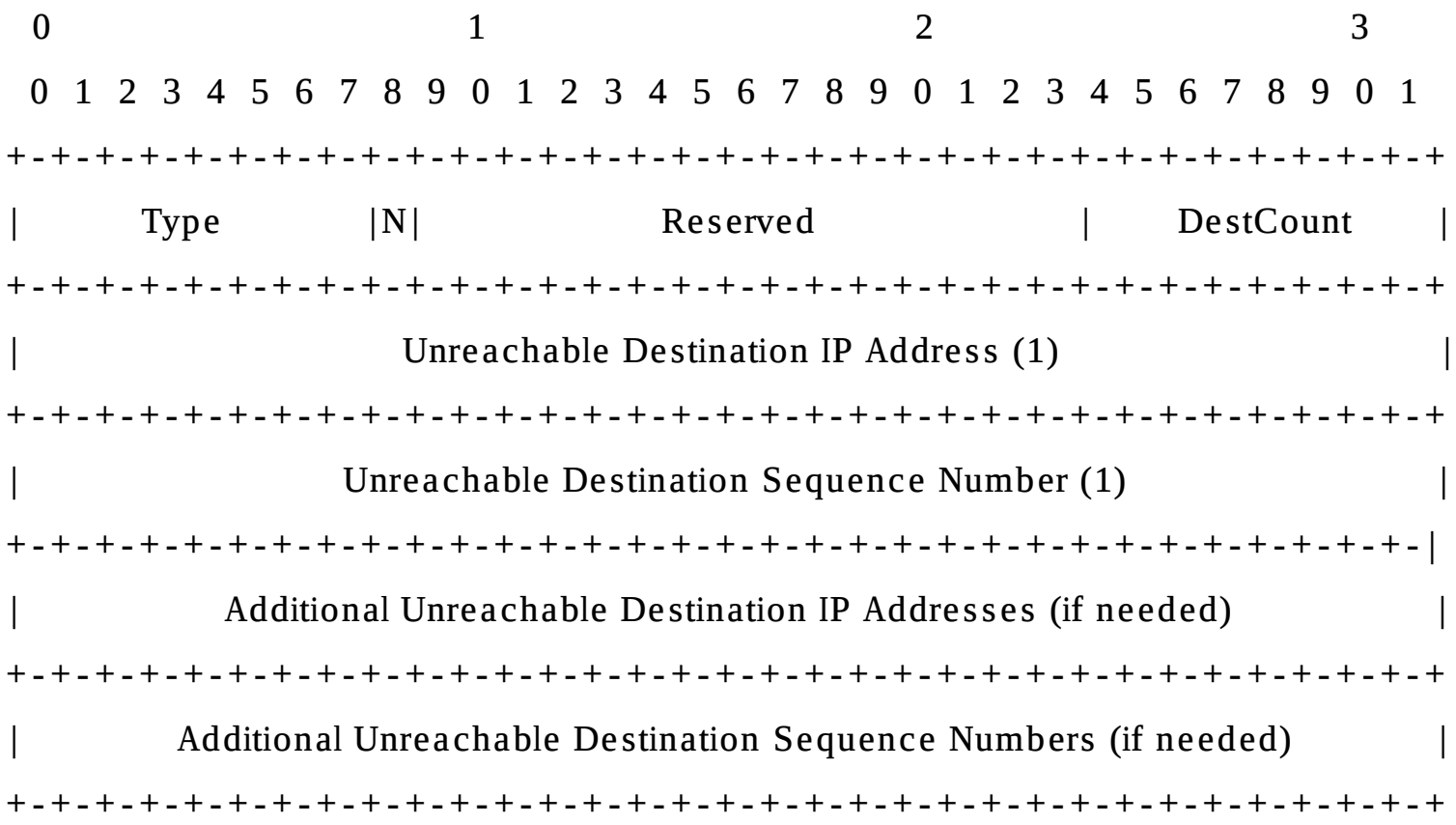
Route Request (RREQ) Message Format



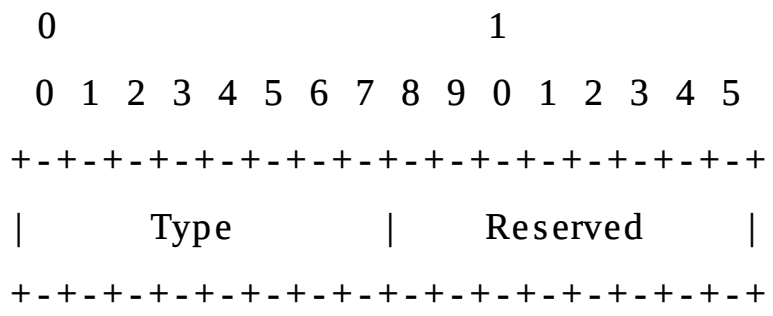
Route Reply (RREP) Message Format



Route Error (RERR) Message Format



Route Reply Acknowledgement (RREP-ACK) Message Format



- RREP-ACK message MUST be sent in response to an RREP message with the 'A' bit set.
  - This is typically done when there is danger of unidirectional links preventing the completion of a Route Discovery cycle.

IPv6 AODV

C. Perkins, E. Belding-Royer and S. Das,
draft-perkins-manet-aodv6-01,
November 2001.

AODV for IPv6 Operation

- Message Handling
  - The handling of IPv6 AODV for IPv6 AODV messages is analogous to IPv4 AODV.
    - AODV messages have the formats appropriate for use with 128-bit IPv6 addresses.
- ICMP Processing
  - Whenever IPv4 AODV specifies use of ICMP, the operation for IPv6 uses ICMPv6.
- Configuration Parameters
  - IPv4 and IPv6 AODVs use the same parameters.

MANET Implementation :
Kernel AODV

Contents

Introduction

- Kernel AODV
  - It has been implemented by NIST (National Institute of Standards and Technology) in the USA.
    - http://w3.antd.nist.gov/wctg/aodv_kernel/
  - AODV draft version 11
    - It has been implemented according to AODV draft version 11, draft-ietf-manet-aodv-11.txt

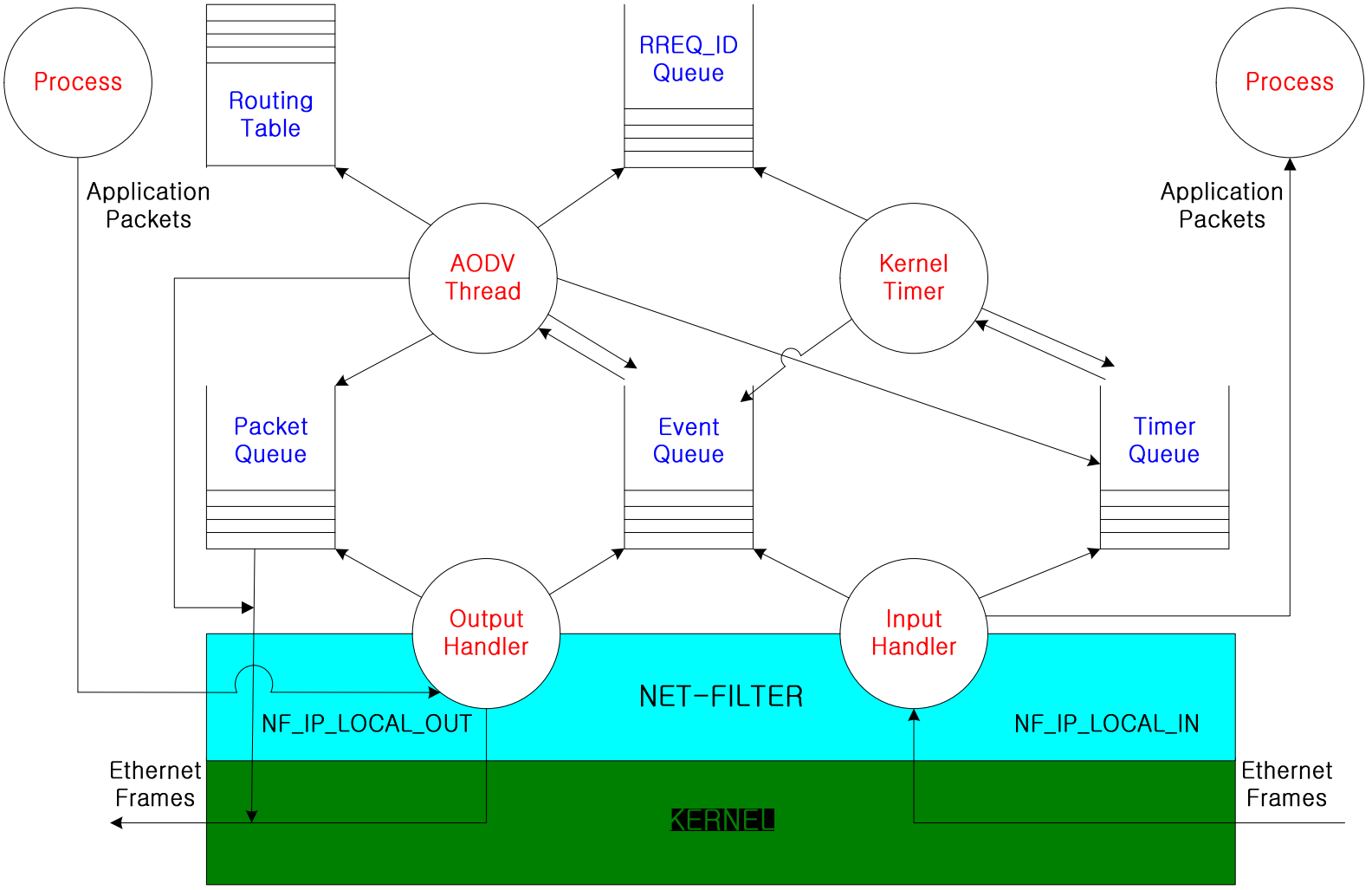
Installation of Kernel AODV

- Download of Kernel AODV
  - URL
    - http://w3.antd.nist.gov/wctg/aodv_kernel/kaodv_download.htm
- Installation Sequence
  - $ tar zxvf kernel-aodv_v2.2.2.tgz
  - $ mv kernel-aodv_v2.2.2
  - $ vi Makefile
    - TARGET:=x86
  - $ make
  - $ make install
  - $ start.sh
    - To start the node as gateway, issue "**start_gateway.sh**" instead of "**start.sh**".

start.sh

```
#!/bin/sh

#AODV Dev
# Tells AODV which interface use wish to use.

AODV_DEV="wlan0"

#Local Subnet
# If there is a small network attached to this node
# and you want to route through it, then describe it
# here. Other nodes will also be able to get routes
# to it. This is not commonly set.
# You must specify a subnet along with a subnet mask.
#LOCAL_SUBNET="local_subnet=192.168.1.0/255.255.255.0"

echo "1" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/route/min_delay

insmod kernel_aodv.o use_dev=$AODV_DEV $LOCAL_SUBNET
```

Architecture of Kernel AODV



Components of Kernel AODV - 1/3

- Input Handler
  - Checks if incoming packets from other nodes are AODV packets, which use UDP protocol with port 654.
- Output Handler
  - Catches local outgoing packets before sending.
    - If the destination does not exist in routing table, it enqueues the packet to packet queue and then sends RREQ.
- AODV Thread
  - Processes events in event queue in aodv().
- Kernel Timer
  - When a timer expires, it enqueues an event or reset the timer in timer_queue_signal().

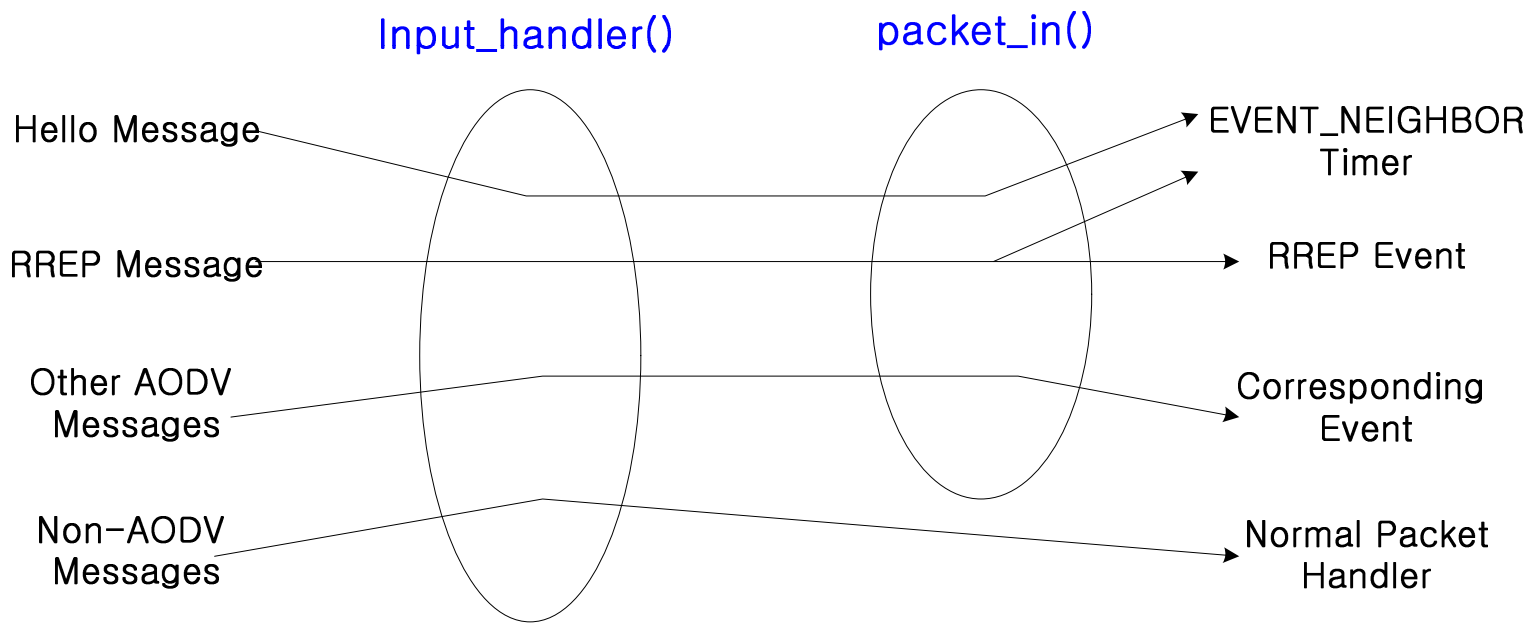
Components of Kernel AODV - 2/3

- Timer Queue
  - Queue for timers with the following types
    - EVENT_RREQ, EVENT_HELLO, EVENT_CLEANUP, and EVENT_NEIGHBOR
- Event Queue
  - Queue for timers with the following types
    - EVENT_RREQ, EVENT_RREP, EVENT_RREP_ACK, EVENT_RERR, and EVENT_CLEANUP
- Packet Queue
  - Queue for local outgoing data packets waiting for a route
    - The data packets wait for an RREP after an RREQ has been sent.
- RREQ_ID Queue
  - Queue for RREQ IDs to determine whether it has received an RREQ with the same Originator IP Address and RREQ ID whenever a node receives an RREQ.

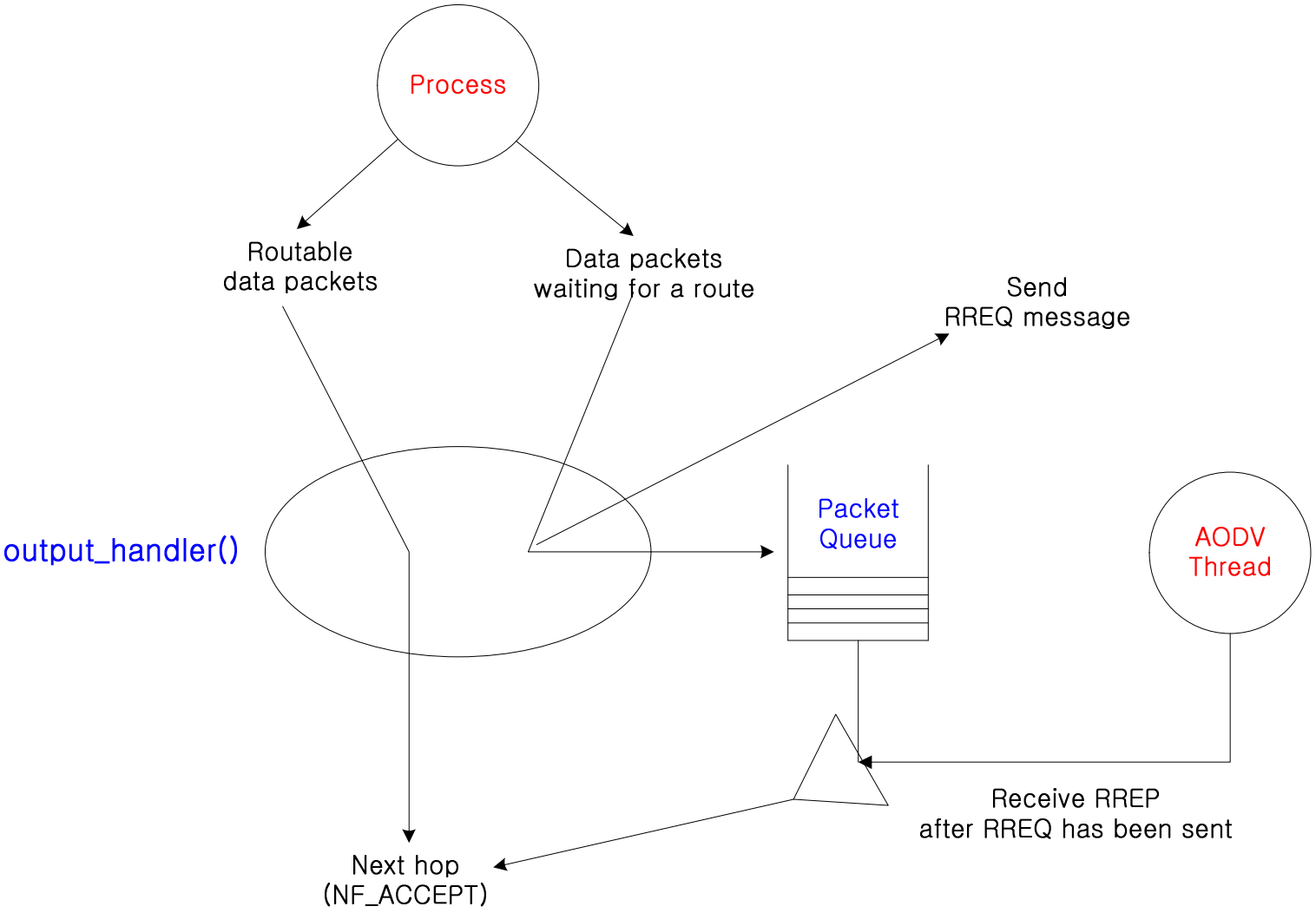
Components of Kernel AODV - 3/3

- Routing Table
  - Routing Table for forwarding data packets
- Precursor List
  - List for processing RERR message to indicate link loss
    - It contains the IP addresses for its neighbors that are likely to use it as a next hop towards each destination.
- Neighbor List
  - List of neighbor nodes discovered by Hello messages
- Interface List
  - List of local network interfaces

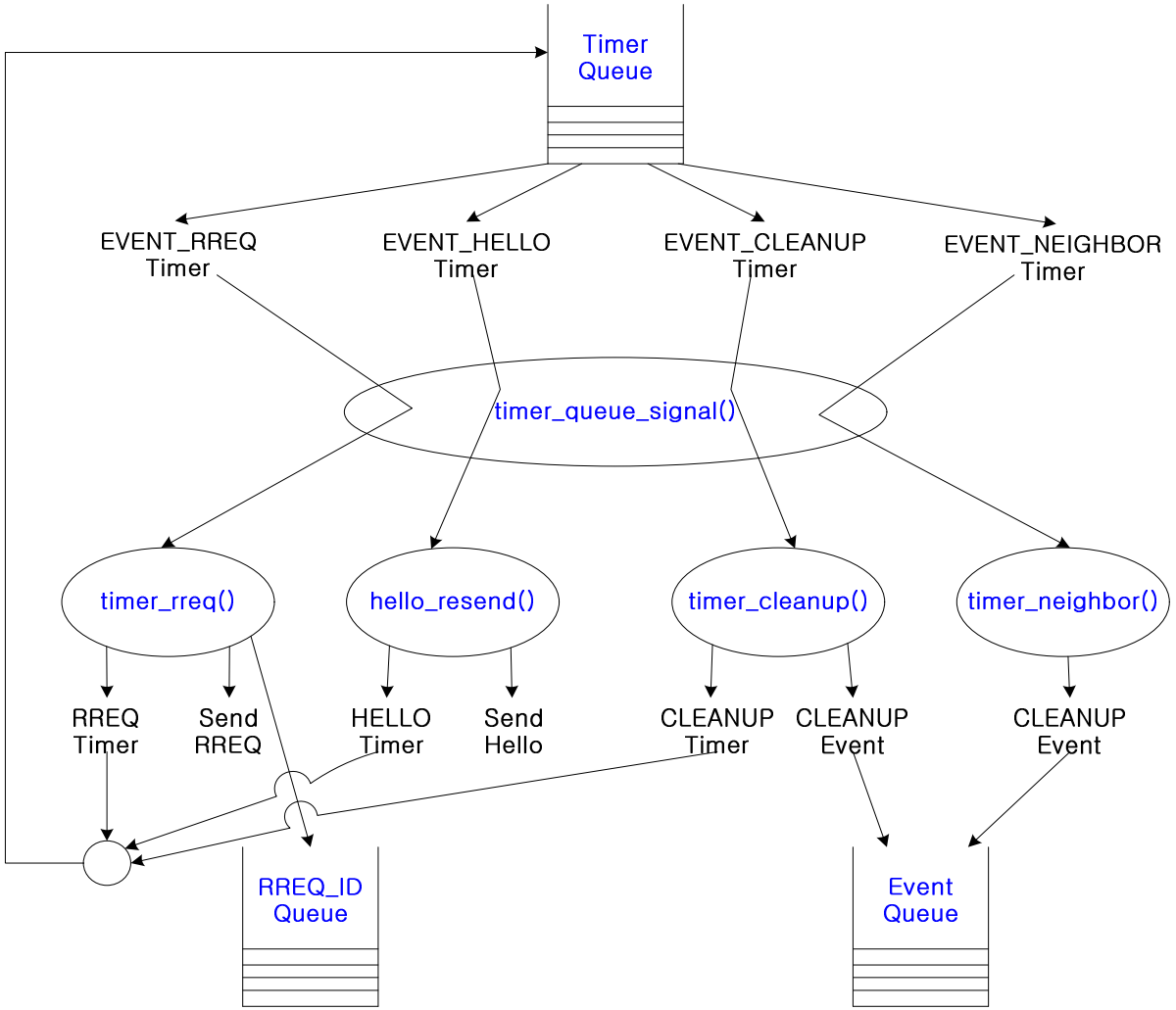
Input Handler



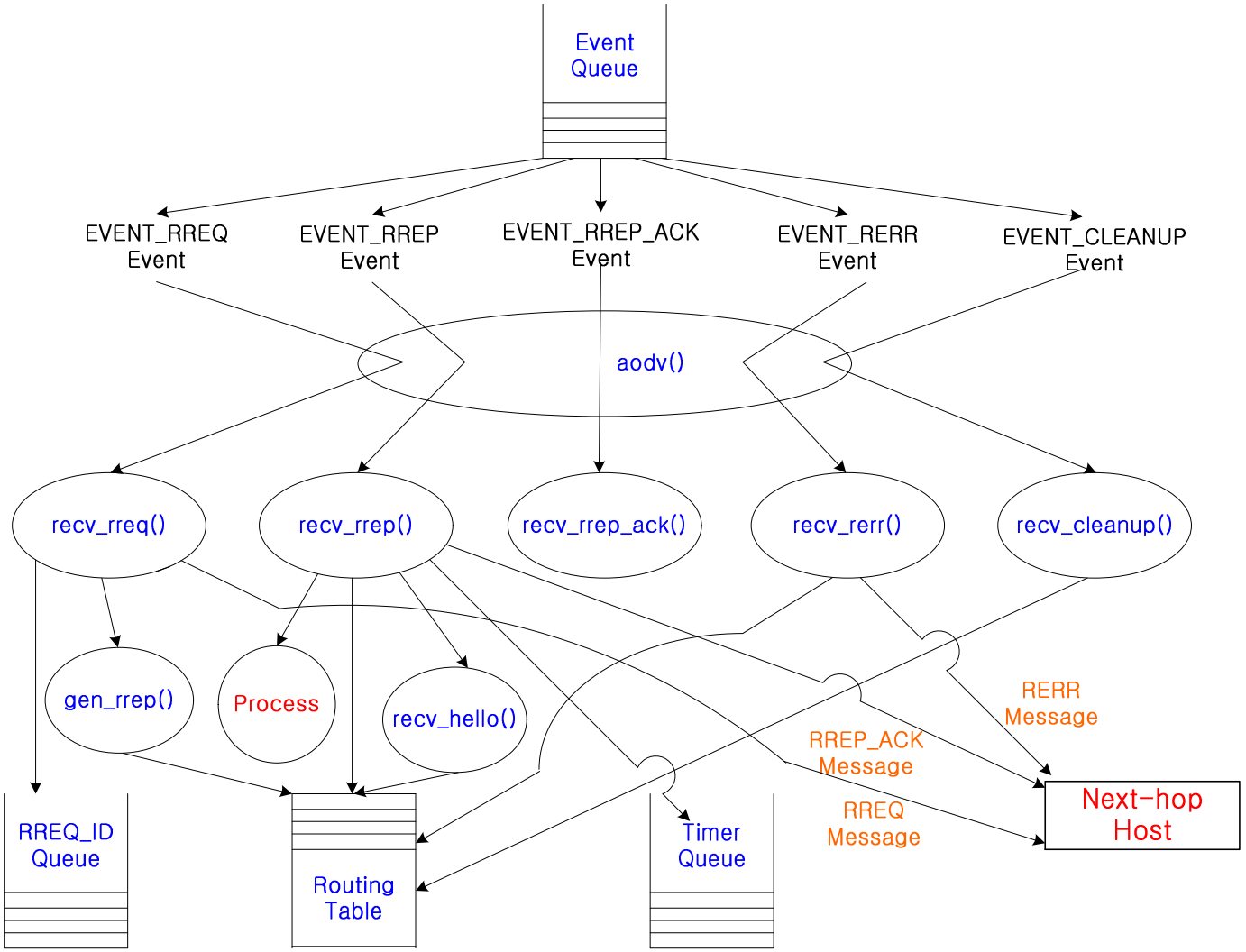
Output Handler



Kernel Timer



AODV Thread


Event Queue
EVENT_RREQ Event
EVENT_RREP Event
EVENT_RREP_ACK Event
EVENT_RERR Event
EVENT_CLEANUP Event
aodv()
recv_rreq()
recv_rrep()
recv_rrep_ack()
recv_rerr()
recv_cleanup()
gen_rrep()
Process
recv_hello()
RERR Message
RREP_ACK Message
RREQ Message
Next-hop Host
RREQ_ID Queue
Routing Table
Timer Queue

Data and Control Flows

- Route Request
  - Generating RREQ
  - Receiving RREQ
- Route Reply
  - Generating RREP
  - Receiving RREP
- Route Error
  - Generating RERR
  - Receiving RERR
- Hello Message
  - Generating Hello
  - Receiving Hello

MANET Testbed

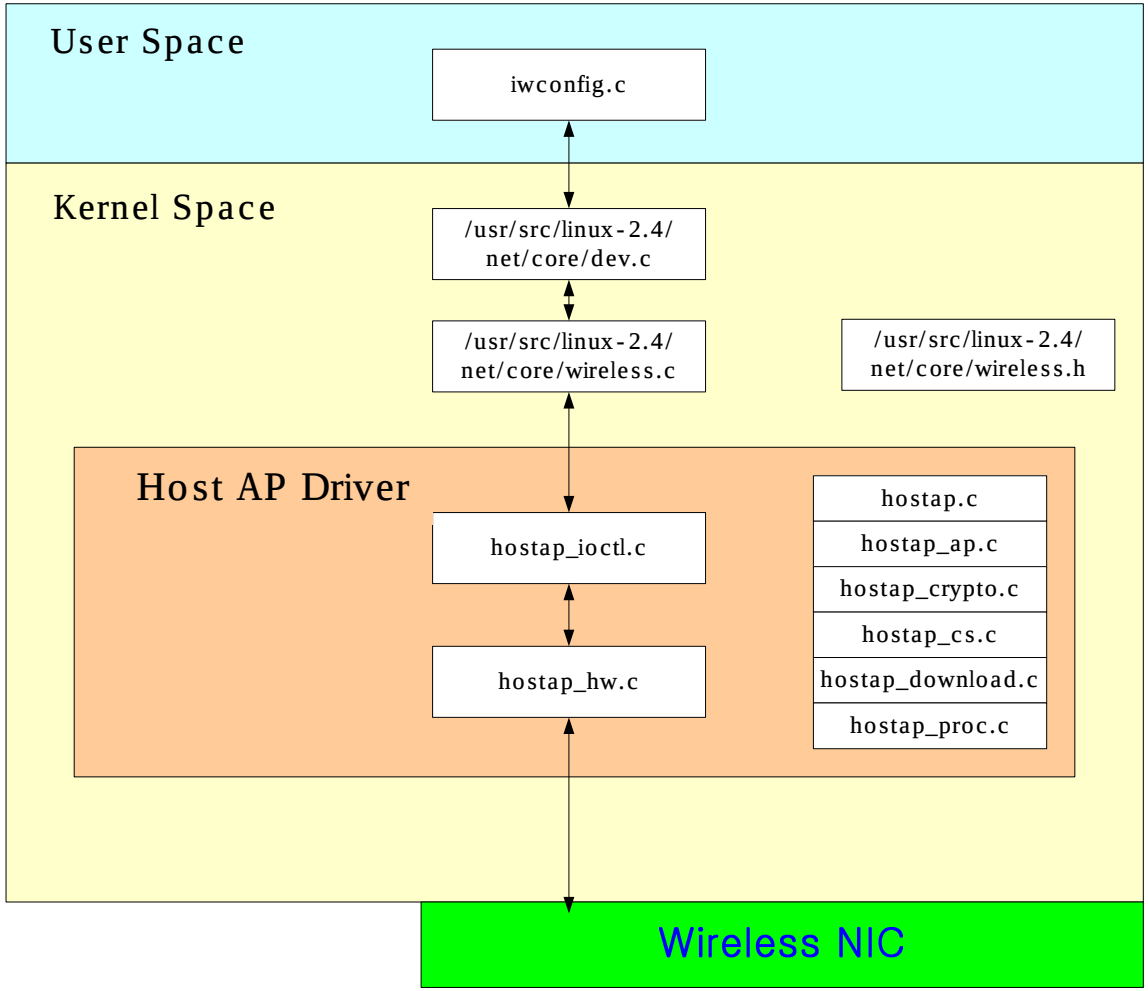
Contents

- Testbed for MANET
- Control Diagram of ETRI MAC Filtering Tool
- Host AP Driver for Intersil Prism2/2.5/3
- ETRI MANET Testbed
- Experiment in MANET Testbed

Testbed for MANET

- Motivation
  - There is much difficulty in managing the topology of MANET for testing protocols and applications.
- Topology Configuration Method
  - For testing multi-hop network configuration,
    - We can use **MAC-filtering** to filter out packets in other links.
      - MAC Filtering Tool of ETRI
        - http://www.adhoc.6ants.net/technology-transfer/index.html
      - MacKill of Uppsala University
        - http://user.it.uu.se/~henrikl/aodv/
- Routing Protocols
  - We used IPv4 AODV and IPv6 MAODV as Ad Hoc routing protocols.

Control Diagram of ETRI MAC Filtering Tool

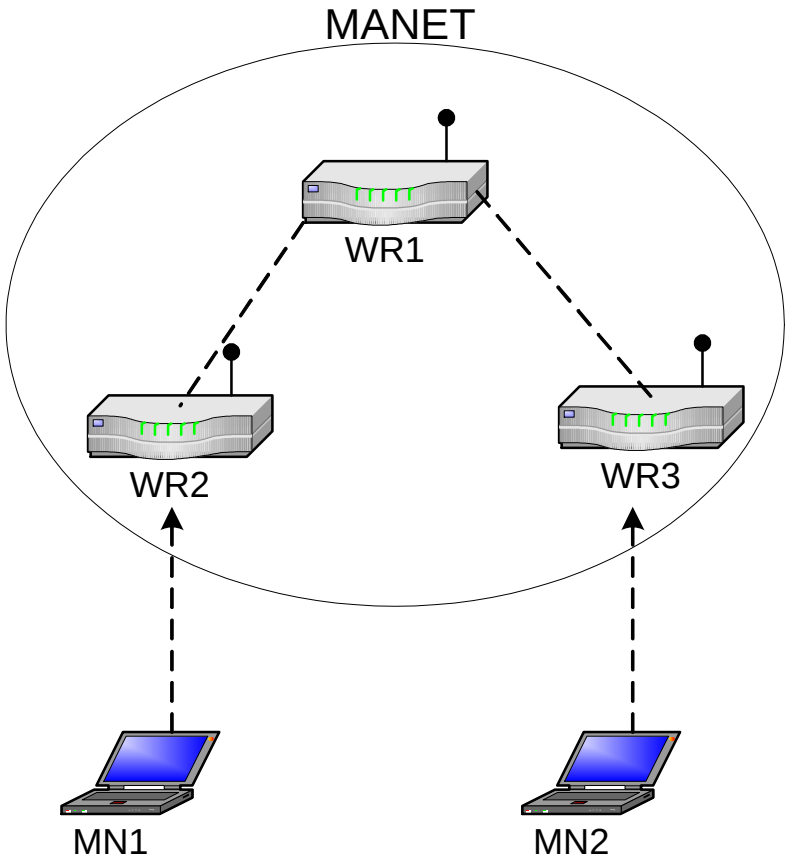


Host AP Driver for Intersil Prism2/2.5/3

- Host AP Driver
  - It is a Linux driver for wireless LAN cards based on Intersil's Prism2/2.5/3 chipset
  - It implements the following basic functionality
    - to initialize and configure Prism2-based cards
    - to send and receive frames, and
    - to gather statistics, etc.
- Host AP mode
  - It takes care of IEEE 802.11 management functions in the host computer and acts as an access point.
- Download of Host AP Driver
  - http://hostap.epitest.fi/

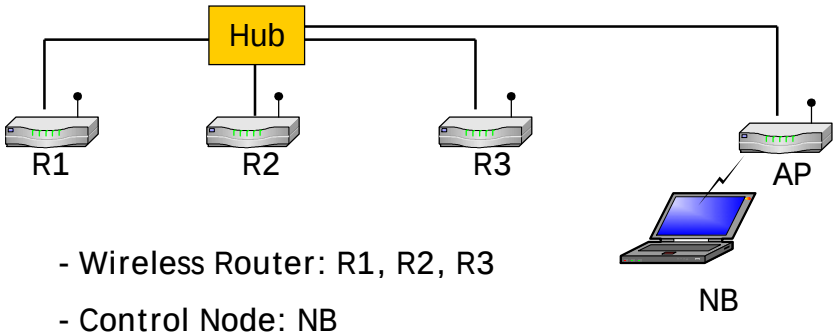
ETRI MANET Testbed

Wireless Router



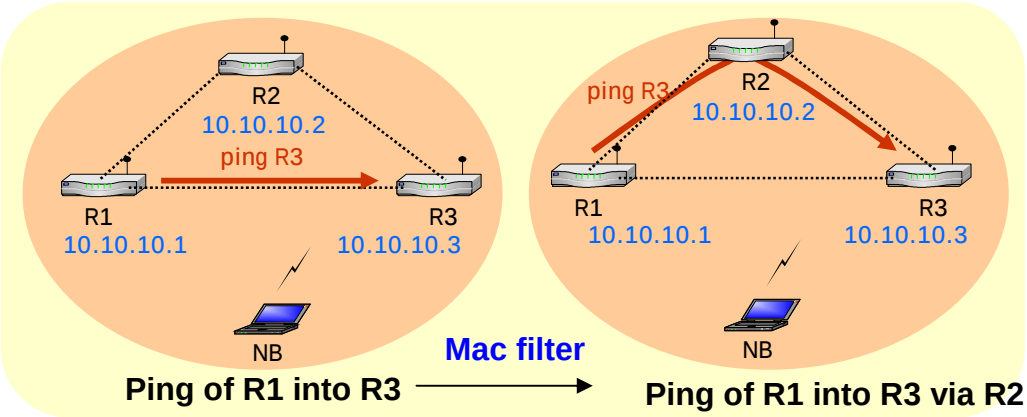
Protocol Test in MANET Testbed

Experiment in MANET Testbed



- Wireless Router: R1, R2, R3
- Control Node: NB

MANET Testbed



Test Scenario

MANET Simulation :
AODV in NS-2

Contents

- Introduction of NS-2
- AODV Code in NS-2
- AODV Simulation
- Schematic of MobileNode
- Simulation Example
  - Ad Hoc IP Address Autoconfiguraton

Introduction of NS-2

- NS-2
  - It provides a variety of simulations in wired and wireless networks.
    - Wired Network Simulation
      - Routing: Distance Vector, Link State, and PIM-SM
      - Transportation: TCP and UDP
      - Traffic Sources: web, ftp, telnet, cbr, and stochastic
      - Queuing Disciplines: drop-tail, RED, FQ, SFQ, and DRR
      - QoS: InterServ and DiffServ
    - Wireless Network Simulation
      - Routing: Ad hoc routing and Mobile IP
      - Directed Diffusion
      - Sensor Network
  - More Information
    - http://www.isi.edu/nsnam/ns/

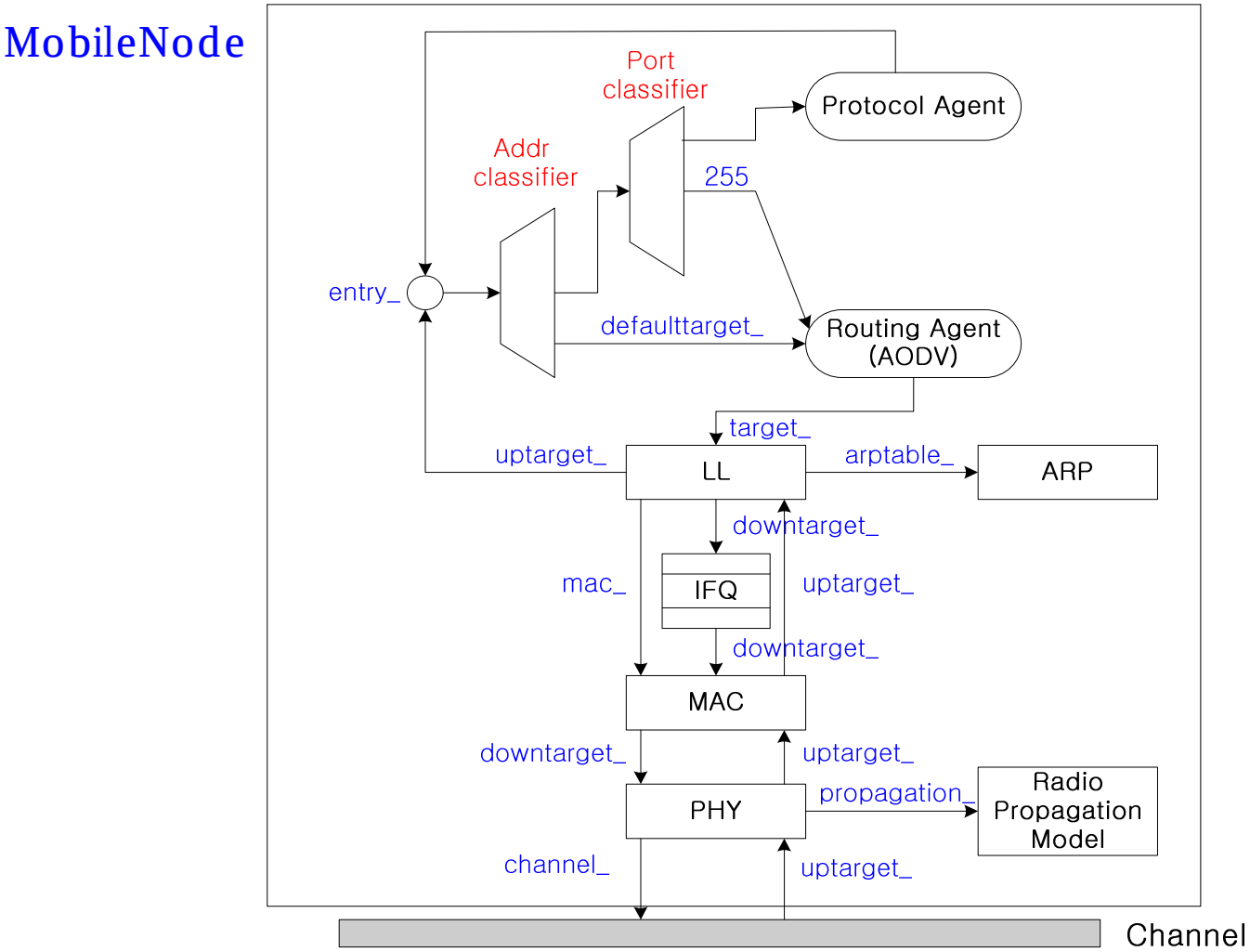
AODV Code in NS-2

- AODV Source Code
  - ns-2.26/aodv/
- AODV Test Suit
  - ns-2.26/tcl/test/test-suite-wireless-lan-aodv.tcl
- Scenario Generators
  - Movement Scenario Generator
    - ns-2.26/indep-utils/cmu-scen-gen/setdest/setdest
  - Traffic Scenario Generator
    - ns-2.26/indep-utils/cmu-scen-gen/cbrgen.tcl
    - e.g., CBR or TCP traffic

AODV Simulation

- Simulation Procedure
  - Define simulation objective
    - e.g., IP Address Autoconfiguration for AODV
  - Design classes and functions for simulation
    - i.e., make object diagram, flowchart, event trace diagram, state transition diagram, and data flow diagram, etc.
  - Consider how to apply the classes and functions to NS-2
    - How to extend AODV source code?
    - How to extend the other codes of NS-2?
  - Consider how to extract simulation result
    - specification of trace and monitoring
    - processing of trace file into graph or table

Schematic of MobileNode



Simulation Example:
Ad Hoc IP Address Autoconfiguraton

Contents

Ad Hoc IP Address Autoconfiguration (AAA)

- Configuration of Unicast Address in Network Interface
  - Precedent step for IP networking
  - Methods of IP address configuration in network interface
    - Manual configuration
    - Automatic configuration
- Consideration of IP address configuration
  - A unique address should be assigned.
  - Automatic configuration is needed for user's convenience.
- Addressing in MANET
  - Each mobile node is necessary to autoconfigure its IP address through DAD.
    - An arbitrary address is selected.
    - The uniqueness of the address is verified though Duplicate Address Detection (DAD).

Strong DAD

- Definition
  - $A_i(t)$: Address assigned to node i at time t.
  - For each address a != *undefined*, $S_a(t) = \{j \mid A_j(t) = a\}$.

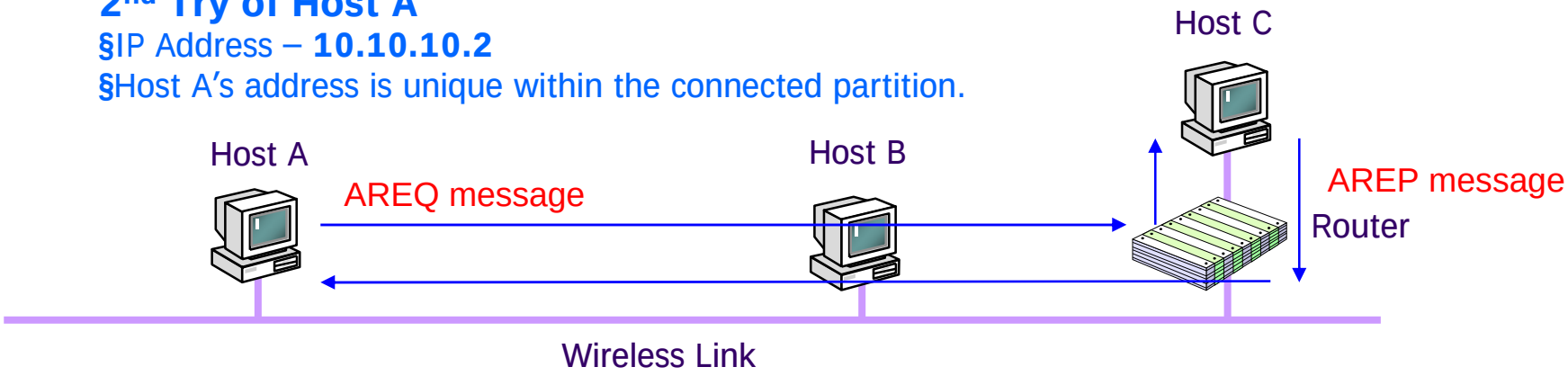
- Condition of Strong DAD
  - Within a finite bounded time interval after t, at least one node in $S_a(t)$ will detect that $|S_a(t)| > 1$.

Example of Strong DAD

1st Try of Host A
- §IP Address – **10.10.10.1**
- §Host A's address conflicts with Host C's.

2nd Try of Host A
- §IP Address – **10.10.10.2**
- §Host A's address is unique within the connected partition.

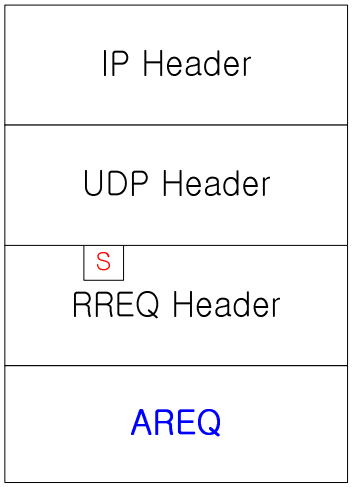
IP Address of Host C
- §IP Address – **10.10.10.1**



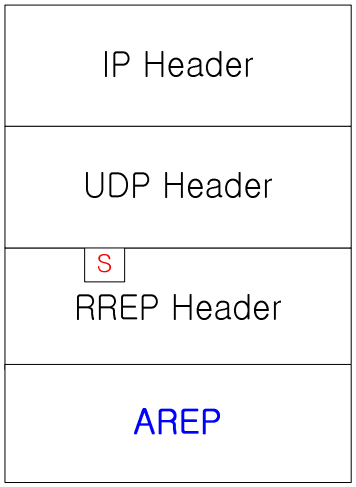
Where AREQ : Address Request message,
AREP : Address Reply message

Delivery of AREQ and AREP Messages

- Delivery through Piggyback
  - AREQ message
    - It is piggybacked on AODV RREQ message.
    - **This allows a reverse path for AREP message to be set up.**
  - AREP message
    - It is piggybacked on AODV RREP message.
- Message Structure

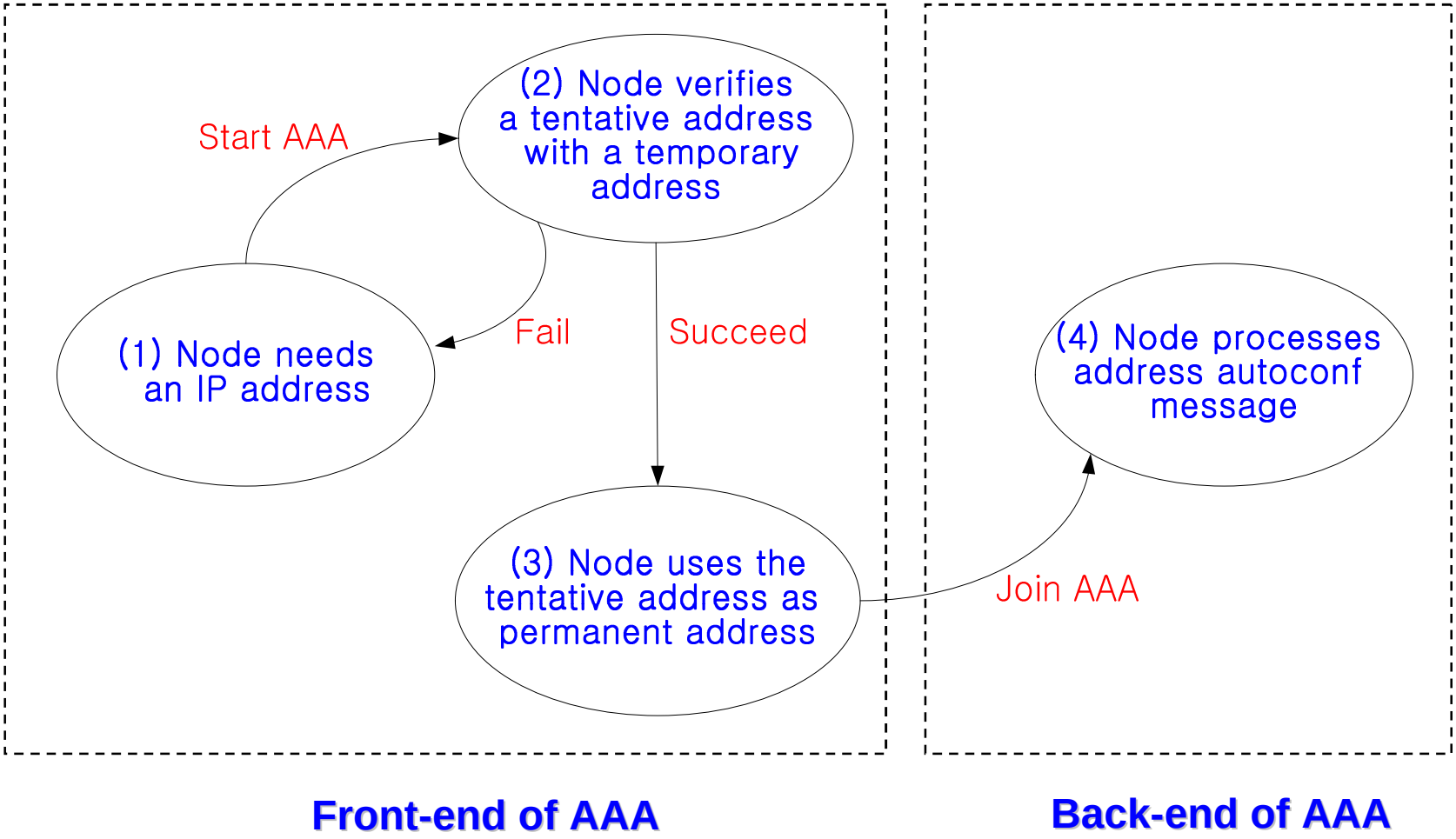


(a) AREQ Message



(b) AREP Message

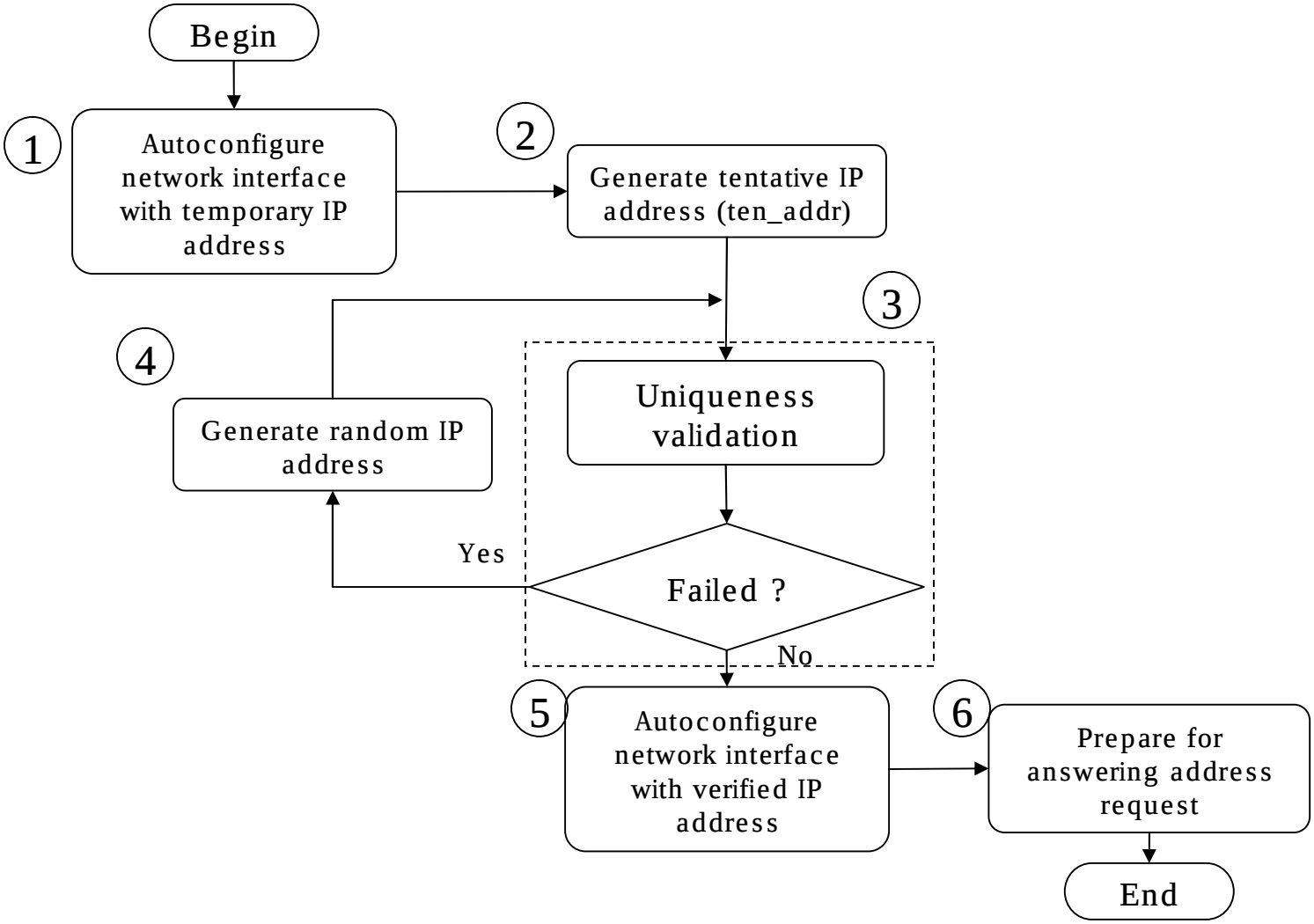
State Transition Diagram of Node for AAA – 1/2



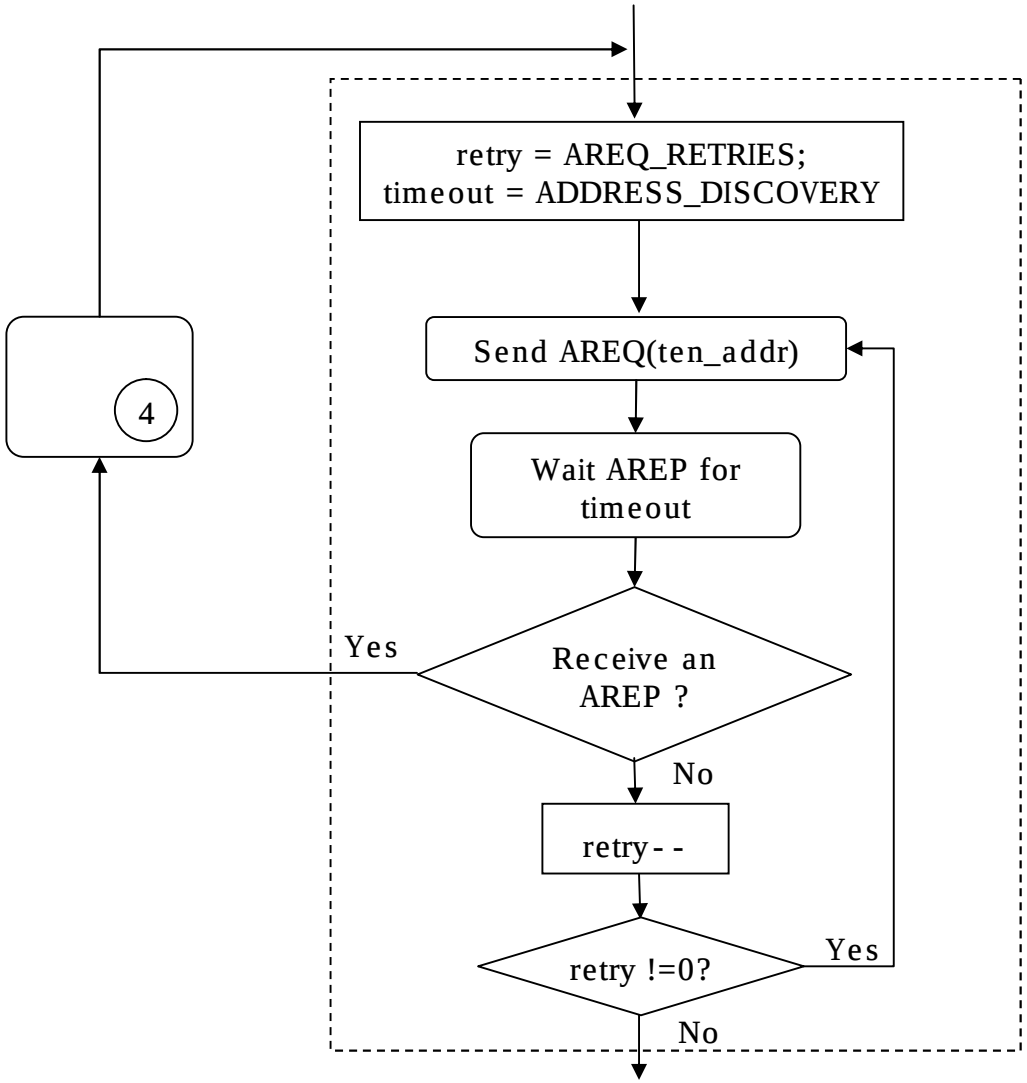
State Transition Diagram of Node for AAA – 2/2

- Node State
  - (1) Node needs an IP address
    - Node's State 1: **ADHOC_AUTOCONF_BEFORE**
  - (2) Node verifies a tentative address with a temporary address
    - Node's State 2: **ADHOC_AUTOCONF_UNDER**
  - (3) Node uses the tentative address as permanent address
    - Node's State 3: **ADHOC_AUTOCONF_AFTER**
  - (4) Node processes address autoconf message
    - Node's State 4: **ADHOC_AUTOCONF_PROCESS**

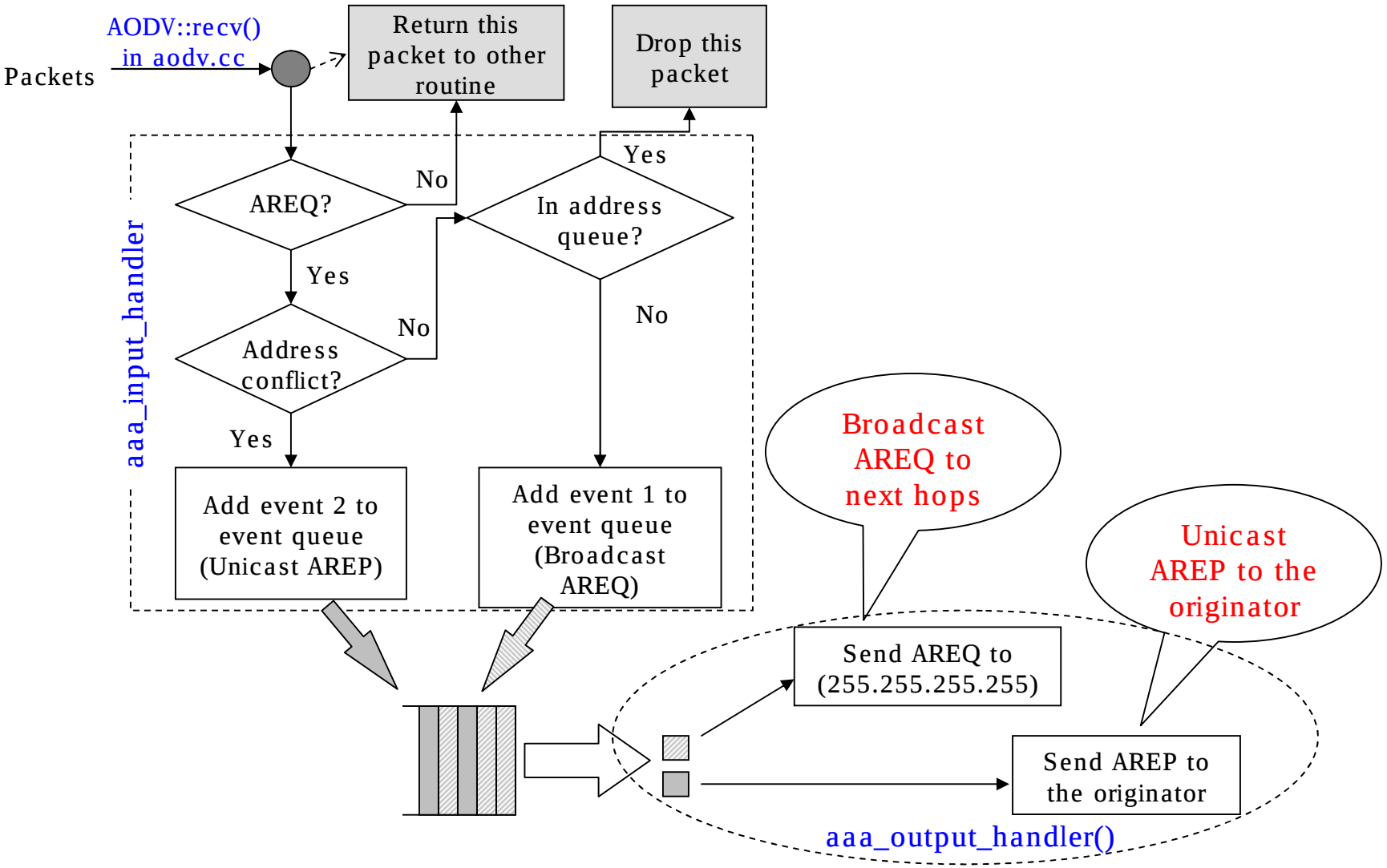
Flowchart of Front-end of AAA – 1/2



Flowchart of Front-end of AAA – 2/2



Flowchart of Back-end of AAA



AAA: Scenario – 1/3

- Test Suit: ns-2.26/tcl/test/aodv-aaa.tcl

```
global opt
set opt(chan)   Channel/WirelessChannel
set opt(prop)   Propagation/TwoRayGround
set opt(netif)  Phy/WirelessPhy
set opt(mac)    Mac/802_11
set opt(ifq)    Queue/DropTail/PriQueue
set opt(ll)     LL
set opt(ant)    Antenna/OmniAntenna
set opt(x)      10 ;# X dimension of the topography
set opt(y)      10 ;# Y dimension of the topography
set opt(ifqlen) 50 ;# max packet in ifq
set opt(seed)   0.0
set opt(tr)     aodv.rands ;# trace file
set opt(lm)     "off" ;# log movement
set opt(rp)     aodv
set opt(cp)     "/home/paul/ns-2.26/tcl/test/scenario/traffic/cbr-2-5-2-1"
set opt(sc)     "/home/paul/ns-2.26/tcl/test/scenario/movement/scen-10x10-2-5-5-60"
set opt(nn)     2
set opt(stop)   60.0
```

AAA: Scenario – 2/3

- Topology: 10 m x 10 m
- Number of Nodes: 2
- Transmission Range: 250 m
- Channel Capacity: 2 Mbps
- Propagation Model: Propagation/TwoRayGround
- MAC: IEEE 802.11
- Traffic Model
  - Number of Sources: 2
  - CBR
    - 512 bytes/packet
    - 1 packet/sec
- Simulation Time: 60 s

AAA: Scenario – 3/3

- Traffic Scenario: ns-2.26/tcl/test/scenario/traffic/cbr-2-5-2-1

(1) Node 0 connecting to Node 1 at time 15.0

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(0) $udp_(0)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(1) $null_(1)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(1)
$ns_ at 12.0 "[$node_(0) set ragent_]
strong-dad 1"
$ns_ at 15.0 "$cbr_(0) start"
```

(2) Node 1 connecting to Node 0 at time 16.0

```
set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(1)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(0) $null_(0)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 1
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(0)
$ns_ at 13.0 "[$node_(1) set ragent_]
strong-dad 0"
$ns_ at 16.0 "$cbr_(1) start “
```

Tcl Code Extension

- ns-2.26/tcl/lib/ns-mobilenode.tcl

```
Node/MobileNode instproc init args {
  eval $self next $args
  $self instvar nifs_ arptable_ X_ Y_ Z_ nodetype_ rtagent_
  # rtagent_ is routing agent
  . . .
}
```

- ns-2.26/tcl/lib/ns-lib.tcl

```
Simulator instproc create-wireless-node args {
  . . .
  switch -exact $routingAgent_ {
    DSDV {
      set ragent [$self create-dsdv-agent $node]
    }
    . . .
    AODV {
    # set ragent_ of $node to pointer of ragent in order to call strong-dad in AODV agent
      set ragent [$self create-aodv-agent $node]
      set [$node set ragent_] $ragent
      . . .
```

C++ Code Extension - 1/10
: AODV Message Extension

- /ns-2.26/aodv/aodv_packet.h

```
/* Message Format for Ad Hoc IP Address Autoconfiguration */

struct icmp_adhoc_autoconf {
    u_int8_t    type; // Type of ICMP message
    u_int8_t    code; // Code of message type
    u_int16_t   checksum; // Checksum for ICMP message and parts of the IP header
    u_int32_t   identification; // Identification for ad hoc address autoconf message
    u_int32_t   originator_address; // Originator's IP Address
    u_int32_t   req_dup_address; // Requested or Duplicate IP Address
}; // sizeof(icmp_adhoc_autoconf) = 16 [byte]
```

C++ Code Extension – 2/10
: AODV Message Extension

- /ns-2.26/aodv/aodv_packet.h

```
/* AODV RREQ Message */

struct hdr_aodv_request {
        u_int8_t        rq_type;    // Packet Type
        //u_int8_t        reserved[2]; /* define flags with bit-field */
#if BYTE_ORDER == BIG_ENDIAN
        u_int8_t        rq_j_flag:1,    // Join flag
                        rq_r_flag:1,    // Repair flag
                        rq_g_flag:1,    // Gratuitous RREP flag
                        rq_d_flag:1,    // Destination only flag
                        rq_u_flag:1,    // Unknown sequence number flag
                       rq_s_flag:1,    // Strong-DAD flag
                       rq_w_flag:1,    // Weak-DAD flag
                        rq_reserved1:1; // reserved-bit 1
        u_int8_t        rq_reserved2:8; // reserved-bit 2
#endif
#if BYTE_ORDER == LITTLE_ENDIAN
. . .
/* Ad Hoc Address Autoconfiguraion (AAA) Message */
        struct icmp_adhoc_autoconf rq_aaa; // Address Request (AREQ) Message
        . . .
```

C++ Code Extension – 3/10
: AODV Message Extension

- /ns-2.26/aodv/aodv_packet.h

```
/* AODV RREQ Message - continue */
. . .
  inline int size() {
  int sz = 0;
  /*
    sz = sizeof(u_int8_t)       // rp_type
         + 2*sizeof(u_int8_t)   // rp_flags + reserved
         + sizeof(u_int8_t)     // rp_hop_count
         + sizeof(double)       // rp_timestamp
         + sizeof(nsaddr_t)     // rp_dst
         + sizeof(u_int32_t)    // rp_dst_seqno
         + sizeof(nsaddr_t)     // rp_src
         + sizeof(u_int32_t)    // rp_lifetime
         + sizeof(icmp_adhoc_autoconf); // rp_aaa == 16 [byte]
  */
    // sz = 6*sizeof(u_int32_t);
    sz = 10*sizeof(u_int32_t);
    assert (sz >= 0);
    return sz;
   }
 . . .
```

C++ Code Extension – 4/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.h

```
/** Definitions of MACRO Constants and Enum Type **/
. . .

/* Ad Hoc Address Autoconf Type */
#define ADHOC_AUTOCONF_AREQ 0x01
#define ADHOC_AUTOCONF_AREP 0x02
#define ADHOC_AUTOCONF_AERR 0x03

/* Enum Type: Node's State */
enum NodeState {ADHOC_AUTOCONF_BEFORE, ADHOC_AUTOCONF_UNDER, ADHOC_AUTOCONF_AFTER,
                ADHOC_AUTOCONF_PROCESS};

. . .
```

C++ Code Extension – 5/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.h

```
/* The Routing Agent */
class AODV: public Agent {
. . .
/* Packet TX Routines */
   //send a RREQ-AREQ message
   void sendRequest_AREQ(nsaddr_t dst);

   //send a RREP-AREP message
   void sendReply_AREP(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst, u_int32_t rpseq,
                       u_int32_t lifetime, double timestamp);
   . . .
/* Packet RX Routines */
   //receive a RREQ-AREQ message
   void recvRequest_AREQ(Packet *p);

   //receive a RREP-AREP message
   void recvReply_AREP(Packet *p);
   . . .

/* Strong DAD */
protected:
   void strong_dad(nsaddr_t tentative_addr); //performs Strong-DAD for tentative address
private:
   NodeState node_state; // Node's state
```

C++ Code Extension - 6/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.cc

```
int AODV::command(int argc, const char*const* argv) {
  bool result; //result of function's execution
  if(argc == 2) {
  . . .
  }
  else if(argc == 3) {
  . . .
    if(strcmp(argv[1], "index") == 0) {
      index = atoi(argv[2]);
      return TCL_OK;
    }
    else if(strncasecmp(argv[1], "strong-dad", 3) == 0)
    { /* perform Strong-DAD */
      int tentative_addr;
      tentative_addr = atoi(argv[2]);
      strong_dad(tentative_addr); //performs Strong-DAD
      return TCL_OK;
    }
  }
  return Agent::command(argc, argv);
}
```

C++ Code Extension - 7/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.cc

```
AODV::AODV(nsaddr_t id) : Agent(PT_AODV),
                          btimer(this), htimer(this), ntimer(this),
                          rtimer(this), lrtimer(this), rqueue()
{
  index = id;
  seqno = 2;
  bid = 1;

  LIST_INIT(&nbhead);
  LIST_INIT(&bihead);

  logtarget = 0;
  ifqueue = 0;

  //initialize Node's state with ADHOC_AUTOCONF_BEFORE
  node_state = ADHOC_AUTOCONF_BEFORE;
}
```

C++ Code Extension – 8/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.cc

```
void AODV::strong_dad(nsaddr_t tentative_addr)
{/* perform Strong-DAD */

  //add a route entry to route table
  aodv_rt_entry* rt;

  rt = rtable.rt_add(tentative_addr);
  if(rt == NULL) {
    cerr<<"AODV::strong_dad() : rt is NULL!"<<endl;
  }

  //set Node's state to ADHOC_AUTOCONF_UNDER
  node_state = ADHOC_AUTOCONF_UNDER;

  //send AREQ message to tentative_addr
  sendRequest_AREQ(tentative_addr);
}
```

C++ Code Extension – 9/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.cc

```
void AODV::sendRequest_AREQ(nsaddr_t tentative_addr)
{/* send AREQ message */
 // allocate a RREQ-AREQ packet
 Packet *p = Packet::alloc();
 struct hdr_cmn *ch = HDR_CMN(p);
 struct hdr_ip *ih = HDR_IP(p);
 struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
 . . .
 /* initialization of Ad Hoc Address Autoconf message */
 rq->rq_adhoc_autoconf.type = ADHOC_AUTOCONF_AREQ;
 rq->rq_adhoc_autoconf.code = 0;
 rq->rq_adhoc_autoconf.checksum = 0; //ignore the checksum
 rq->rq_adhoc_autoconf.identification = rq->rq_bcast_id;
 rq->rq_adhoc_autoconf.originator_address = rq->rq_src;
 rq->rq_adhoc_autoconf.req_dup_address = rq->rq_dst;

 /* register the current time with rq->rq_timestamp */
 rq->rq_timestamp = CURRENT_TIME;

 Scheduler::instance().schedule(target_, p, 0.);
}
```

C++ Code Extension - 10/10
: AODV Class Extension

- /ns-2.26/aodv/aodv.cc

```
void AODV::recvRequest(nsaddr_t tentative_addr)
{/* receive AREQ message */
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
  aodv_rt_entry *rt;

  /* identify if this RREQ is normal RREQ or RREQ-AREQ. */
  if(rq->rq_s_flag == 1)
  {
    recvRequest_AREQ(p); //process RREQ-AREQ message
    return;
  }
  /* Only when Node's state is ADHOC_AUTOCONF_PROCESS, Node processes AODV packets.
     Otherwise, Node drops the received packets. */
  if(node_state != ADHOC_AUTOCONF_PROCESS)
  {
    Packet::free(p);
    return;
  }
. . .
}
```

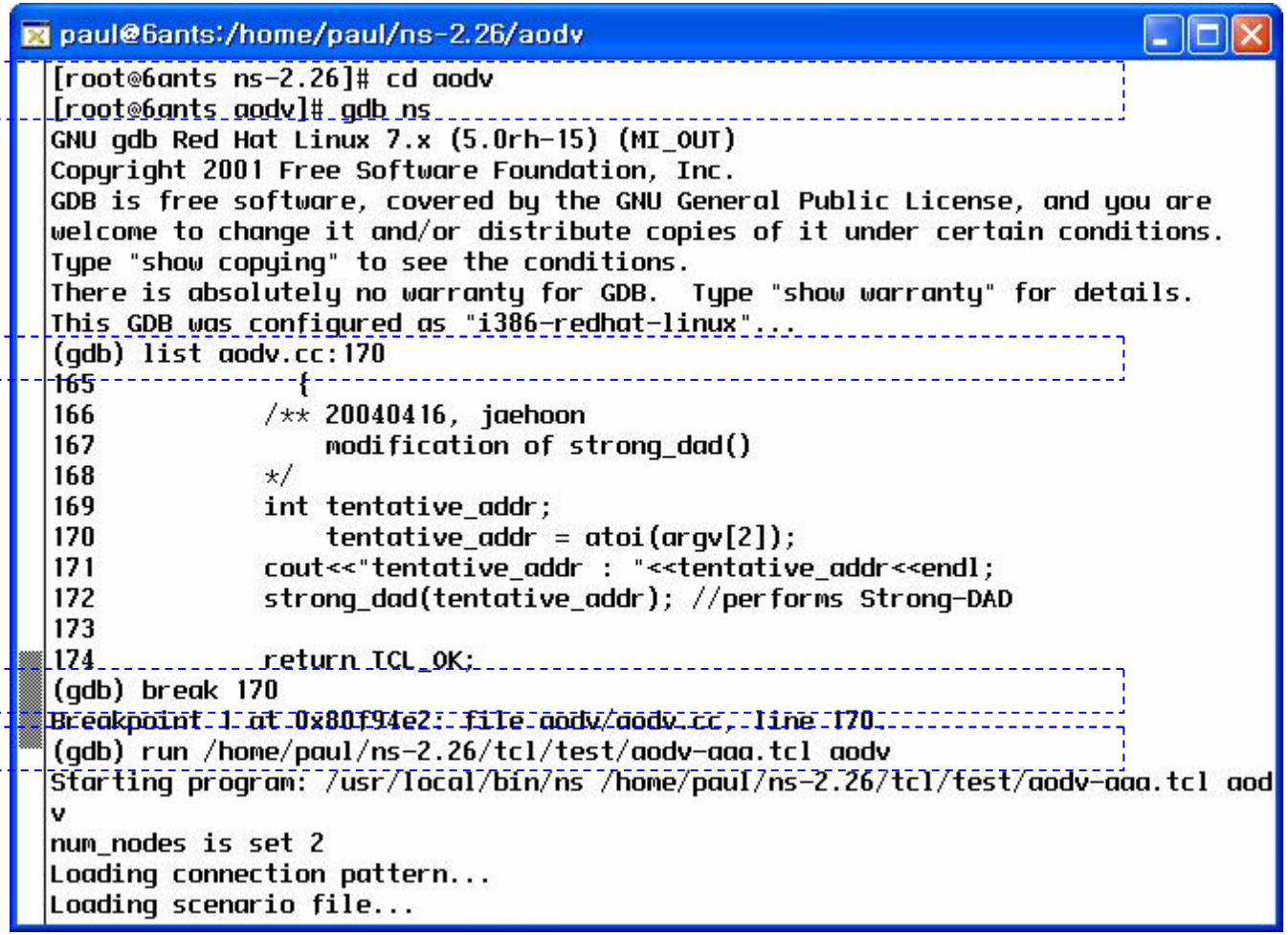
NS-2 Debugging – 1/4

- Installation for Debugging
  - Configure and compile NS-2 sources to support debugging
    - NS-2 Source and Other Libraries
      - Otcl, Tcl, Tclcl, Tcl-debug, Tk, and Ns2-2.26
  - Compilation Procedure

```
#cd /home/paul/simulation/ns2/ns-allinone-2.26
#cd otcl-1.0a8
#make clean; configure --enable-debug; make; make install
#cd ../tcl8.3.2/unix
#make clean; configure --enable-symbols; make; make install
#cd ../tclcl-1.0b13;
#make clean; configure --enable-debug; make; make install
#cd ../tcl-debug-2.0;
#make clean; configure; make; make install
#cd ../tk8.3.2/unix
#make clean; configure --enable-symbols; make; make install
#cd ../ns-2.26
#make clean; configure --enable-debug --with-tcldebug=../tcl-debug-2.0
#vi Makefile
  % specify g++ instead of c++ as c++ compiler
#make; make install
```

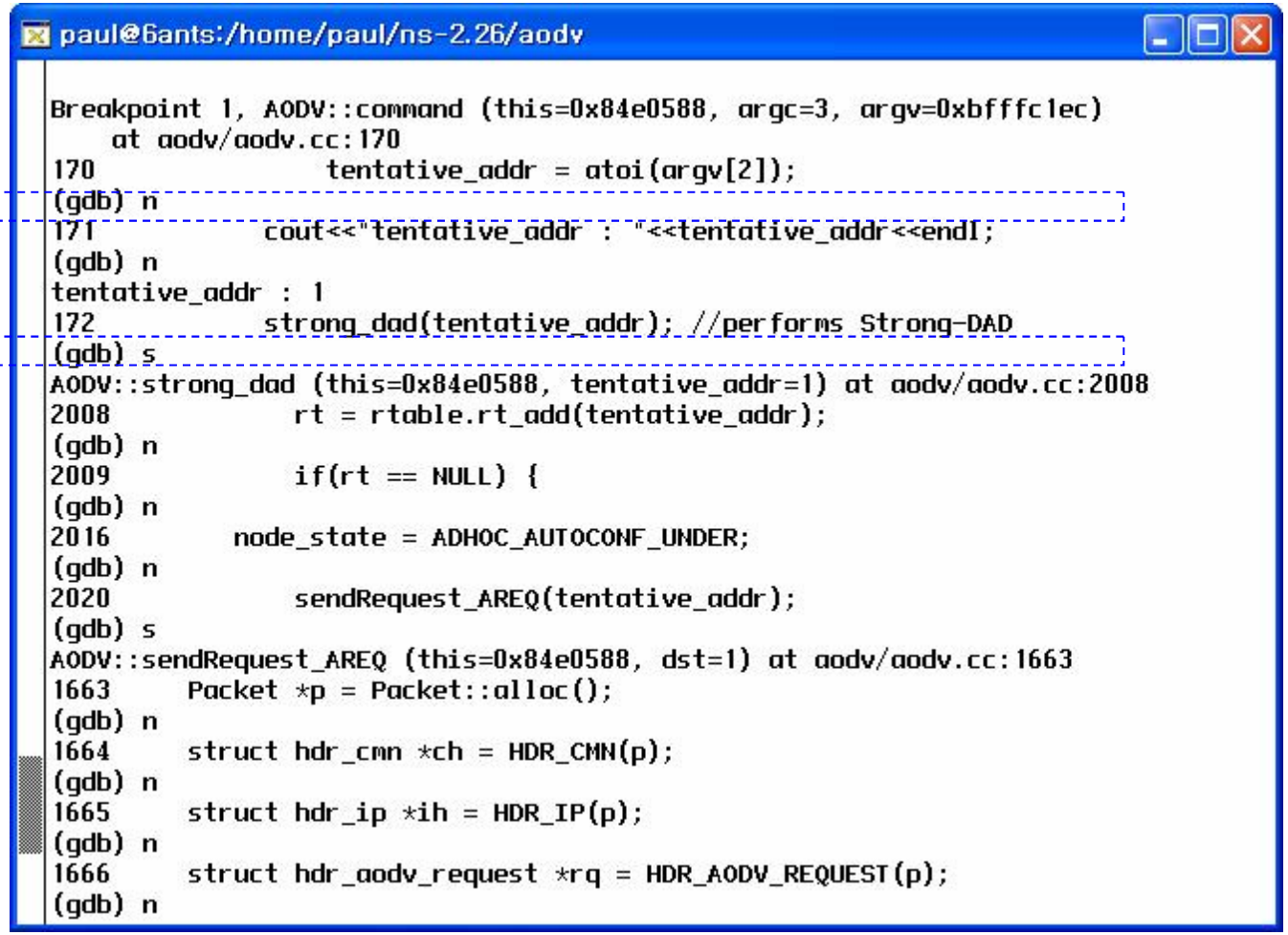
NS-2 Debugging – 2/4

- Debugging of AAA



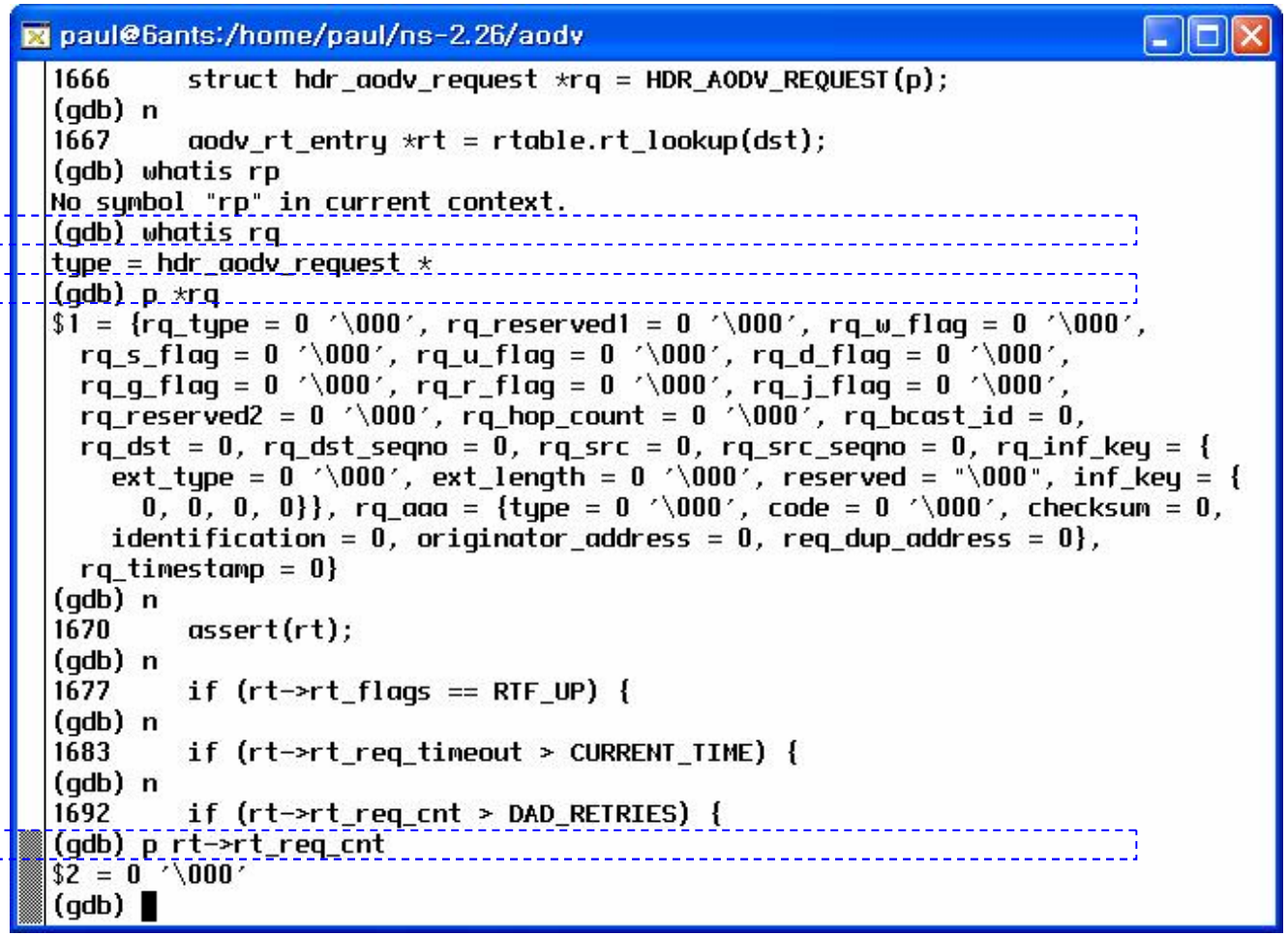
NS-2 Debugging – 3/4

- Debugging of AAA - Continue



NS-2 Debugging – 4/4

- Debugging of AAA - Continue



Conclusion

- MANET Networking is an important base technology for Ubiquitous Networking in Wired and Wired Networks.
- MANET Implementation and Test of Ad Hoc routing protocol is prerequisite to MANET Networking Development.
- Network Simulation is also needed for testing a variety of technologies for MANET as preceding step of Implementation.
  - NS-2 is a useful and free software for MANET simulation.

References

[1] C. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, July 2003.

[2] C. Perkins et al, "Ad hoc On-Demand Distance Vector (AODV) Routing for IP version 6", draft-perkins-manet-aodv6-01.txt, November 2001.

[3] Jaehoon Paul Jeong et al., "Requirements for Ad Hoc IP Address Autoconfiuguration", draft-jeong-manet-addr-autoconf-reqts-01.txt, February 2004.

[4] Jaehoon Paul Jeong et al., "Ad Hoc IP Address Autoconfiguration", draft-jeong-adhoc-ip-addr-autoconf-02.txt, February 2004.

[5] Jaehoon Paul Jeong et al., "Ad Hoc IP Address Autoconfiguration for AODV", draft-jeong-manet-aodv-addr-autoconf-00.txt, February 2004.

[6] Charles E. Perkins et al., "IP Address Autoconfiguration for Ad Hoc Networks", draft-ietf-manet-autoconf-01.txt, November 2001.

[7] Jaehoon Paul Jeong et al., "Auto-Networking Technologies for IPv6 Mobile Ad Hoc Networks", ICOIN 2004, February 2004.

[8] Ryuji Wakikawa et al., "Global connectivity for IPv6 Mobile Ad Hoc Networks", draft-wakikawa-manet-globalv6-03.txt, October 2003.

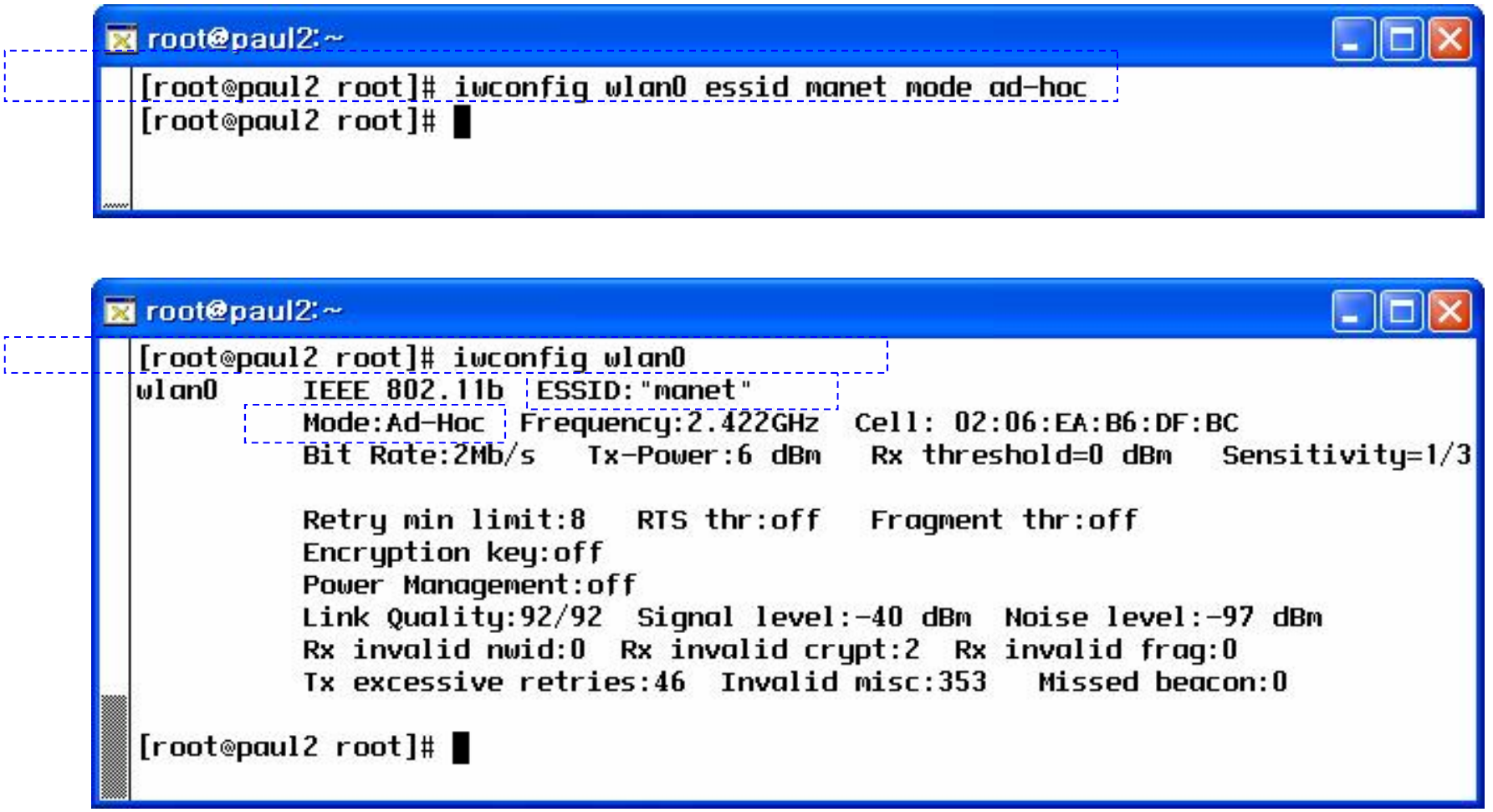

Thank you !

Appendix

ETRI MAC Filtering Tool

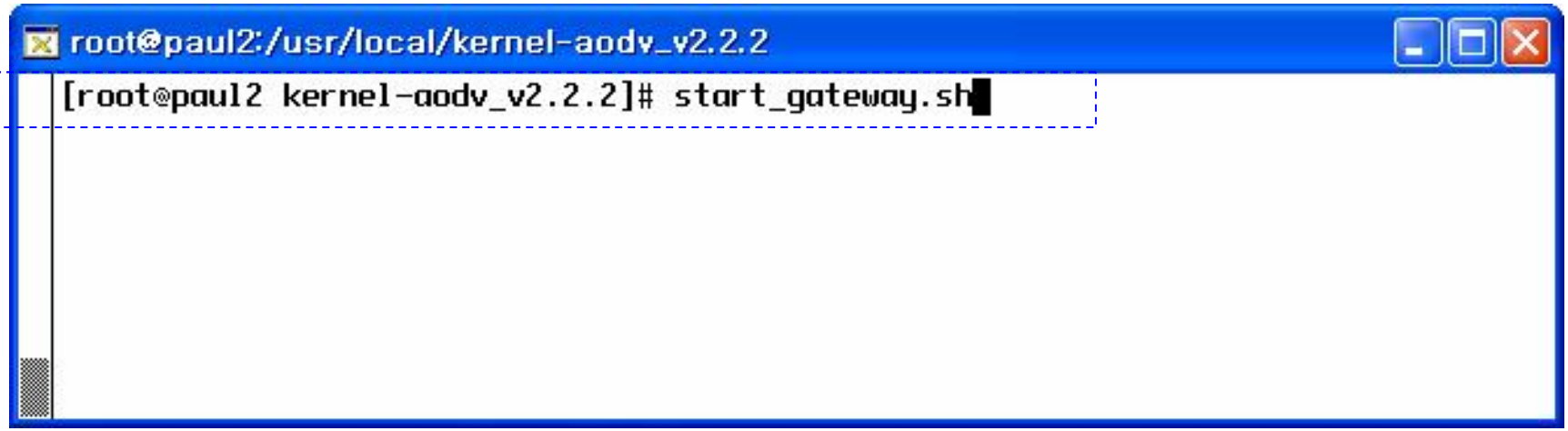
ETRI MAC Filtering Tool - 1

Configure Ad-hoc Mode with iwconfig

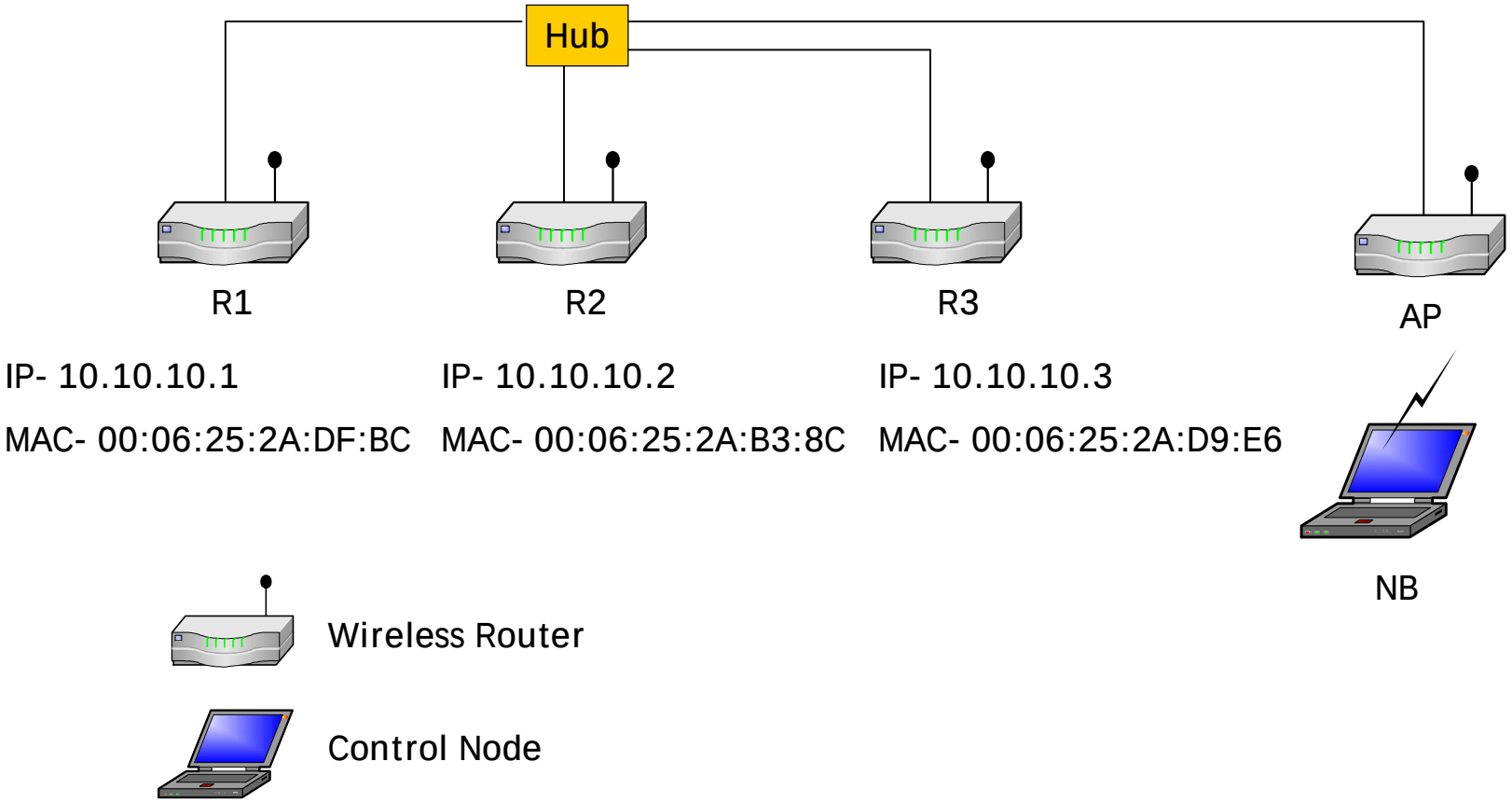


ETRI MAC Filtering Tool - 2

Start R1 as Gateway in AODV

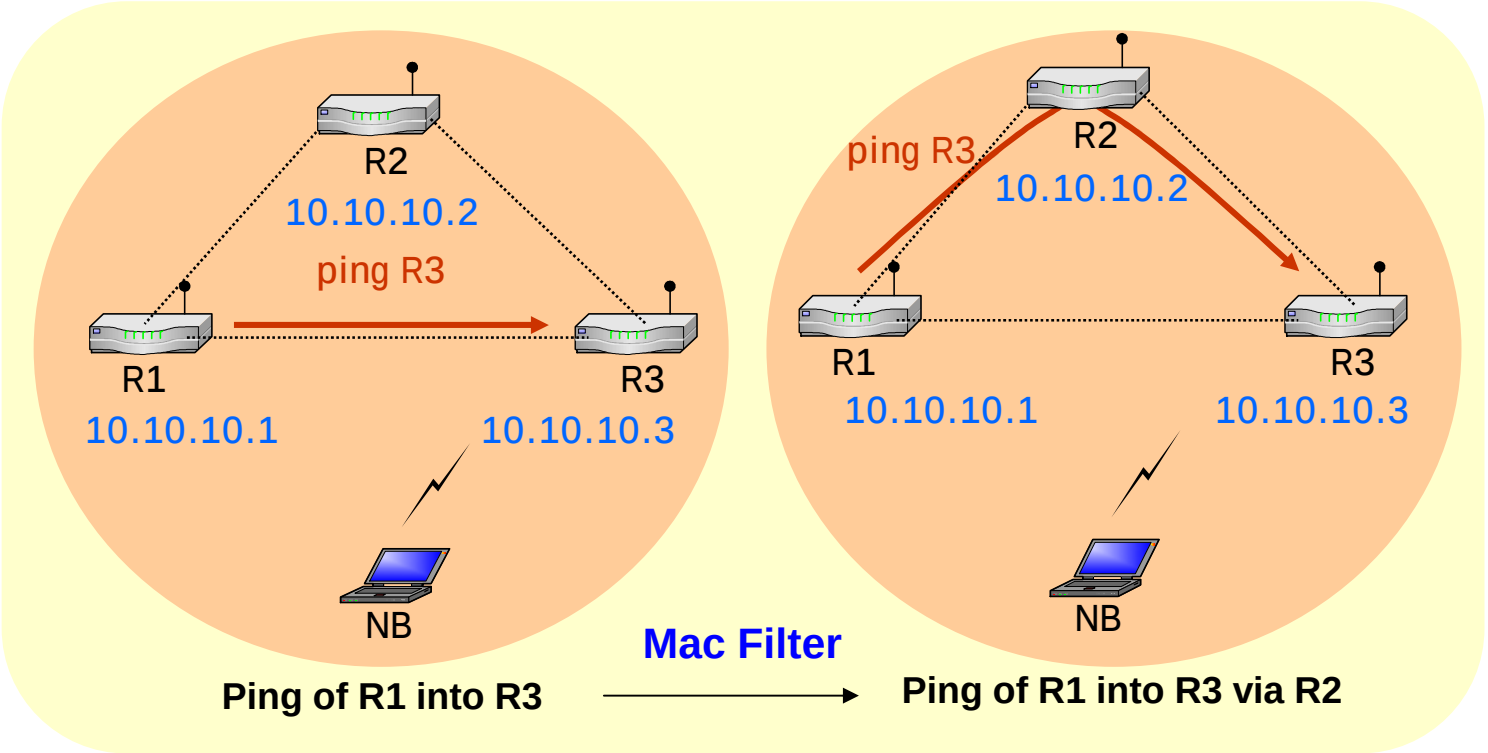


ETRI MAC Filtering Tool - 3
MANET Testbed



ETRI MAC Filtering Tool - 4

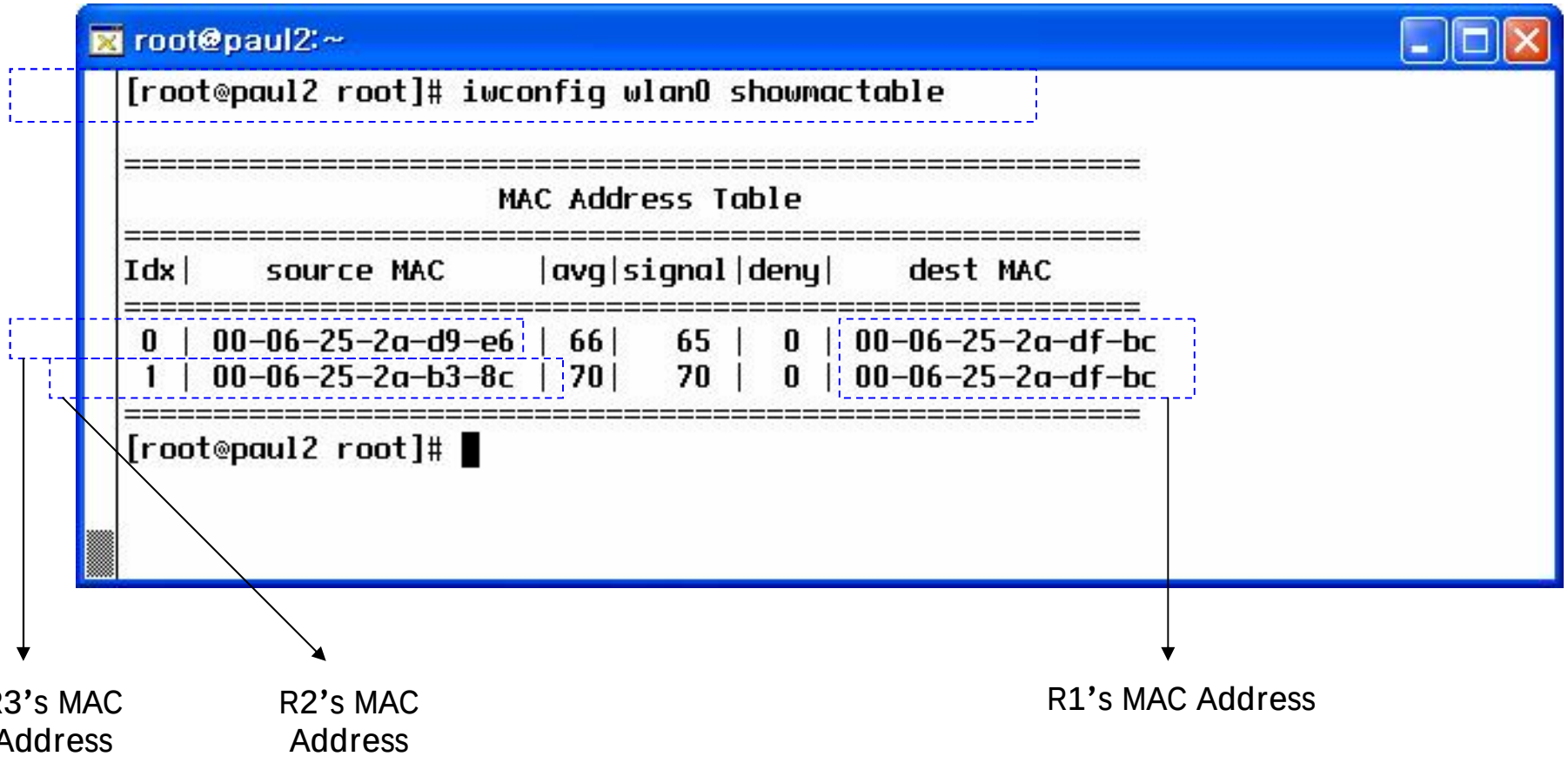
Test Scenario



Test Scenario

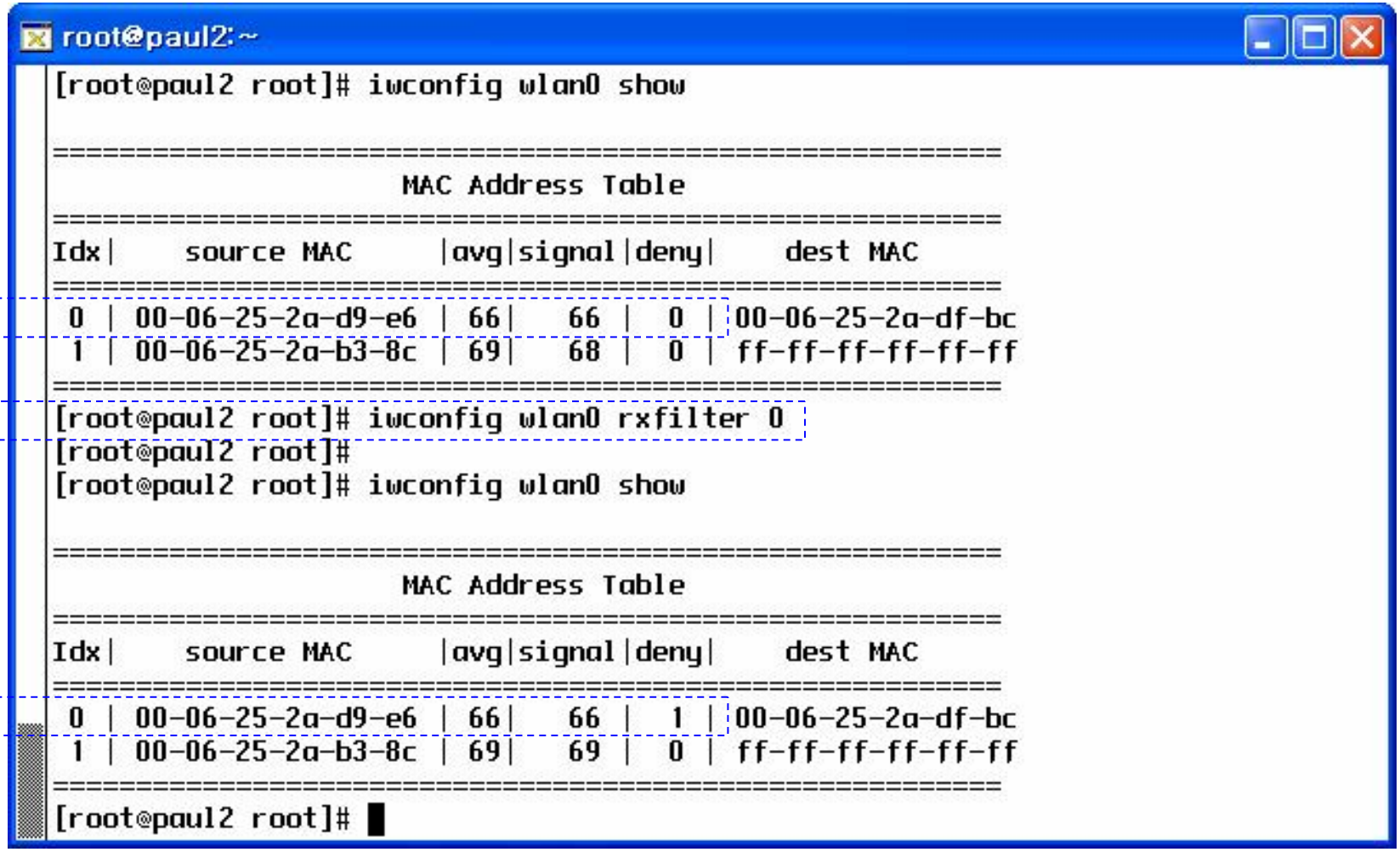
ETRI MAC Filtering Tool - 5

MAC Table



ETRI MAC Filtering Tool - 6

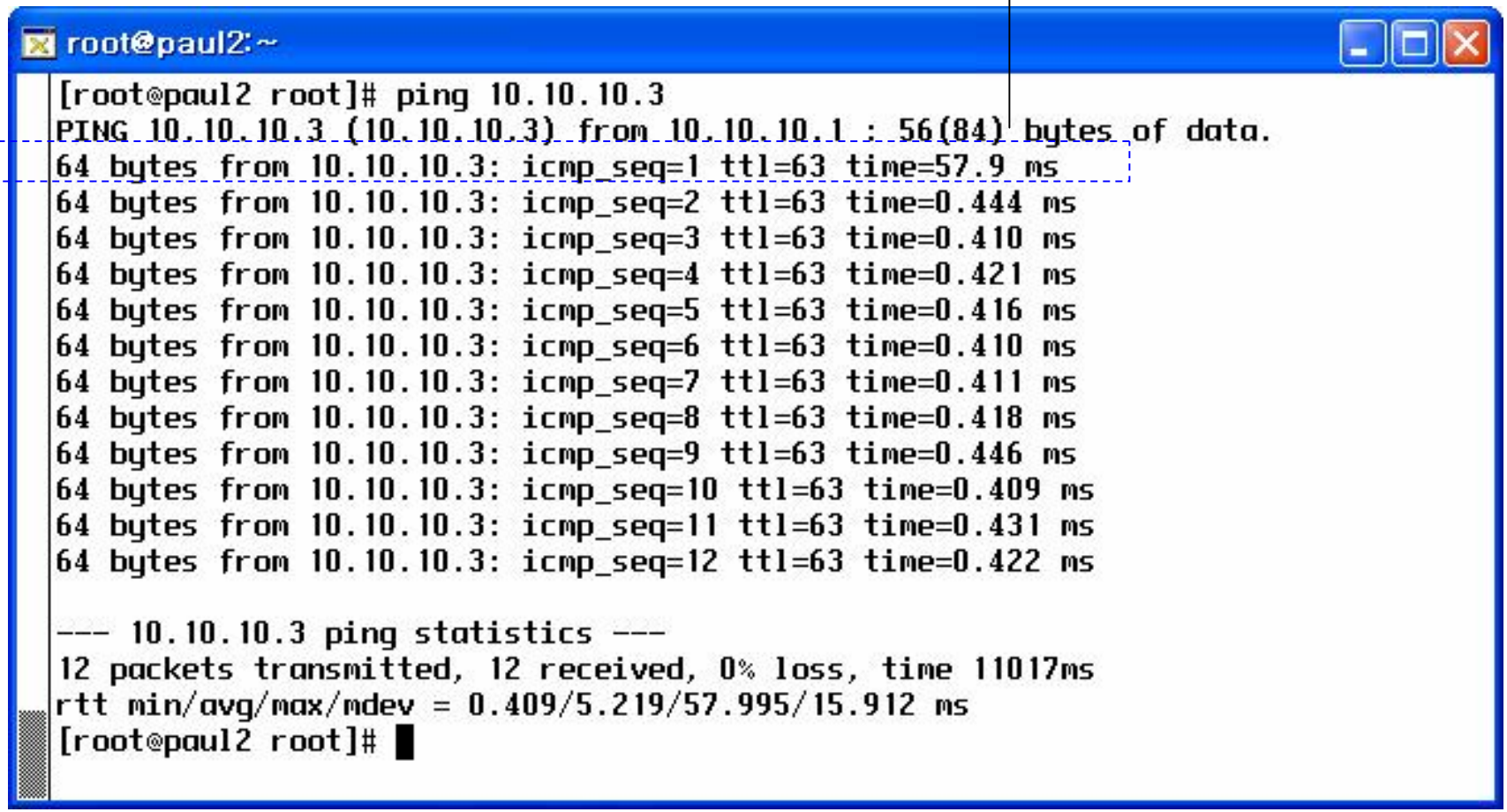
Set MAC Filtering at index 0 for R3



ETRI MAC Filtering Tool - 7

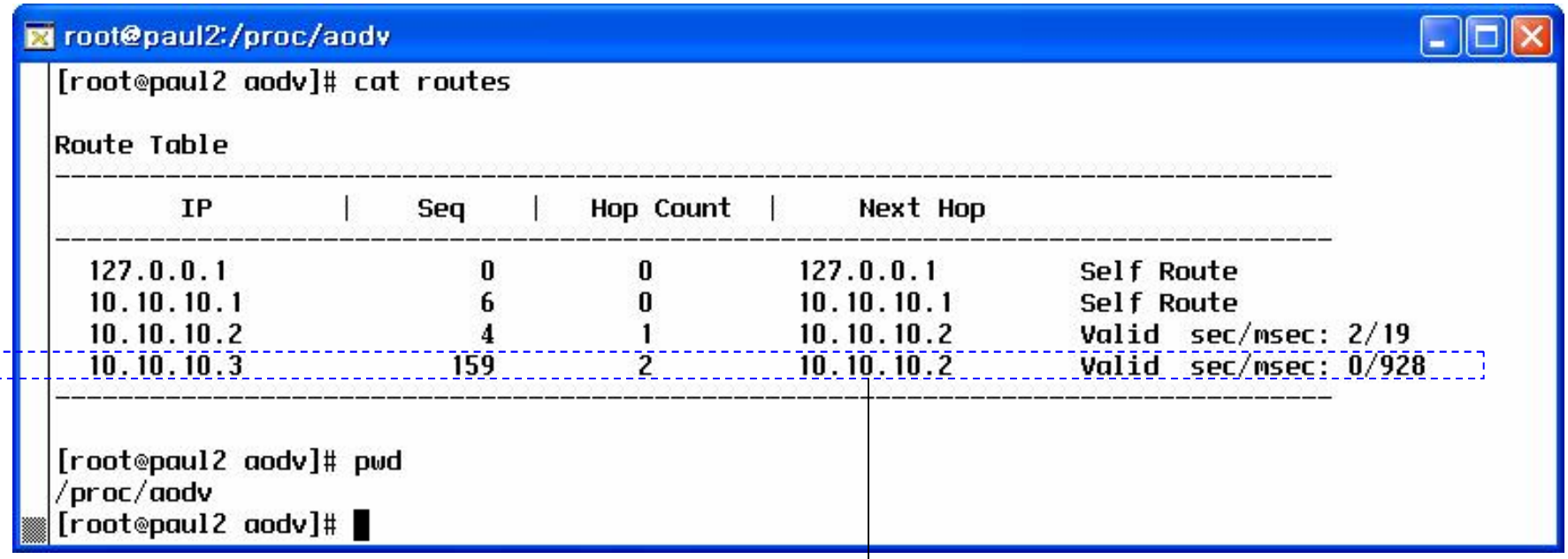
Ping from R1 to R3

Include Route Discovery Delay



ETRI MAC Filtering Tool - 8

Show AODV Routing Table in R1



When R1 sends data packets to R3, the next hop is R2.