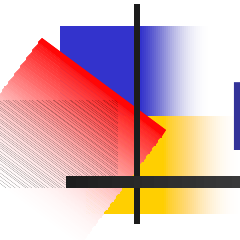


ENSC 833 Presentation

OPNET Simulation of CDPD MAC Layer

Eric Keung

and

Savio Lau

Overview

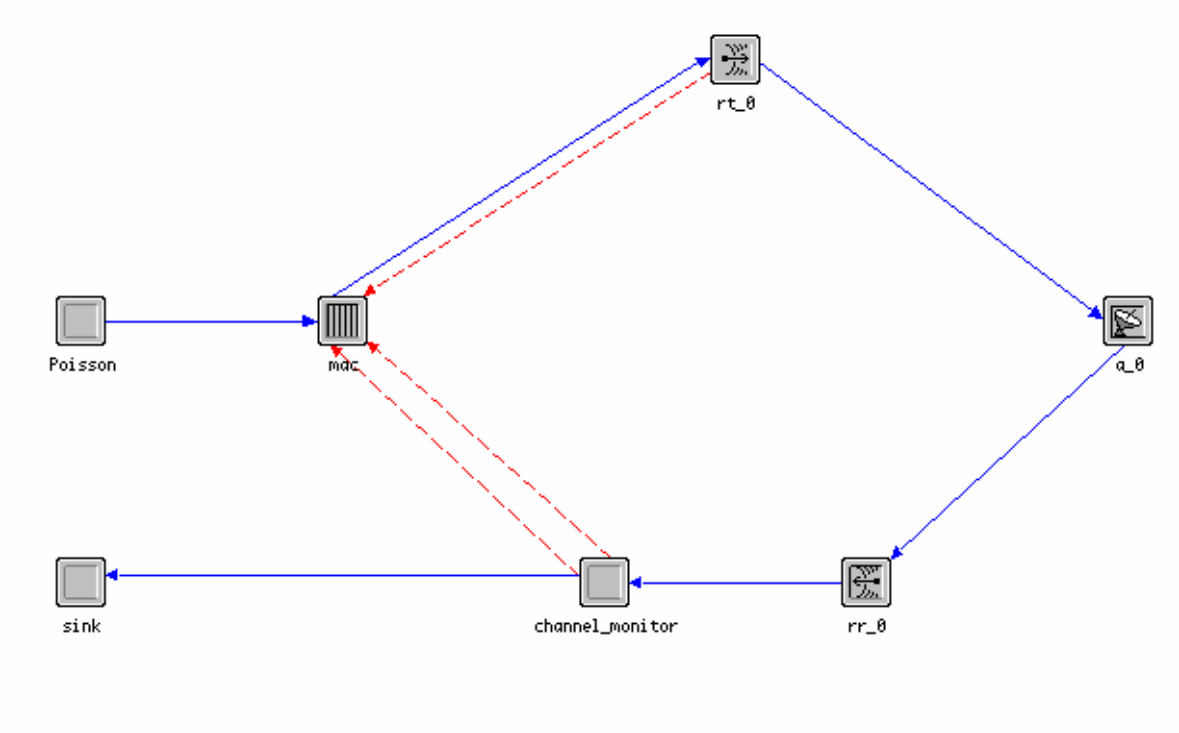
- Introduction
- Kernel procedures
- OPNET techniques
  - Declaring user attributes
  - Queue and Subqueues
  - Segmentation and Reassembly
  - Data encapsulation
  - Transmitter usage techniques
  - Receiver Pipeline models
  - Using traces
  - Animations
- Conclusion

Introduction

- Enhancement of the CDPD MAC model
- Features:
  - Frame segmentation and encapsulation into MAC layer blocks
  - Competing Mobile-End Stations (M-ES)
  - Burst uplink transmissions
  - Collision and channel error detection

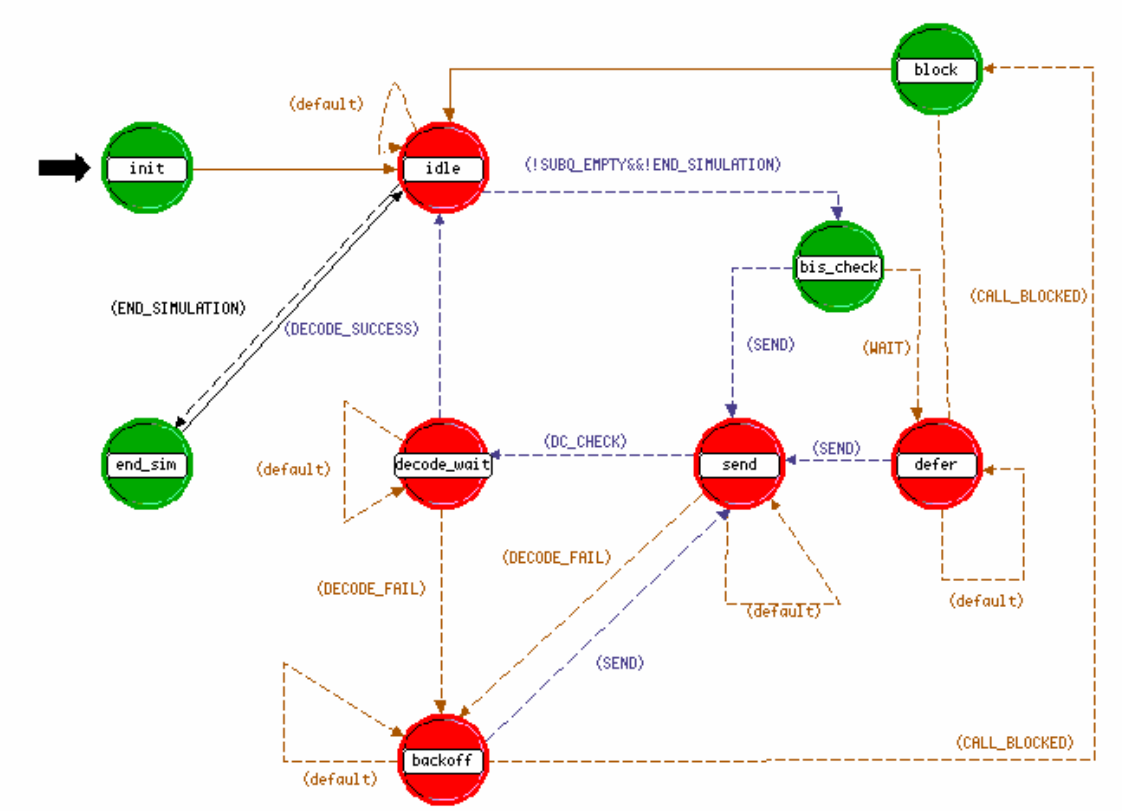
Introduction

- M-ES Node Model



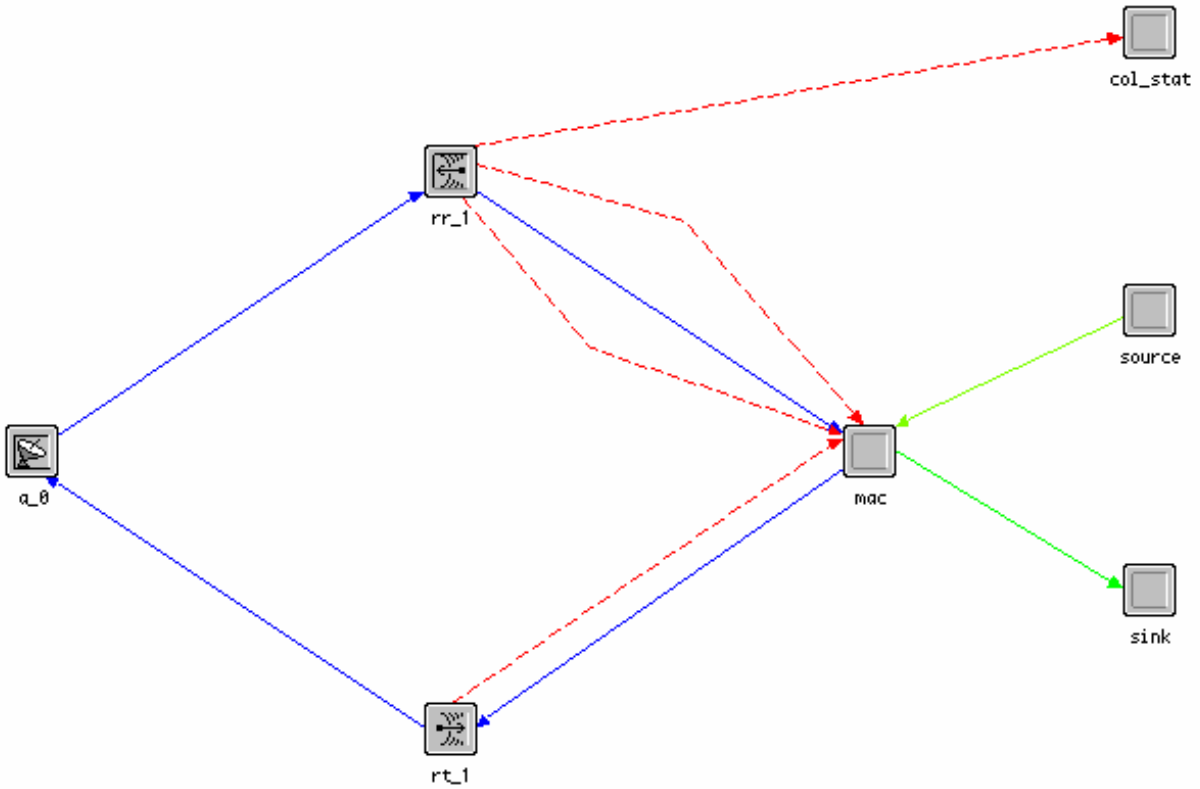
Introduction

- M-ES MAC Process Model



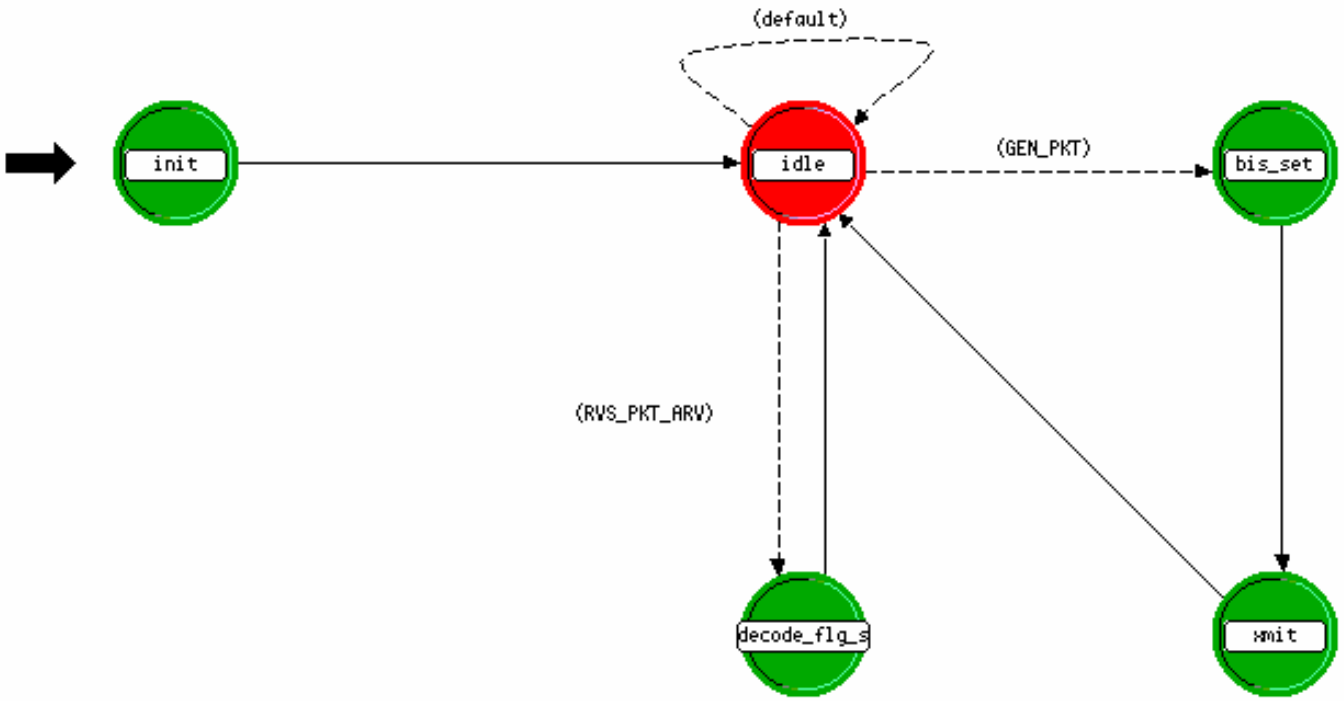
Introduction

- MDBS Node Model



Introduction

- MDBS MAC Model



Kernel Procedures

- OPNET predefined functions
- Subset of packages used in the CDPD model
  - Distribution Package (Dist)
  - Internal Model Access Package (Ima)
  - Interrupt Package (Intrpt)
  - Packet Package (Pk)
  - Programming package (Prg)
  - Queue (Q) and Subqueue (Subq) packages
  - Segmentation and Reassembly (Sar)

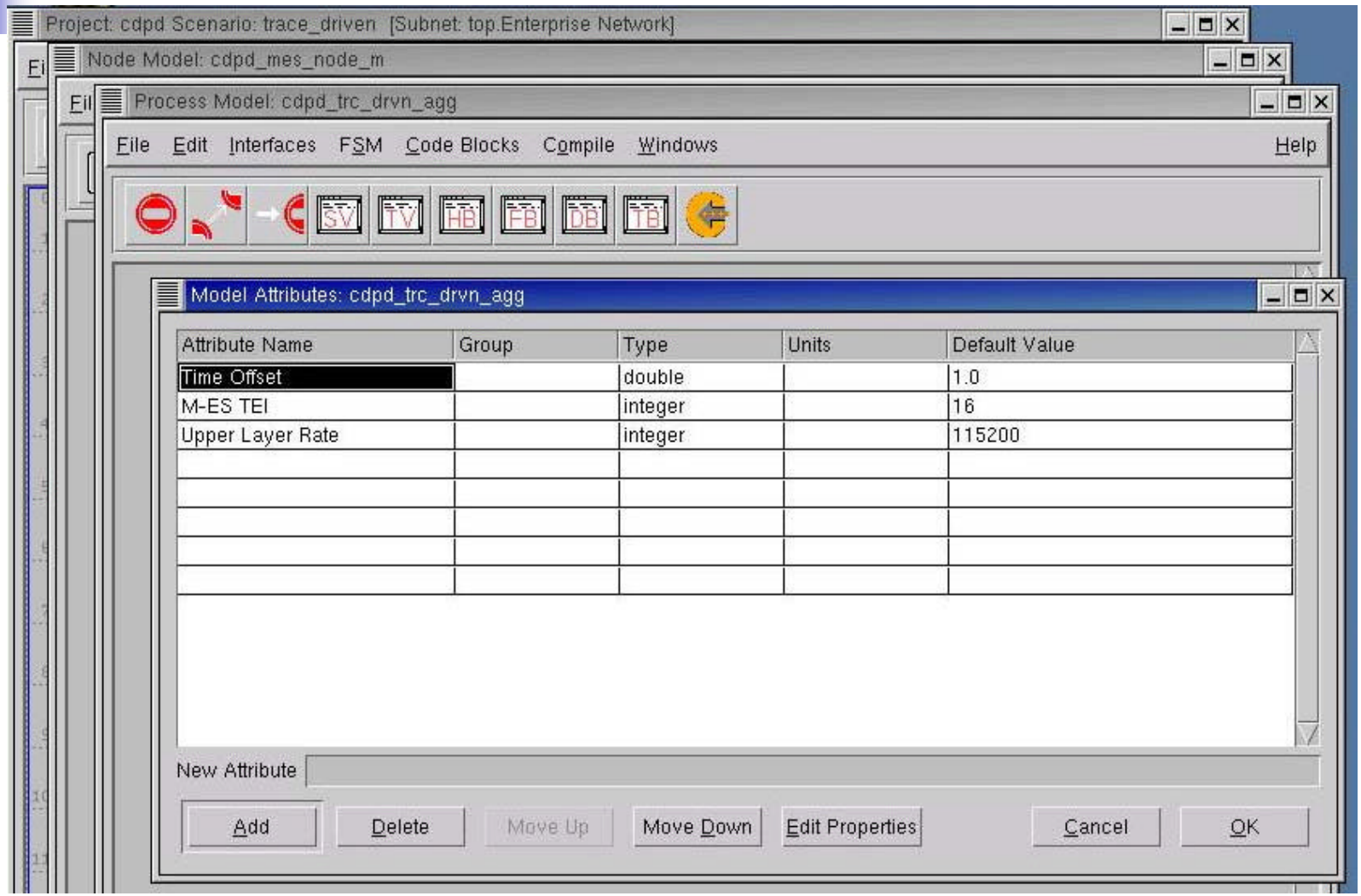
Kernel Procedures

- OPNET Symbolic Constants
  - Used with kernel packages
  - Meaningful name holders
    - Pointer to head and tail of queue
    - True / False OPNET constants
    - OPNET package options

Declaring User Attributes

- Users can define attributes during modeling or runtime (promoted)
- Example: changing channel data rate before starting simulation
- Code illustrating attribute access:
  - `my_id = op_id_self();`
  - `op_ima_obj_attr_get(my_id, “M-ES TEI”, &assigned_tei)`

Declaring User Attributes



Queue and Subqueues

- OPNET queues are initialized and defined once the queue icon is used

- Subqueue functions is a subset of queues
  - Priority streams
  - Streams from different sources (ex. control packets and data packets)
- CDPD M-ES MAC model use subqueue 0 to determine overall transmission delay

Queue and Subqueues

- Symbolic constants used with OPNET queues
  - `OPC_QPOS_HEAD`
  - `OPC_QPOS_TAIL`
- Some useful functions for queues:
  - `op_subq_pk_insert`
  - `op_subq_pk_remove`
  - `op_subq_pk_access`
  - `op_subq_stat`

Segmentation and Reassembly

- In the CDPD model, MDLP frames are segmented, inserted into MAC blocks and transmitted
- Reassembly back to MDLP frames occurs at the opposite end of the air interface
- Segmentation buffer is used when packets needs to be cut into smaller chunks
  - Used with encapsulating packets
- Reassembly of packets is automatic upon insertion of incomplete chunks

Segmentation and Reassembly

- Segmentation can be cascaded – so chucks of chucks can be tracked
- Resizing of chunks can be deal with using resegmentation buffers
- In OPNET, once a SAR buffer is declared its function cannot be changed
- A SAR buffer can be declared like this:
  - `seg_buf_ptr = op_sar_buf_create (OPC_SAR_BUF_TYPE_SEGMENT, OPC_SAR_BUF_OPT_DEFAULT);`

Segmentation and Reassembly

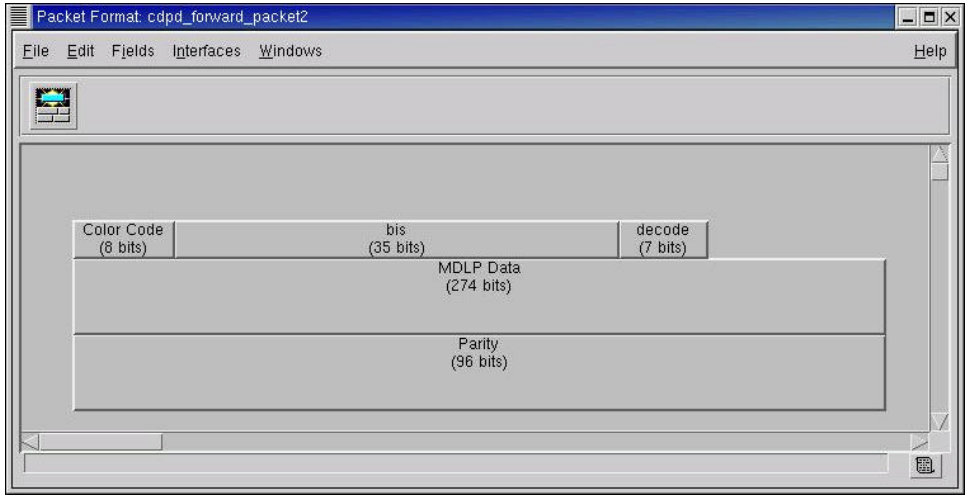
- Functions for insertion to and removal from SAR buffers is type dependent
  - Segmentation and Resegmentation
    - `op_sar_segbuf_pk_insert`
    - `op_sar_rsgbuf_seg_insert`
    - `op_sar_srcbuf_seg_remove`
  - Reassembly
    - `op_sar_rsmbuf_seg_insert`
    - `op_sar_rsmbuf_pk_remove`

Segmentation and Reassembly

- OPNET caveats
  - All of the pointer variables passed with SAR package must be state variables
  - Temporary variables results in unowned or non-referenced packets
  - May be extended to other KPS but unconfirmed

Data Encapsulation

- Data encapsulation occurs often between different protocol layers (ex. IP and MAC)
- OPNET provides custom packet formats
- CDPD downlink channel packet example:



Data Encapsulation

- In the case of CDPD model, frames up to 1088 bits long are cut into 274 bits chunks
- Example code:
  - `blk_ptr = op_sar_srcbuf_remove (seg_buf_ptr, 274);`
  - `send_pkptr = op_pk_create_format (“cdpd forward packet2”);`
  - `op_pk_nfd_set(send_ptr, “MDLP Data”, blk_ptr);`

Data Encapsulation

- Example code continued:
  - `op_pk_nfd_set(send_pkptr, “bis”, bis);`
  - `op_pk_nfd_set(send_pkptr, “decode”, decode);`
  - `op_pk_send(send_pkptr, FWD_OUTSTRM);`
- In other words, a MDLP frame segment is put into the “MDLP Data” of the CDPD downlink channel packet and send to the transmitter
- Multiple layers of “packet-in-packet” can be used in OPNET

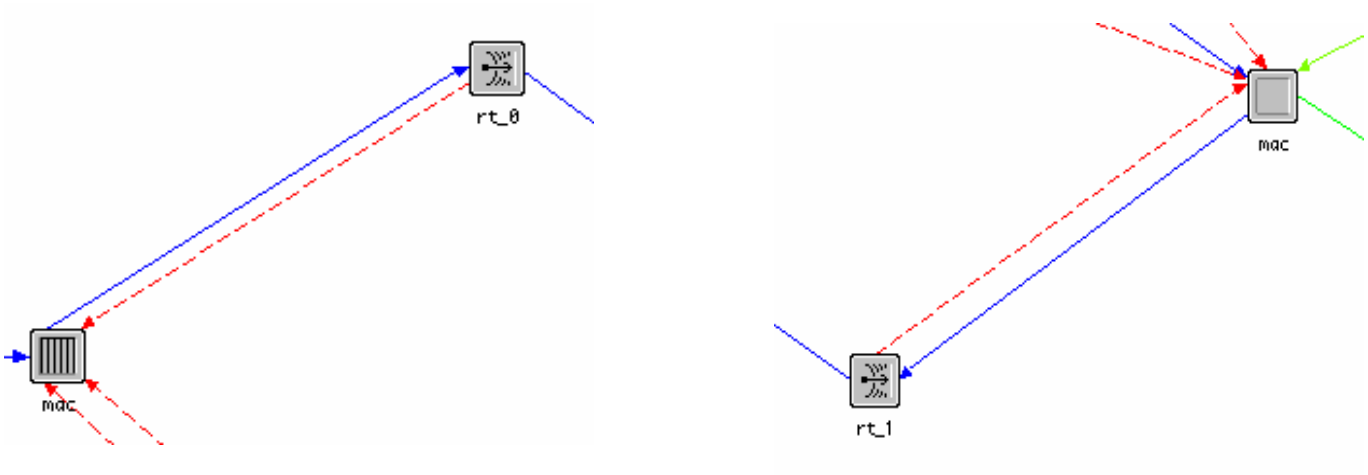
Transmitter Usage Techniques

- Each Transmitter has its own simple FIFO queue
- Having a large number of packets residing in this queue is not good practice
  - All packets in the transmitters are sent before the channel is released even after collision is notified
  - No preemption possible for waiting packets in priority type transmission

Transmitter Usage Techniques

- Thus, it is good practice to keep at most one packet in the transmitter
- The trick to make certain the transmitter is free when transmitting a packet is to monitor the transmitter “busy” status
  - Make a statistical wire that generates an interrupt on a downward edge (from on to off)
  - Use the above signal as an indicator to send another packet

Transmitter Usage Techniques



Statistical Wire

Packet Stream

Receiver Pipeline Models

- Each OPNET receiver has a pipeline to determine the quality of the receiving packet
- It includes modulation type, receiver gain model, power model, background noise model, interference noise model, snr (signal-to-noise ratio) model, ber (bit error rate) model, and ecc (error correction code) model
- Only radio receivers has all of the above, point-to-point and bus receivers have a subset

Receiver Pipeline Models

- Two types of standard models:
  - None
  - Basic
- Custom models can be made
- In CDPD model, the modulation type is set to QPSK, an approximation to the actual GMSK
- A custom background noise model was made to add Raleigh channel fading noise

Receiver Pipeline Models

- The basic error correction model is used to approximate Reed Solomon Forward Error Correction (FEC) encoding
  - *2t+1* bits of FEC code can guarantee to correct *t* errors
  - So 96 bits of error correction code can correct up to 95/2 = 47.5 bits of errors
  - That gives a “ecc threshold” of 47.5/378 = 0.1256

Using Traces

- Traffic trace is very useful in simulations
- Sanitized trace in delimited form works well in OPNET
- Example Format:

Using Traces

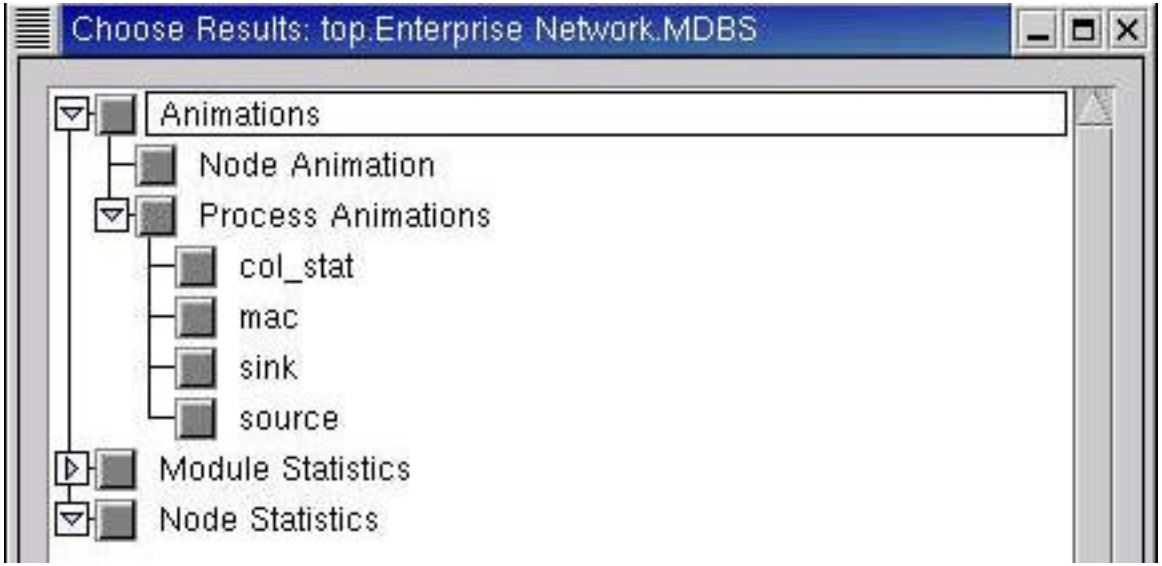
- Traces can be accessed via function:
  - `line_list_ptr = op_prg_gdf_read (“sierra”);`
- A line can be read with:
  - `field_list_ptr = op_prg_str_decomp (op_prg_list_access(line_list_ptr, line_index);`
- A field can be read with:
  - `tei = atoi (op_prg_list_access (field_list_ptr, 2));`

Animations

- OPNET includes a built-in animation tool to capture Node and Process model activities
- Custom models can also be made using the Animation (Anim) Package Kernel Procedures
  - Very useful in debugging – to figure out where the finite state machine (FSM) is stuck
  - Very useful as a visual tool to show the FSM in action

Animations

- Select through the “Choose Individual Statistics” window under “Animations” in Scenario Editor



Conclusion

- OPNET has a lot of little used but powerful functions
- The OPNET kernel procedures save simulation time and development effort
- A lot of techniques and tricks can be used to make the model work as desired
- Play with attributes and read their comments