Simplex Control and Message/ACK Exchange

- Module 6 Final Presentation

Le Wang: lewang@wpi.edu

Renato Iida: rfiida@wpi.edu

Rebecca Cooper: rrcooper@wpi.edu

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Purpose

- Establishing end-to-end network resource allocation management in simplex mode.

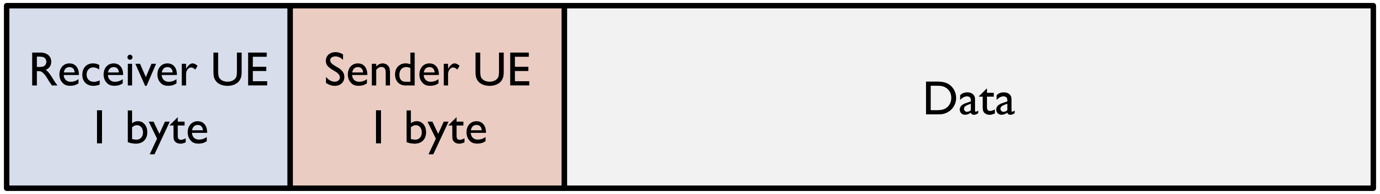
- Exchanging of control information and message/ACK forwarding between the UEs and the BS units.

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Building the MAC Frame

- The package format must be small enough that it is easy to handle
- It must also convey useful information
  - Receiver UE
  - Sender UE
  - Data

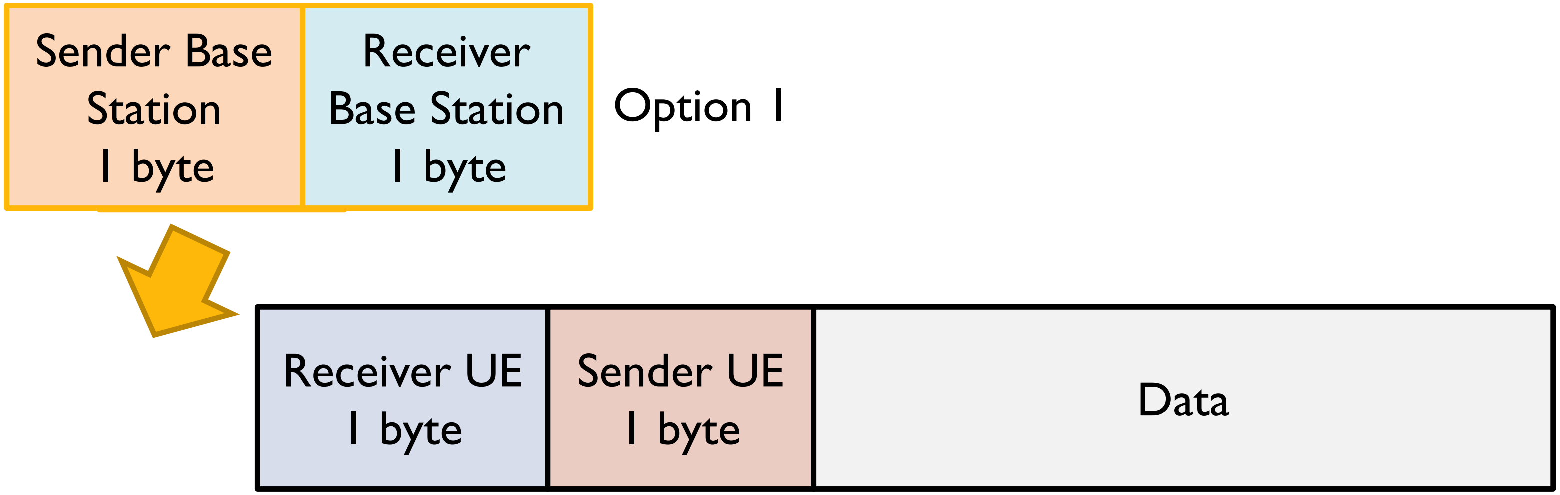


Receiver UE 1 byte	Sender UE 1 byte	Data

Building the MAC Frame

The Routing between Base Stations:

- Transmit the entire routing path
  - This is very long
  - It requires additional processing at the Base Station



Sender Base Station
1 byte

Receiver Base Station
1 byte

Option 1

Receiver UE
1 byte

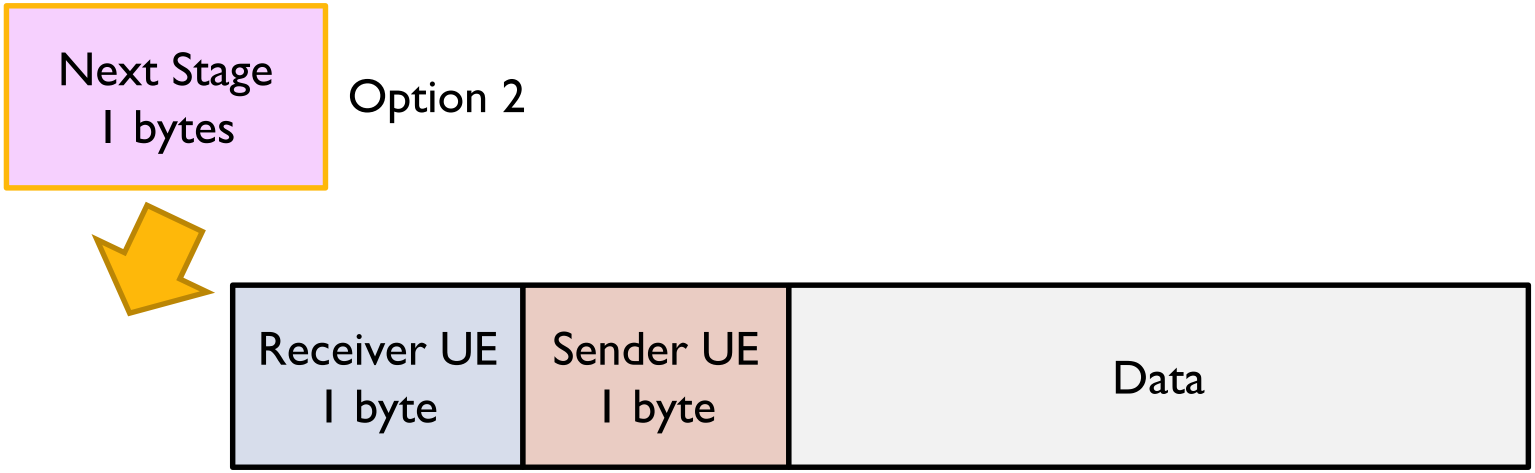
Sender UE
1 byte

Data

Building the MAC Frame

The Routing between Base Stations:

- Transmit the next stage in the path
  - The Base Station must change the value of this field every time



Next Stage
1 bytes

Option 2

Receiver UE
1 byte

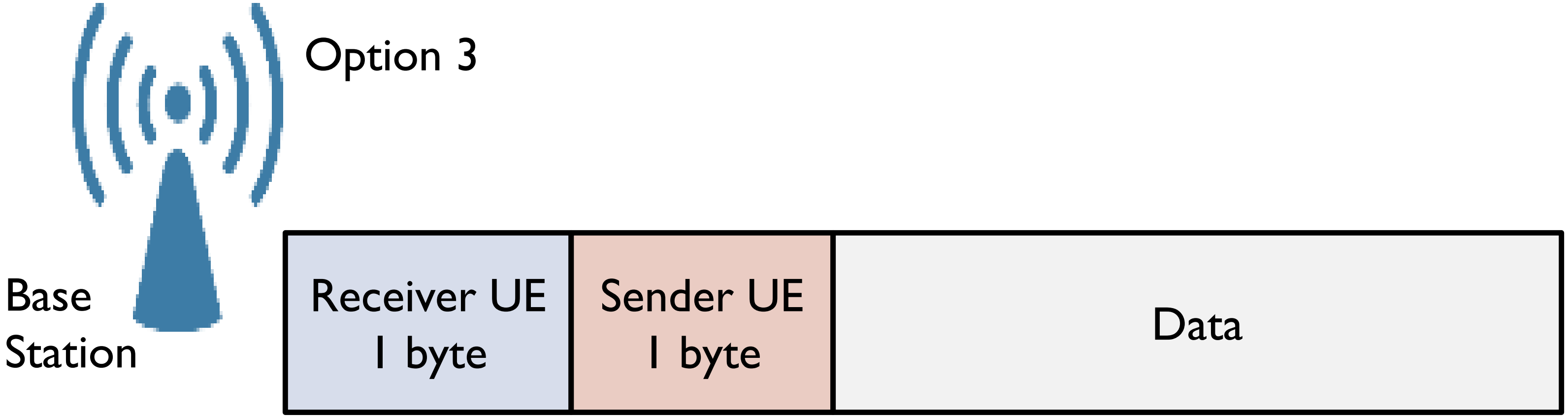
Sender UE
1 byte

Data

Building the MAC Frame

The Routing between Base Stations:

- Read the receiver at the Base Station and determine the routing to destination
  - Easy to understand
  - The header stays small



Option 3

Base Station

Receiver UE 1 byte	Sender UE 1 byte	Data

Building the MAC Frame

Frame Type:

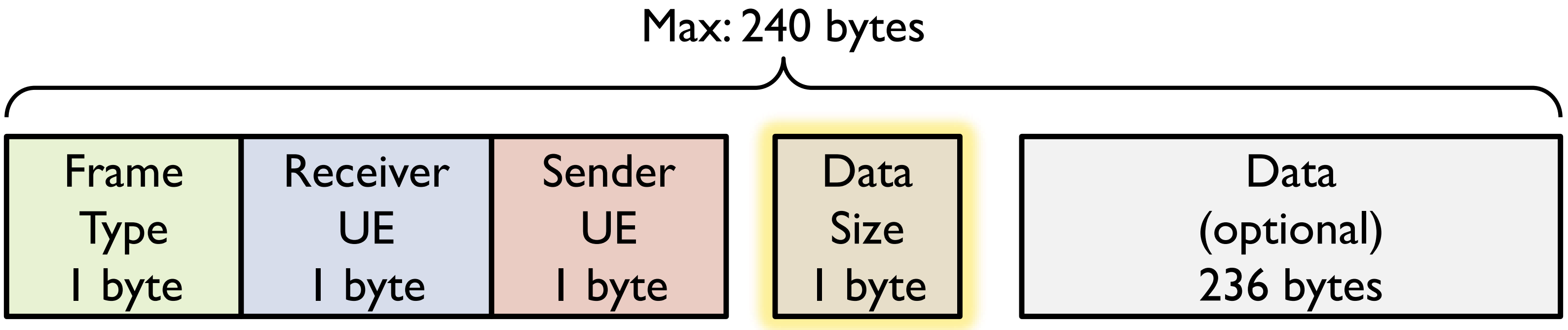
- Differentiates between the uses of the MAC frames
- The INVALID frame type is used to drop corrupt frames
- Also helps with error detection through hamming distance
  - DATA 1111 0000 = 240
  - ACK 1111 1111 = 255
  - INVALID 0000 0000 = 0

Frame Type 1 byte	Receiver UE 1 byte	Sender UE 1 byte	Data (optional)

Building the MAC Frame

Data Size:

- Set a maximum size for all MAC frames of 240 bytes or 1920 bits
- Transmitting the data size allows us to cut off padding added by the physical layers



Max: 240 bytes

Frame Type 1 byte

Receiver UE 1 byte

Sender UE 1 byte

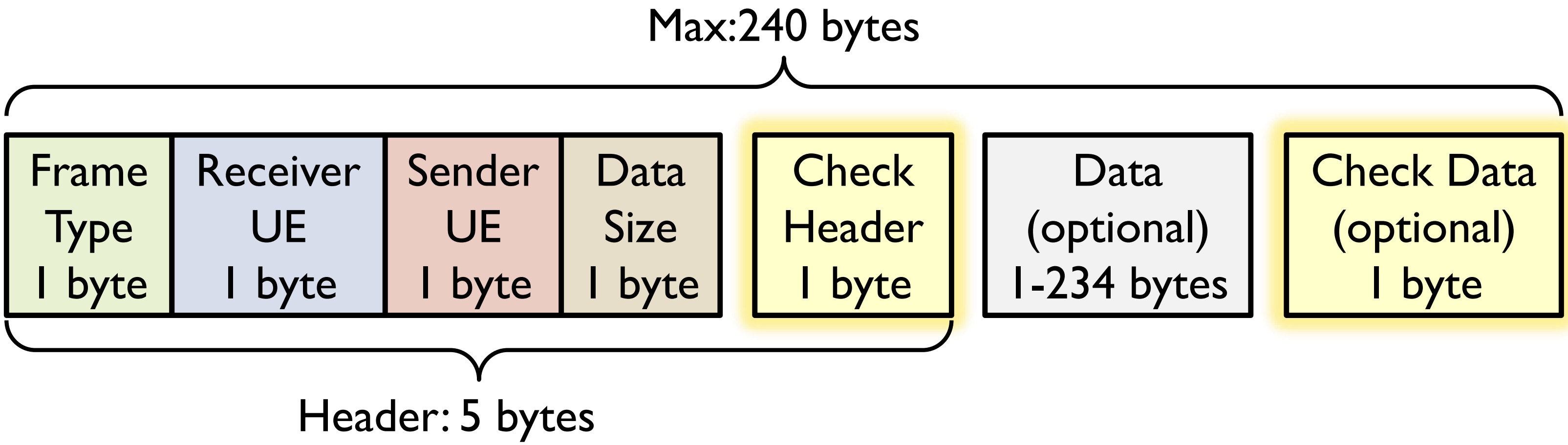
Data Size 1 byte

Data (optional) 236 bytes

Building the MAC Frame

CRC:

- We must be sure that the Data is transmitted without errors
- CRC-8 to check the integrity of the bits
  - simple and fast but does not correct errors.
- Team 4 needs to be sure that the header is correct



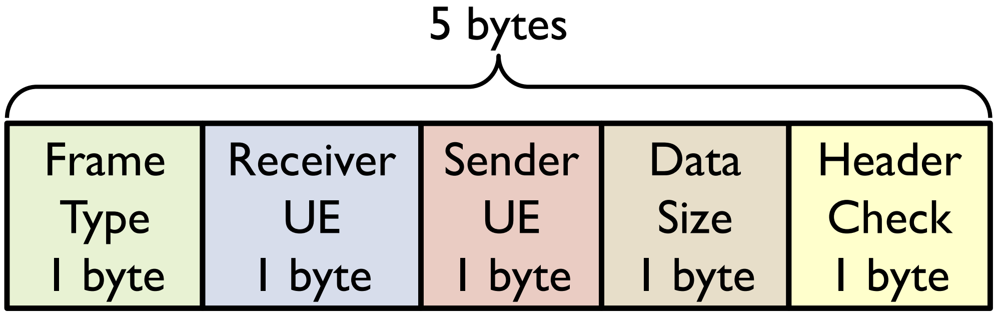
Max:240 bytes

Frame Type 1 byte | Receiver UE 1 byte | Sender UE 1 byte | Data Size 1 byte | Check Header 1 byte | Data (optional) 1-234 bytes | Check Data (optional) 1 byte

Header: 5 bytes

Building the MAC Frame

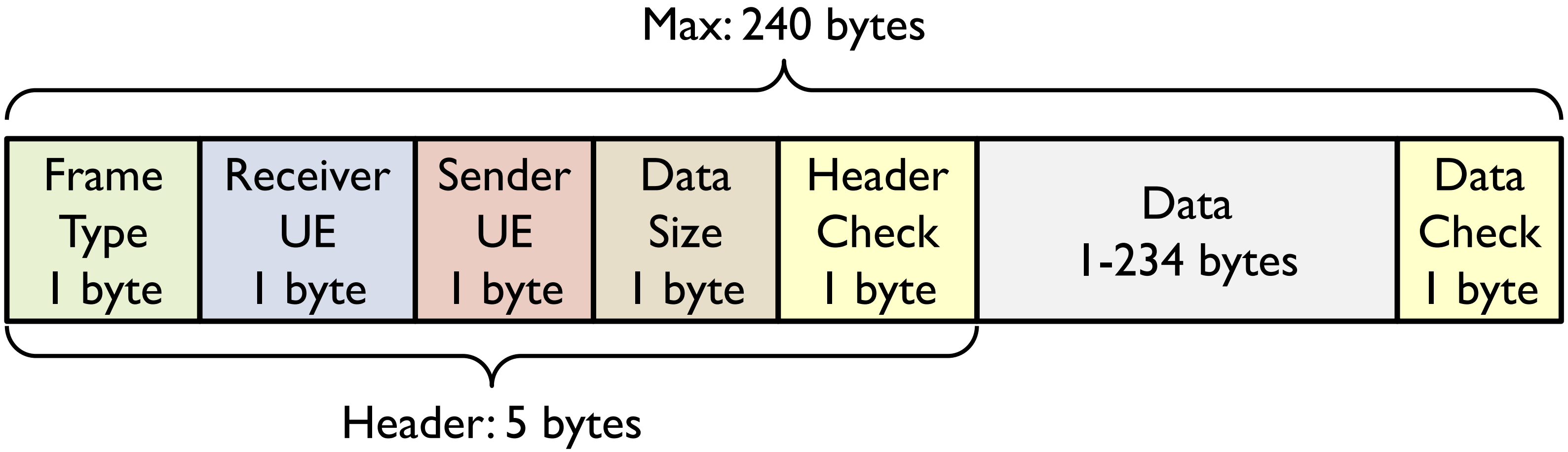
ACK Frame:



5 bytes

Frame Type 1 byte	Receiver UE 1 byte	Sender UE 1 byte	Data Size 1 byte	Header Check 1 byte

Data Frame:



Max: 240 bytes

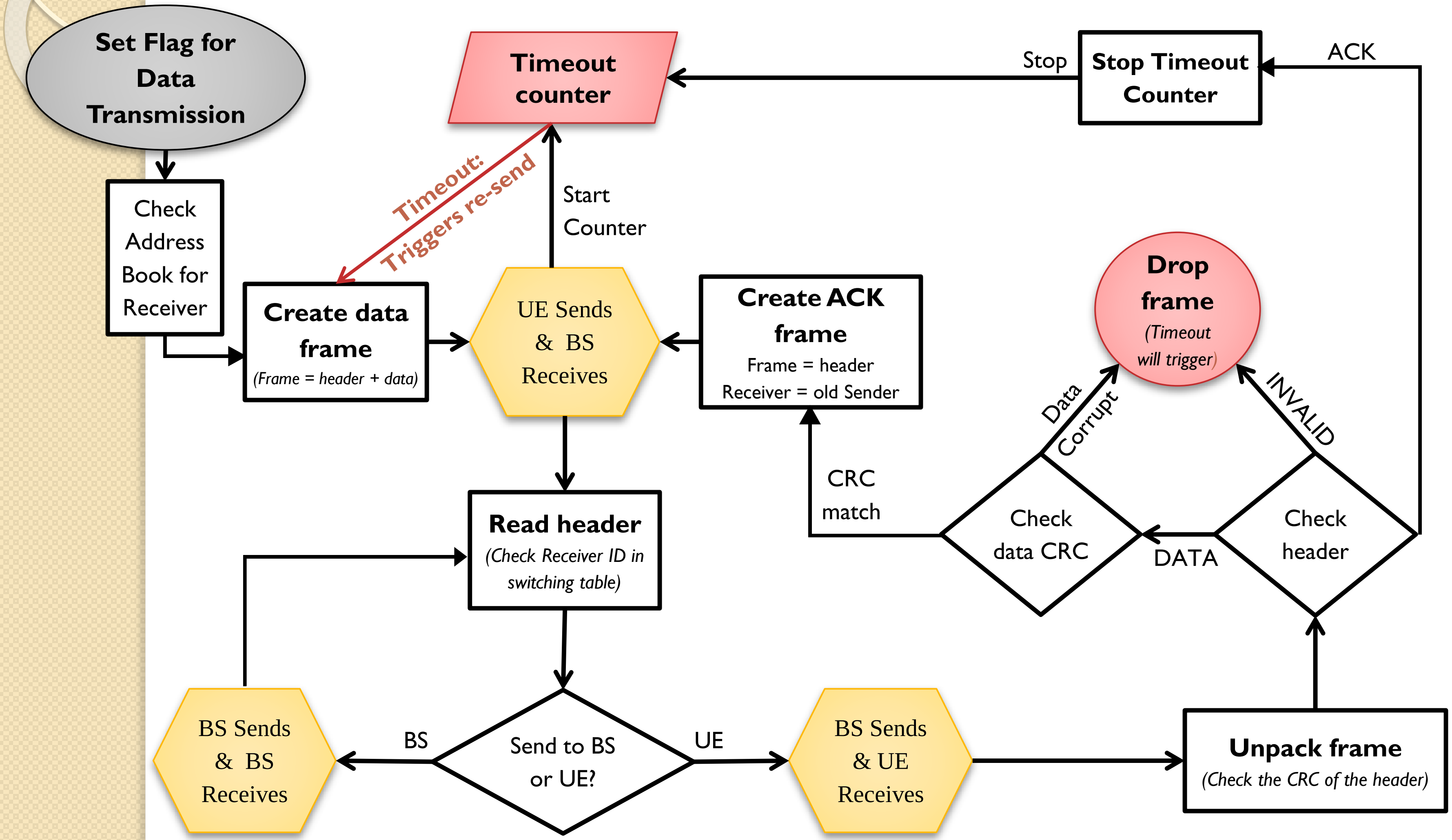
Frame Type 1 byte	Receiver UE 1 byte	Sender UE 1 byte	Data Size 1 byte	Header Check 1 byte	Data 1-234 bytes	Data Check 1 byte

Header: 5 bytes

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

State Machine for Frame Transmission

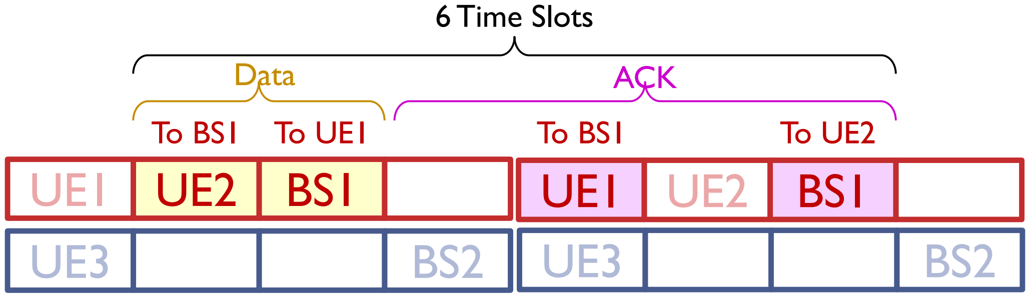


Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

ACK Time Out

Shortest message
UE2 to UE1



6 Time Slots

Data

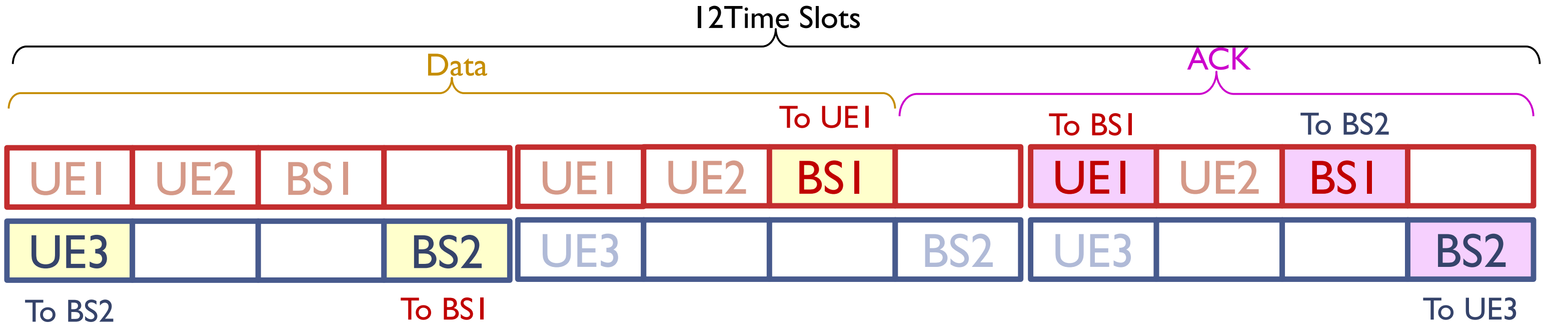
ACK

To BS1 To UE1

To BS1 To UE2

UE1	UE2	BS1		UE1	UE2	BS1	
UE3			BS2	UE3			BS2

Longest message
UE3 to UE1



12Time Slots

Data

ACK

To UE1

To BS1 To BS2

UE1	UE2	BS1		UE1	UE2	BS1		UE1	UE2	BS1	
UE3			BS2	UE3			BS2	UE3			BS2

To BS2 To BS1 To UE3

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Other Features Considered

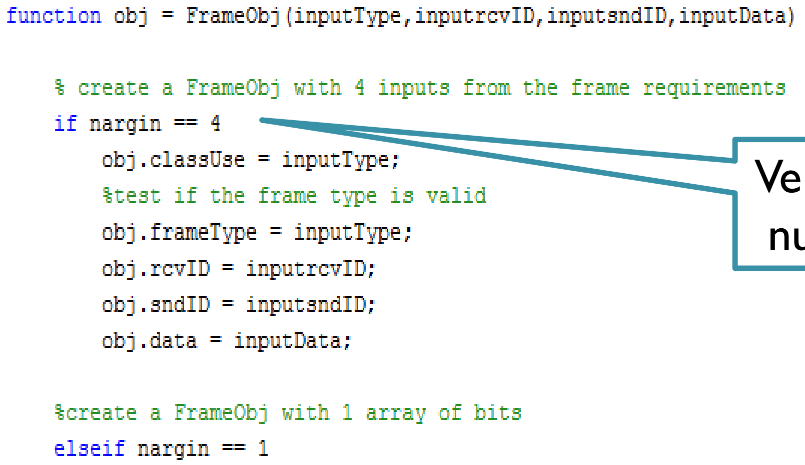
- Sequence number
  - Sliding window
  - Group ACK
- NACK

Outline

- Purpose
- Data Transmission in the MAC Layer
    - Building the MAC Frame
    - State Machine for the Frame Transmission
    - ACK Timeout
    - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Implementation
Constructor

Constructor method



```
function obj = FrameObj(inputType,inputrcvID,inputsndID,inputData)

    % create a FrameObj with 4 inputs from the frame requirements
    if nargin == 4
        obj.classUse = inputType;
        %test if the frame type is valid
        obj.frameType = inputType;
        obj.rcvID = inputrcvID;
        obj.sndID = inputsndID;
        obj.data = inputData;

    %create a FrameObj with 1 array of bits
    elseif nargin == 1
```

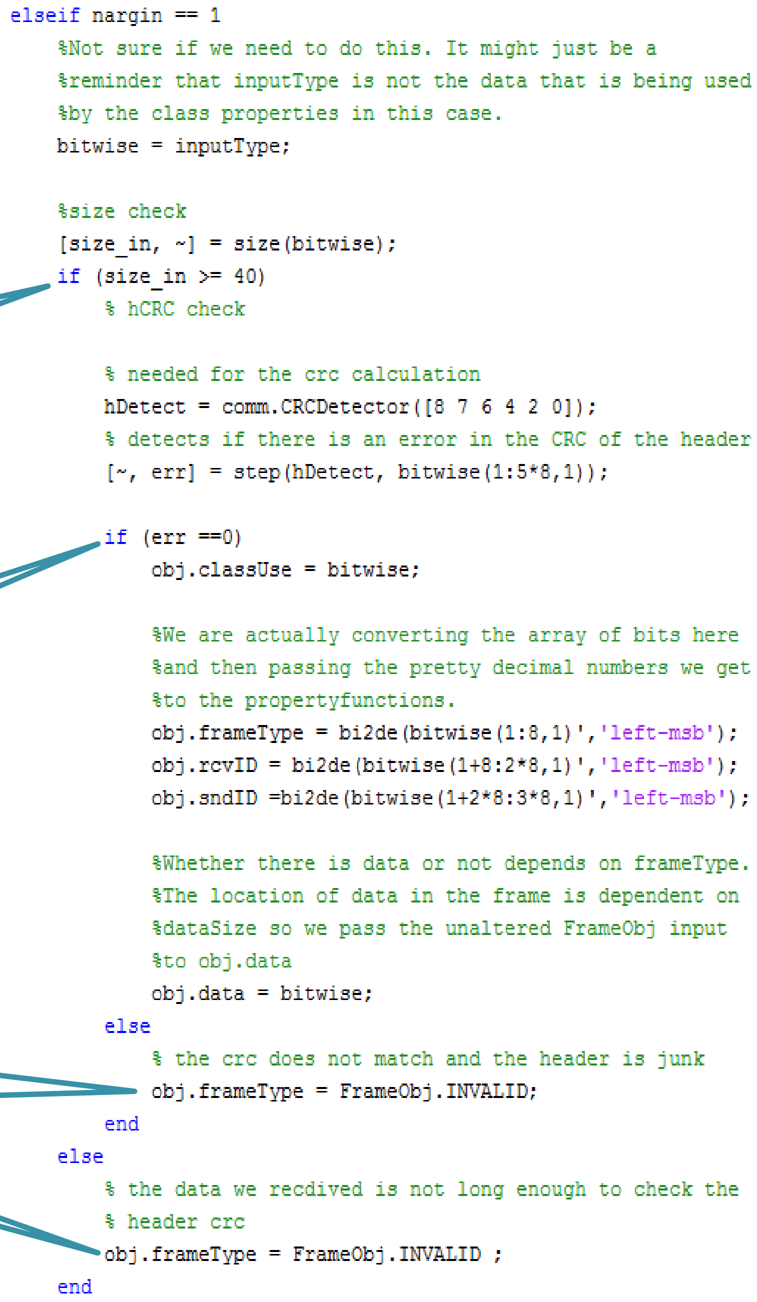
Verification of the number of inputs

Implementation Constructor

Check for the minimum size

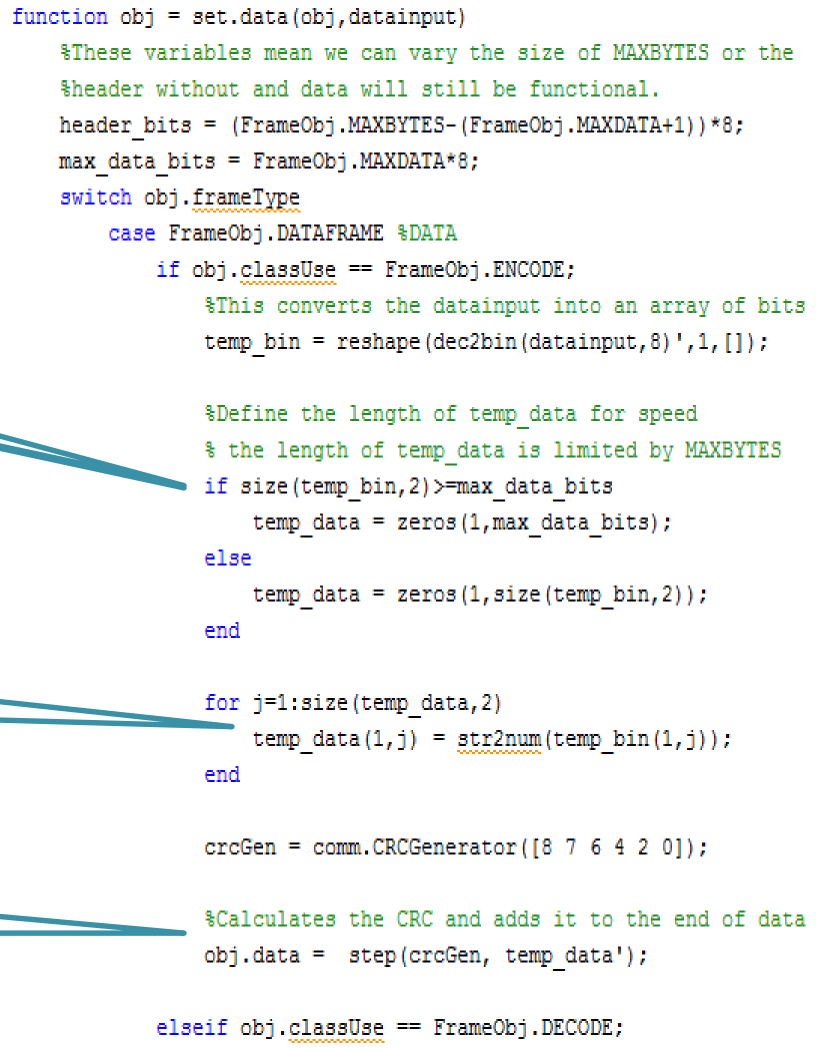
CRC check the header.

Failed checks result in INVALID frame types



```
elseif nargin == 1
    %Not sure if we need to do this. It might just be a
    %reminder that inputType is not the data that is being used
    %by the class properties in this case.
    bitwise = inputType;

    %size check
    [size_in, ~] = size(bitwise);
    if (size_in >= 40)
        % hCRC check

        % needed for the crc calculation
        hDetect = comm.CRCDetector([8 7 6 4 2 0]);
        % detects if there is an error in the CRC of the header
        [~, err] = step(hDetect, bitwise(1:5*8,1));

        if (err ==0)
            obj.classUse = bitwise;

            %We are actually converting the array of bits here
            %and then passing the pretty decimal numbers we get
            %to the propertyfunctions.
            obj.frameType = bi2de(bitwise(1:8,1)','left-msb');
            obj.rcvID = bi2de(bitwise(1+8:2*8,1)','left-msb');
            obj.sndID =bi2de(bitwise(1+2*8:3*8,1)','left-msb');

            %Whether there is data or not depends on frameType.
            %The location of data in the frame is dependent on
            %dataSize so we pass the unaltered FrameObj input
            %to obj.data
            obj.data = bitwise;
        else
            % the crc does not match and the header is junk
            obj.frameType = FrameObj.INVALID;
        end
    else
        % the data we recdived is not long enough to check the
        % header crc
        obj.frameType = FrameObj.INVALID ;
    end
```

Implementation
Data field



```
function obj = set.data(obj,datainput)
    %These variables mean we can vary the size of MAXBYTES or the
    %header without and data will still be functional.
    header_bits = (FrameObj.MAXBYTES-(FrameObj.MAXDATA+1))*8;
    max_data_bits = FrameObj.MAXDATA*8;
    switch obj.frameType
        case FrameObj.DATAFRAME %DATA
            if obj.classUse == FrameObj.ENCODE;
                %This converts the datainput into an array of bits
                temp_bin = reshape(dec2bin(datainput,8)',1,[]);

                %Define the length of temp_data for speed
                % the length of temp_data is limited by MAXBYTES
                if size(temp_bin,2)>=max_data_bits
                    temp_data = zeros(1,max_data_bits);
                else
                    temp_data = zeros(1,size(temp_bin,2));
                end

                for j=1:size(temp_data,2)
                    temp_data(1,j) = str2num(temp_bin(1,j));
                end

                crcGen = comm.CRCGenerator([8 7 6 4 2 0]);

                %Calculates the CRC and adds it to the end of data
                obj.data =  step(crcGen, temp_data');

            elseif obj.classUse == FrameObj.DECODE;
```

Maximum size of data field

Mapping chars to bits

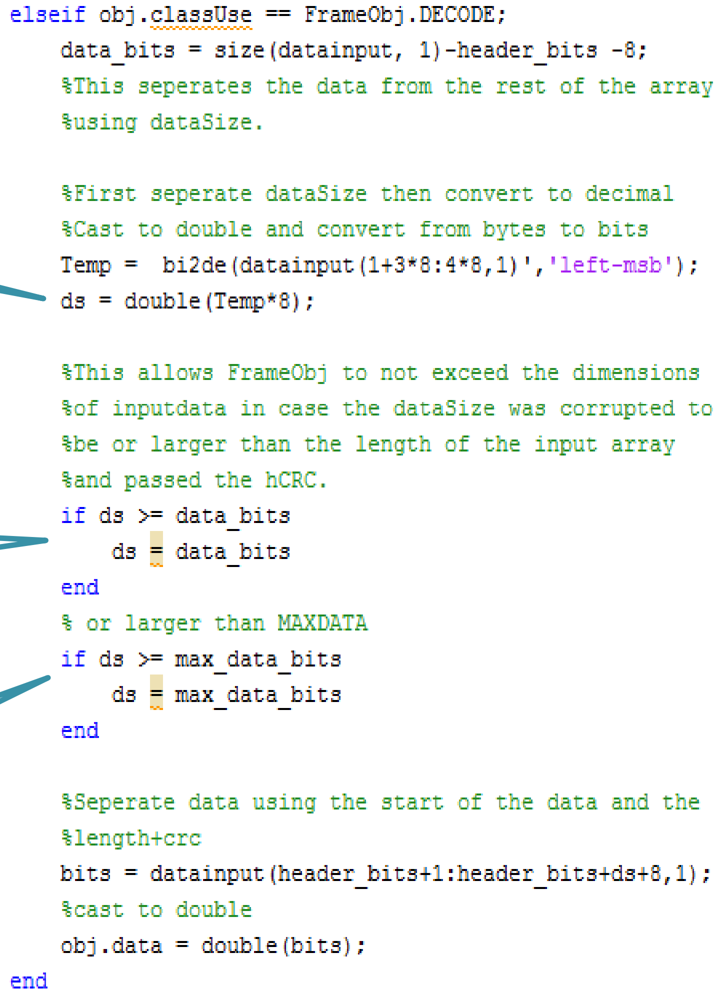
Creation of CRC

Implementation Data field

Extract the data size

The dimensions of the input limit the data size

Maximum size of data field



```
elseif obj.classUse == FrameObj.DECODE;
    data_bits = size(datainput, 1)-header_bits -8;
    %This seperates the data from the rest of the array
    %using dataSize.

    %First seperate dataSize then convert to decimal
    %Cast to double and convert from bytes to bits
    Temp =  bi2de(datainput(1+3*8:4*8,1)','left-msb');
    ds = double(Temp*8);

    %This allows FrameObj to not exceed the dimensions
    %of inputdata in case the dataSize was corrupted to
    %be or larger than the length of the input array
    %and passed the hCRC.
    if ds >= data_bits
        ds = data_bits
    end
    % or larger than MAXDATA
    if ds >= max_data_bits
        ds = max_data_bits
    end

    %Seperate data using the start of the data and the
    %length+crc
    bits = datainput(header_bits+1:header_bits+ds+8,1);
    %cast to double
    obj.data = double(bits);
end
```

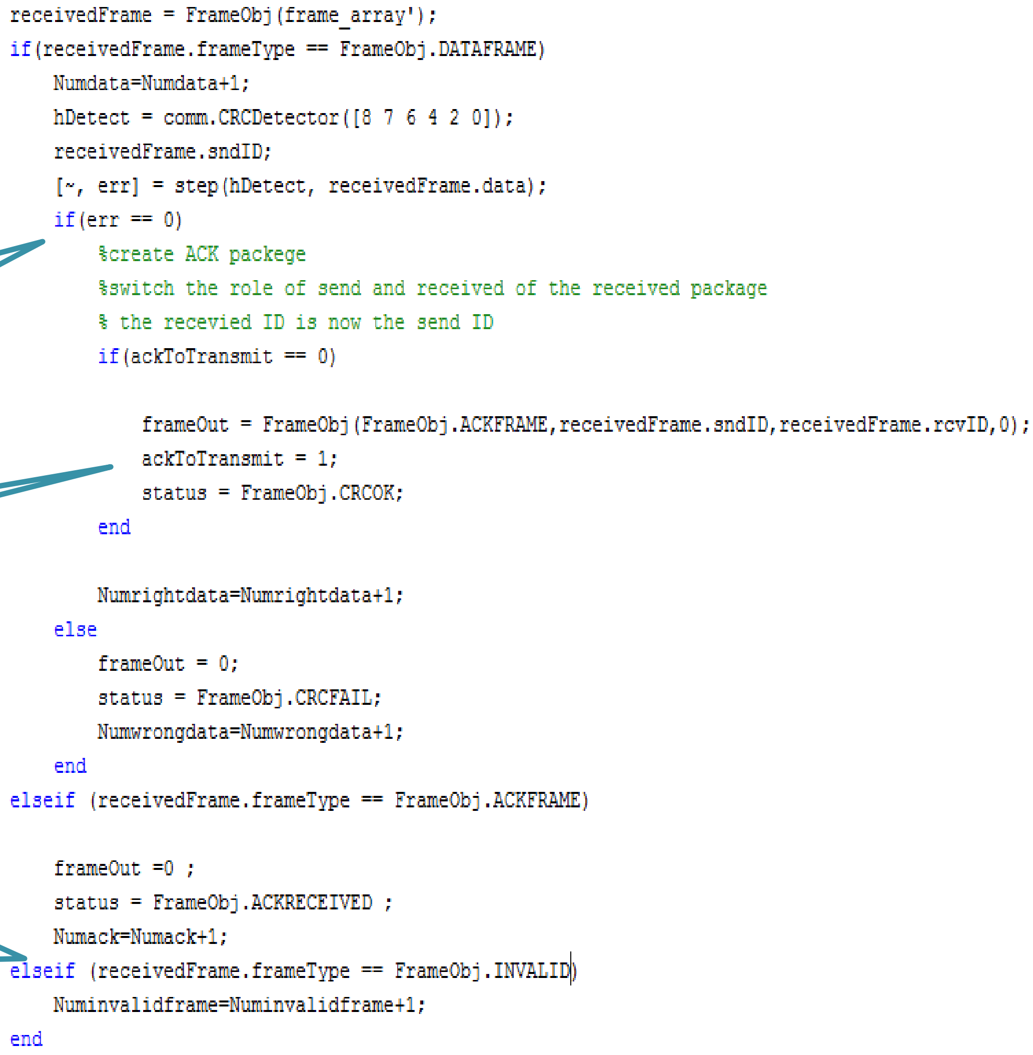
Implementation
Receiving a Frame

Check the FrameType

Verify the data CRC

Create only one ACK to send back

Drop INVALID Frame

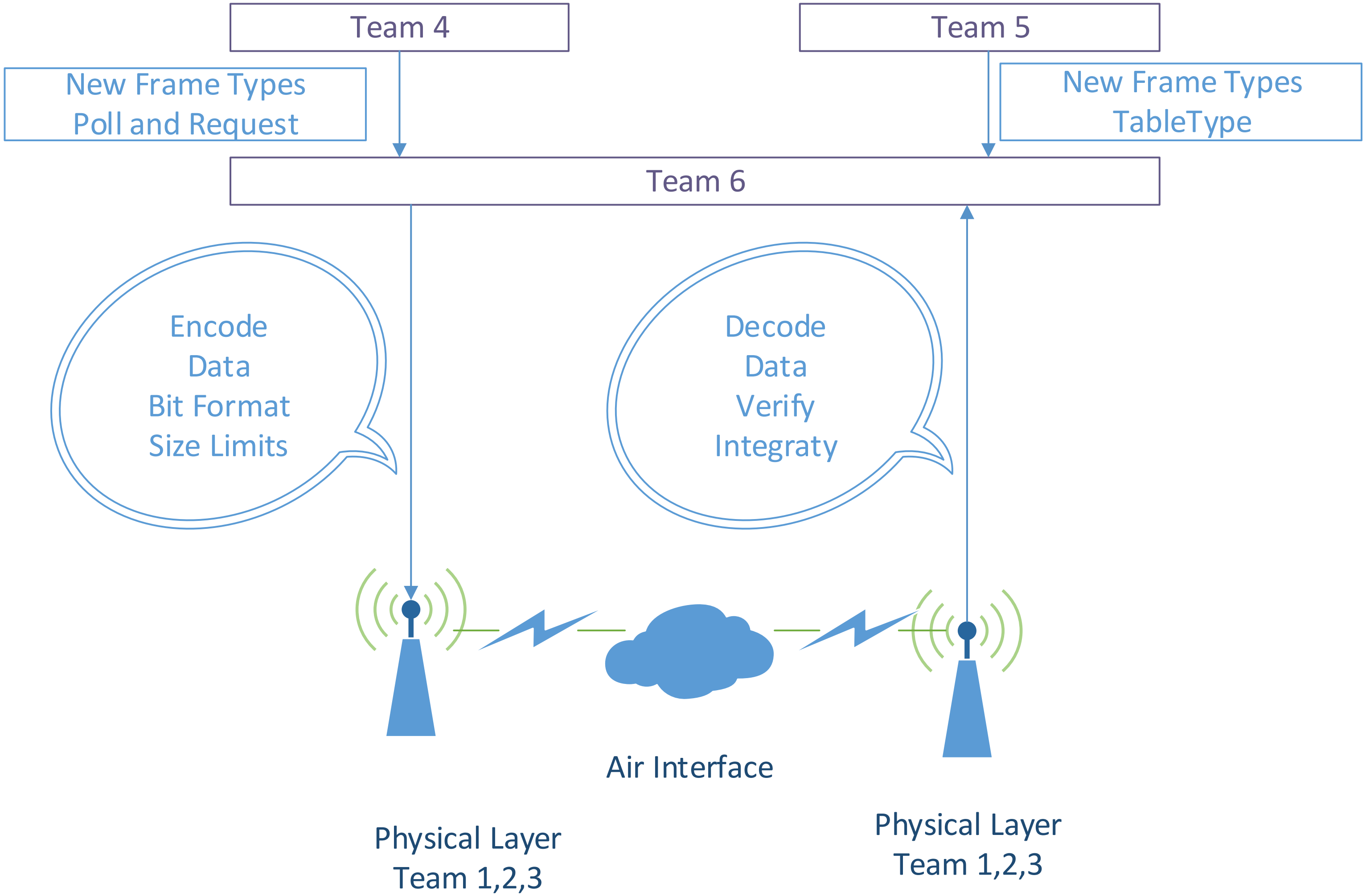


```
receivedFrame = FrameObj(frame_array');
if(receivedFrame.frameType == FrameObj.DATAFRAME)
    Numdata=Numdata+1;
    hDetect = comm.CRCDetector([8 7 6 4 2 0]);
    receivedFrame.sndID;
    [~, err] = step(hDetect, receivedFrame.data);
    if(err == 0)
        %create ACK packege
        %switch the role of send and received of the received package
        % the recevied ID is now the send ID
        if(ackToTransmit == 0)

            frameOut = FrameObj(FrameObj.ACKFRAME,receivedFrame.sndID,receivedFrame.rcvID,0);
            ackToTransmit = 1;
            status = FrameObj.CRCOK;
        end

        Numrightdata=Numrightdata+1;
    else
        frameOut = 0;
        status = FrameObj.CRCFAIL;
        Numwrongdata=Numwrongdata+1;
    end
elseif (receivedFrame.frameType == FrameObj.ACKFRAME)

    frameOut =0 ;
    status = FrameObj.ACKRECEIVED ;
    Numack=Numack+1;
elseif (receivedFrame.frameType == FrameObj.INVALID)
    Numinvalidframe=Numinvalidframe+1;
end
```

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Interfacing Diagram



Team 4

Team 5

New Frame Types
Poll and Request

New Frame Types
TableType

Team 6

Encode
Data
Bit Format
Size Limits

Decode
Data
Verify
Integraty

Air Interface

Physical Layer
Team 1,2,3

Physical Layer
Team 1,2,3

Interfacing with the Physical Modules

- Teams 1, 2 & 3
  - Format of the bits that are generated by FrameObj
    - Length of the frame
    - Order of the bits
  - How to exchange data between the layers
    - Flags to indicate new messages
    - Callbacks
- Team 3
  - Additional processing to route the message

Interfacing with the Network Modules

- Team 4
  - The ACK frame is reused
  - The same basic frame construction is used
    - Polling Frame
    - Request Frame
- Team 5
  - The address table determines which physical resource will route the frames.
  - The same basic frame construction is used
    - Table Frame

Frame Type Functionality

Data Transmission with Transceiver from Lab 2

Frame Type	Number Received
ACKFRAME	2
DATAFRAME • Correct CRC • Corrupt CRC	102 • 100 • 2
INVALID	3466

False Detection of ACK frame because similar frameType numeric representation.

Frame Type	Number Received
ACKFRAME	0
DATAFRAME • Correct CRC • Corrupt CRC	165 • 161 • 4
INVALID	3559

We increased the hamming distance between the values of the types of frames to avoid false detection

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

‘End-to-End’ Simulation of Data Transmission and ACK Response Using Team 4’s Transceiver

Outline

- Purpose
- Data Transmission in the MAC Layer
  - Building the MAC Frame
  - State Machine for the Frame Transmission
  - ACK Timeout
  - Other Features Considered
- Implementation
- Interfacing with the other Modules
- ‘End-to-End’ Simulation of Data Transmission and ACK Response
- Lessons Learned

Lessons Learned

- The Physical layer is very important, we cannot compensate for unreliable data transmission.
- We can accommodate erroneous data through verifying the data range and considering possible received values
- Hamming distance of the frame type are very important
- More data types are needed for the control messages in the system than for transmitting data.