Student - O. Guzelyte (160421859) Report

CSC2026 - Computer Networks – Protocol PAR

Part 1. Protocol PAR in the absence of errors.

Step 1 and 2. I unzipped and put the project into /TinyOS/apps directory under the name of BlinkToRadioCURRENT. Then, I edited the BlinkToRadio.h file and ensured the AM_BLINKTORADIO channel is set to 6 so Echo mote does not drop messages.

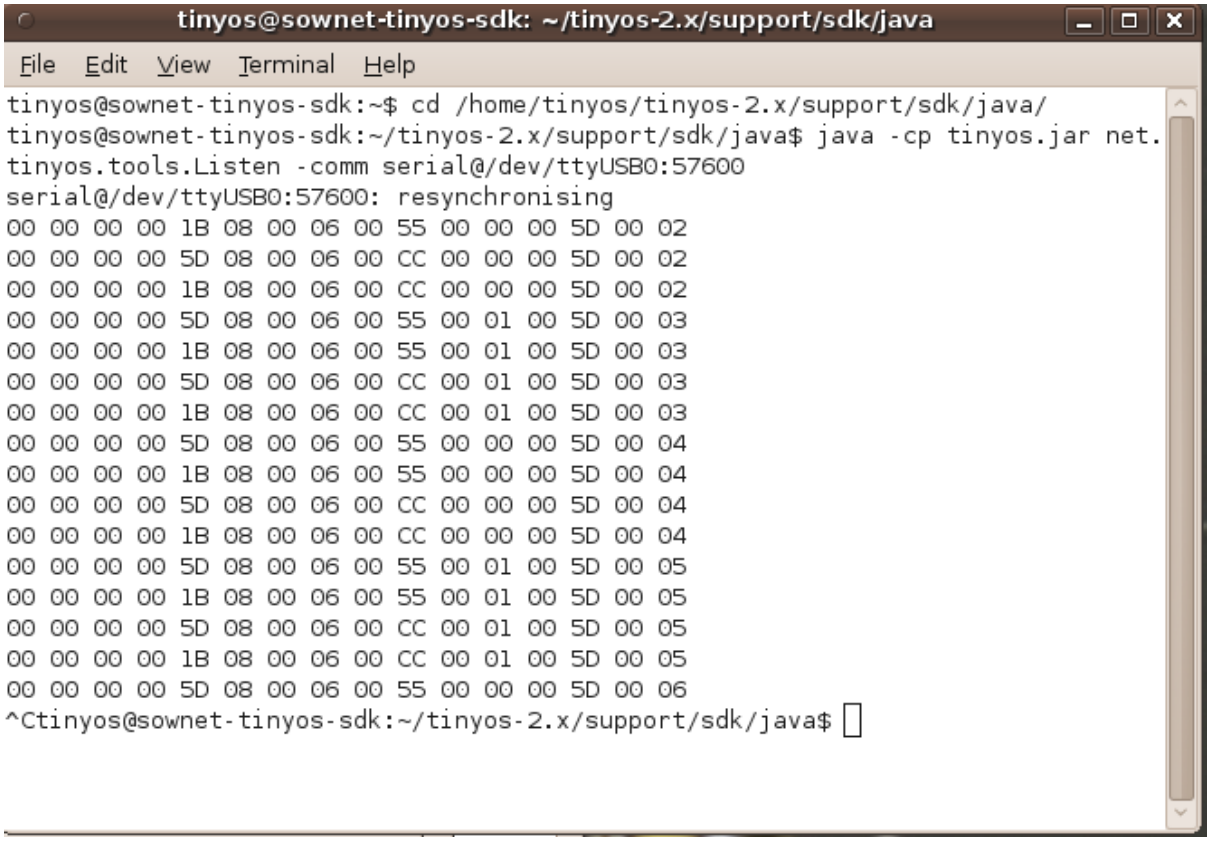
Step 3. In the AMSendReceiveI.receive event I created an additional local pointer to the BlinkToRadioMsg struct so I could construct an acknowledgement message. In the implementation block, I also created an ackMsg pointer to the address of ackMsgBuffer in order to use these to store the information about the acknowledgement message. I was advised to do so as sending the acknowledgement message on the same pointer as sendMsg might create erroneous Terminal input and the data and ack messages might mix up. Subsequently, I added an "if" statement checking whether the received message type is TYPE_DATA in order to be able to send the acknowledgement message. Within the if statement body I called the AMPacket interface to set the active message's fields, such as: type, source, destination, size and assigned it to the local pointer ack_btrpkt, that points to the BlinkToRadioMsg struct. I then defined its fields with the appropriate values. The trickiest part was to change ack_btrpkt counter to that of btrpkt, in order for the acknowledge message to have the same counter as the data message. Then I called the AMSendReceiveI.send(ackMsg) event and sent the message over to Echo mote.

Step 4 and 5. I introduced a Boolean variable called "busy" and put it inside the implementation block to make it visible to all the events. I encapsulated all the code in Timer0.fired() within an if statement. I made sure it would only execute if the busy flag was set to FALSE. I also made sure that every time a data message would be sent the busy flag would be set to TRUE, so the sending would block and no duplicate messages would be sent. In the AMSendReceiveI.receive event I added an additional "else if" statement checking whether the message received is of TYPE_ACK (btrpkt->type==TYPE_ACK). Inside of the "else if" body I simply set the busy flag to FALSE, so the periodic timer that would set off the Timer0.fired() event would be unblocked and data messages could be sent again.

Step 6, 7 and 8. Data and acknowledge messages were sent as expected according to java Listen program. When I commented out the sending of acknowledge messages I witnessed that the busy flag gets set appropriately and Timer0.fired() gets blocked waiting for an acknowledge message. I uncommented and restored sending of acknowledge messages.

Step 9, 10, 11 and 12. I modified my code to increment the counter only in the Timer0.fired() event so I could use that value to correctly set the sequence field. I set the sequence number by taking the remainder of the counter's division by two using the mod operator (btrpkt->seq=counter%2). In result, since each time a data message is sent the counter is incremented only once, the sequence number can only be 0/1.

In the AMSendReceiveI.receive() event I set the acknowledgement message's sequence field to the data message's sequence field so no inconsistencies between sequence numbers would happen (ack_btrpkt->seq=btrpkt->seq). I then added an additional check in the "else if" statement within the AMSendReceiveI.receive() event so that it does not only check whether the message type is TYPE_ACK but also its sequence number is the same as the current data message sent (btrpkt->seq=counter%2).

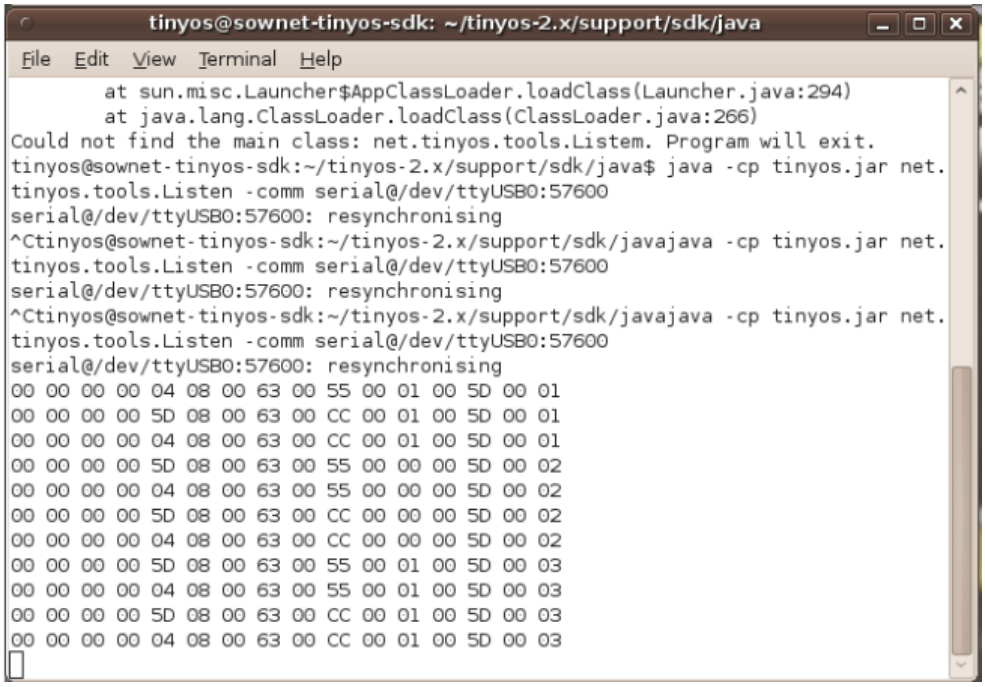


Screenshot 1. Protocol PAR in the absence of errors.

In the screenshot above it is visible that the messages from my mote 93 (hex: 5D) are sent to another mote (I was using a friend's mote, therefore his mote's hex value is 1B). A data message (00 55) is being sent and echoed and an acknowledgement message (00 CC) follows after that and gets echoed back.

Part 1 Protocol PAR in the presence of errors.

Steps 1, 2, 3, 4, 5. I made sure AM_BLINKTORADIO is set to channel 99 to force it drop messages. I then proceeded to test my application using java Listen and it successfully deadlocked after the first dropped message. I am attaching a screenshot as proof below.



Screenshot 2. Protocol PAR in the presence of errors, deadlock

As seen in the screenshot after the sequence number 3 is reached the 4th one gets blocked and nothing else gets executed.

Step 6. In order to correctly save a copy of each data message I created another pointer sendMsg_copy that would point to a buffer called sendMsgBufCopy. Since I was copying structs, I could not use a simple assignment from one struct to another (as a struct can contain a pointer within), so I used memcpy(). I defined it to point the sendMsg pointer to the newly created pointer sendMsg_copy and allocate the size of message_t (memcpy(sendMsg_copy, sendMsg, sizeof(message_t))).

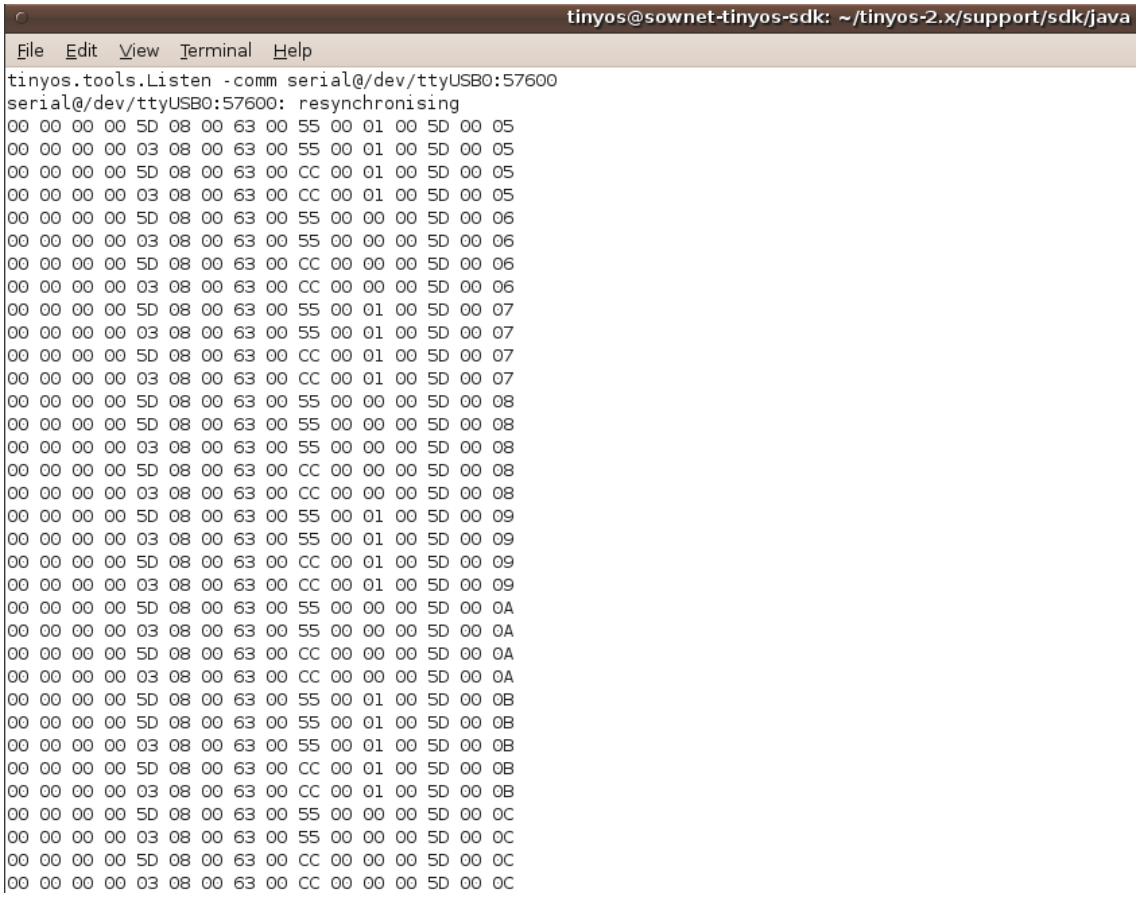
Note: changed this later to deepCopySendMsg(), therefore no memcpy left in code.

Step 7. Inside the implementation block I introduced a new variable called t and set it to 900ms so it would not collide with the Timer0.fired() and a new interface Timer<TMilli> as Timer_ack so I could use it in single-shot mode and resend the data message if an acknowledgement message is not received in 900ms. After sending the data message inside the Timer0.fired() event I call Timer_ack.startOneShot(t), which is meant to set off the Timer_ack timer and let it run for 900ms. I then implemented another event called Timer_ack.fired(), which would resend the copied data message and restart the timer (if the acknowledge message does not get sent again). To stop the timer and prevent it from running indefinitely, inside the AMSendReceiveI.receive() "else if" statement I call another event Timer_ack.stop().

Note: value of t changed to 2000 in the end.

In order to wire the new Timer<TMilli> to the program, I edited the BlinkToRadioAppC.nc and included it as a new component (components new TimerMilliC() as Timer_ack). In it I wired the Timer<TMilli> to the component that provides it (BlinkToRadioC.Timer_ack -> Timer_ack).

Step 8 and 9. Larger values of t make the application less robust but slower, when the value t is smaller, the data messages get resent more so it is easier to get an acknowledgement back.

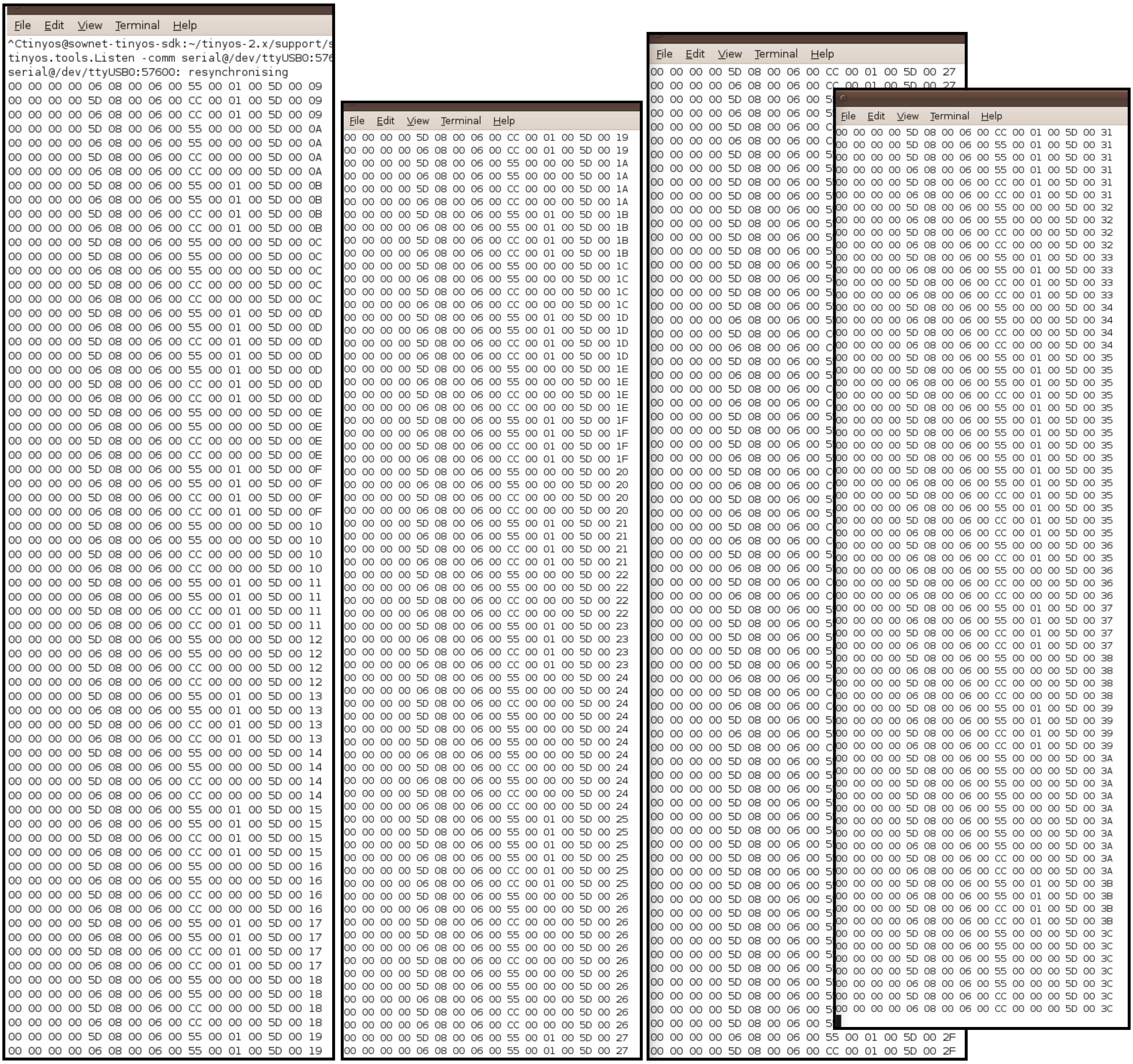


Screenshot 3. Protocol PAR in the presence of errors, result

Part 2. Protocol PAR at speed in the absence of errors.

Step 1, 2, 3 and 4. I set the AM_BLINKTORADIO channel to channel 6 so that Echo does not drop messages.

In order to send the first 60 messages as fast as I can I create a task called send60msgs(), in which I put Timer0.fired() code in order to just send one data message to the Echo mote. I introduce a global variable counter "i" in order to count whether 60 messages were sent. Within the send60msgs() task I increment the aforementioned counter "i" to track how many messages were sent. Subsequently, inside the AMSendReceiveI.receive() method I create an "if" statement, which controls the sending of the messages. Whenever I receive an acknowledgement from Echo (meaning that it correctly echoed my data message), I post my task within the said "if" statement again in order for it to re-send the data message to Echo mote 60 times as if it was in a loop. Otherwise, in the "else if" statement I check whether the 60th message was reached and if so, I start the Timer0.startPeriodic(TIMER_PERIOD_MILLI) to continue sending the messages as normal. I also



increment the counter the last time in order not to periodically set the Timer0.startPeriodic(TIMER_PERIOD_MILLI) unnecessarily.

Step 5. Due to some errors with my counter randomly resetting to 0 sometimes, I decided to create a deep copy of the sendMsg, because memcpy might not have been copying it correctly. I created a method called void deepCopySendMsg() and within it I created the same message to be sent as the sendMsg except I used the sendMsg_copy to be sent around.

Furthermore, since I had my oneShot acknowledgement timer (Timer_ack.startOneShot(t)) firing very fast, it might have interfered with my results as I kept getting too many acknowledgements from an uncongested Echo mote, thus I set my t value to 2000 and it cleared up my results. The t value was not clashing with the Timer0 firing anymore.

In order to test whether my task was working correctly, I commented out the Timer0.startPeriodic(TIMER_PERIOD_MILLI), just to see whether the messages are sent to 3C (60 in hexadecimal). The 4 screenshots above show how my counter goes from beginning to 3C and messages are successfully sent and received.

Part 2. Protocol PAR at speed in the presence of errors.

Step 1 and 2. I set the AM_BLINKTORADIO channel to 99 and inspected the output. The messages get properly sent and received from the Echo mote, so the code keeps up with the dropped packages.

End note: code in BlinkToRadioC.nc and BlinkToRadioAppC.nc is commented according to the parts and steps mentioned in this document. That is the reason why there are no code snippets included in this report.