Internet of Things Laboratory

November 23, 2015

A. Capossele, G.Koutsandria, D. Spenza

Contacts

- Capossele: capossele@di.uniroma1.it
- Koutsandria: koutsandria@di.uniroma1.it
- Spenza: spenza@di.uniroma1.it
  - Tel: 06-49918430
  - Room: 333
  - Slides: www.dsi.uniroma1.it/~spenza/

- SENSES lab
  - http://senseslab.di.uniroma1.it

Lessons Schedule

09.11.2015
8:30-12:00
Lesson 1

10.11.2015
17:15-19:30
Tutoring session

13.11.2015
14:00-17:30
Lesson 2

23.11.2015
08:30-12:00
Lesson 3

24.11.2015
17:15-19:30
Tutoring

27.11.2015
14:00-17:30
Final project

Outline

- Yet another BlinkToRadio exercise..
- Mote-PC serial communication
  - TestSerial Application
  - SerialForwarder
  - BaseStation
- RSSI Demo
- Duty Cycling
  - Low Power Listening

Excercise: Modify BlinkToRadio

1. Modify the content of the BlinkToRadio structure
2. Add the string “HELLO” to the structure
3. Print the received message

Hints

- printf(“%s”, (char*) string);
- memcpy(dst, src, sizeof(bytes));

Mote-PC Communication

Mote-PC Serial Communication

- Collect data from the network
- Send commands to motes
- Monitor the network traffic
- Java/python based infrastructure for communicating with motes

- **Reference:**
  http://tinyos.stanford.edu/tinyos-wiki/index.php/Mote-PC_serial_communication_and_SerialForwarder_(TOS_2.1_and_later)

Mote-PC Serial Communication

- TinyOS provides high-level communication interfaces
  - Similar for radio and serial communication

- **Basic interfaces:**
  - **Packet**: Set/get payload of TinyOS message_t packets
  - **Send**: Send packet by calling send() command
  - **Receive**: Reception of packets signaled by receive() event
- **Active Message interfaces allow for multiplexing:**
  - **AMPacket**: Provide source and destination address to packet
  - **AMSend**: Send packet to destination address

TestSerial Application

- Located in apps/tests/TestSerial
- Sends a packet per second to the serial port
- Displays the packet's sequence number on the LEDs upon reception of a packet

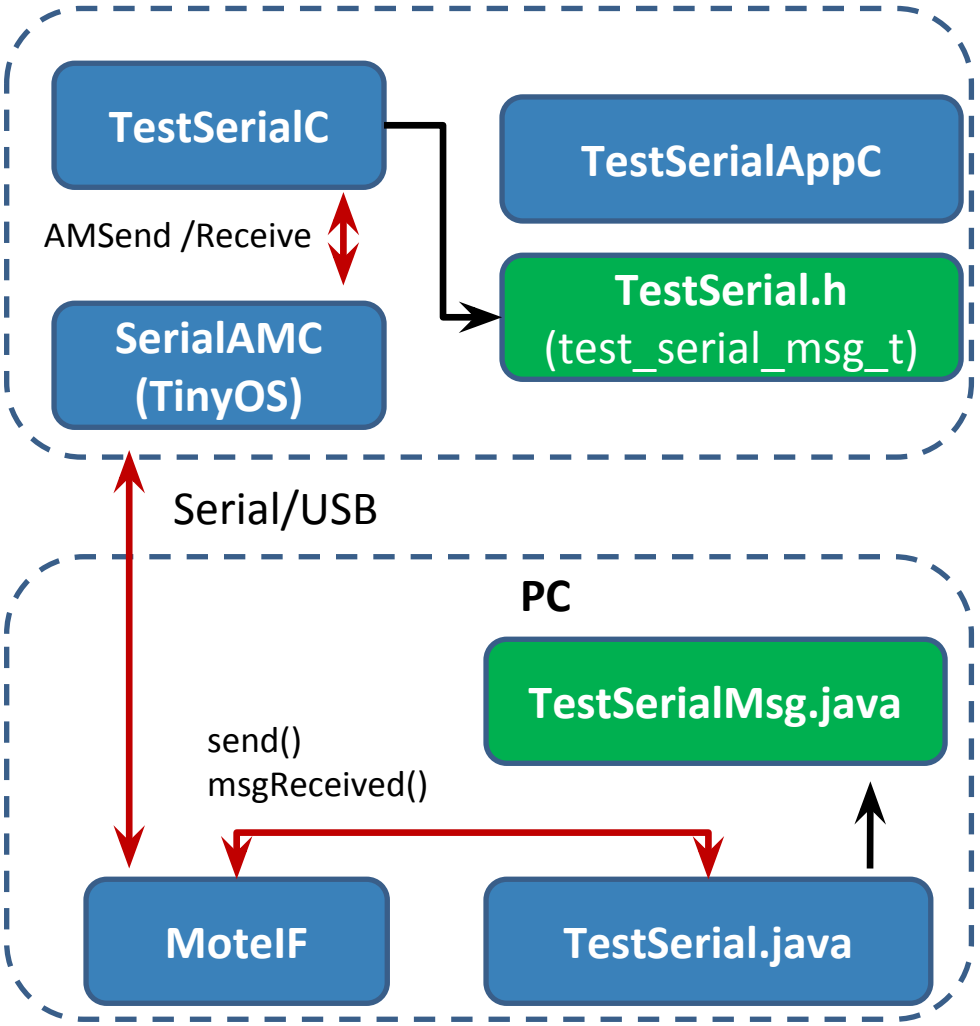
- Test your serial port
  a. Install the app on a mote
  b. Run the java application
     - java TestSerial

TestSerial Application

- Mote and PC components
  - Both increment counter values and send to the other
- Mote: nesC and TinyOS
  - Outputs last three bits of PC counter value to LEDs
- PC: Java and TinyOS Java libraries
  - Outputs mote counter value to stdout
- Demonstration

TestSerialC

- Interfaces AMSend, Receive
- test_serial_msg_t
  - Payload struct
- MoteIF
  - TinyOS Java library to send and receive packets
- TestSerial.java
  - Prints received counter values
  - Increments and sends counter values
- TestSerialMsg.java
  - Payload encapsulation
  - Generated from TestSerial.h



Packet Payload Format

- Usually defined in C header file (TestSerial.h)
- nx-types abstract away big/little endian
- Default payload size is 29 bytes, but can be enlarged
  - PFLAGS +=-DTOSH_DATA_LENGTH=X
- Active Message type AM_TEST_SERIAL_MSG
  - Integer value to distinguish between multiple packet types (multiplexing)
- TinyOS libraries convert struct to a Java class with set/get methods (TestSerialMsg.java)
  - Message Interface Generator (example in TestSerial Makefile)

TestSerialAppC Wiring

```
configuration TestSerialAppC {}
implementation {
 components TestSerialC as App, LedsC, MainC;
 components SerialActiveMessageC as AM;
 components new TimerMilliC();


 App.Boot -> MainC.Boot;
 App.Control -> AM;
 App.Receive -> AM.Receive[AM_TEST_SERIAL_MSG];
 App.AMSend -> AM.AMSend[AM_TEST_SERIAL_MSG];
 App.Leds -> LedsC;
 App.MilliTimer -> TimerMilliC;
 App.Packet -> AM;
}
```

TestSerialAppC: Wiring

- SerialActiveMessageC allows for multiplexing
  - Multiple packet types (e.g. sensor control/data)
  - Differentiate through AM types: AM_TEST_SERIAL_MSG
  - Parameters defined in brackets []
- SerialActiveMessageC provides several interfaces
  - Wired to TestSerialC
  - SplitControl to turn on/off the UART/serial bus
  - AMSend and Receive for transmitting/receiving
  - Packet to set and get payload

TestSerialC: Booting

- When mote boots, turn on UART
- When UART is powered, start timer to send packets
- Implement Control.stopDone() to turn off UART

```
event void Boot.booted() {
  call Control.start();
}
event void Control.startDone(error_t err) {
  if (err == SUCCESS) {
    call MilliTimer.startPeriodic(1000);
  }
}
event void Control.stopDone(error_t err) {}
```

TestSerialC: Sending Packets

- Timer fires, increment counter
- Get message_t payload pointer: Packet. getPayload();
- Set payload value: rcm->counter = counter;
- Send packet: AMSend.send();
  - Provide AM destination address, message_t packet address, payload size
- Packet sent: AMSend.sendDone();

TestSerialC: Sending Packets

```
event void MilliTimer.fired() {
  counter++;
  message_t packet;
  test_serial_msg_t* rcm = (test_serial_msg_t*)call …Packet.
getPayload(&packet, sizeof(test_serial_msg_t));

  rcm->counter = counter;
  call  AMSend.send(AM_BROADCAST_ADDR, &packet, …sizeof
(test_serial_msg_t));
  }

event void AMSend.sendDone(message_t* bufPtr, error_t error){
}
```

TestSerialC: Receiving Packets

- Packet received: Receive.receive();
  - Provides message_t packet, payload pointer, and payload size
  - Get payload: cast from void* to test_serial_msg_t*
  - Set LEDs according to value of last 3 bits

```
event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t
…len){
  test_serial_msg_t* rcm = (test_serial_msg_t*)payload;
  if (rcm->counter & 0x1) {
    call Leds.led0On();
  }
  // turn on other LEDs accordingly
  ...
  return bufPtr;
}
```

PC: TestSerial

- Initialization
  - Creates packet source from args[]: "-comm serial@\dev\ttyUSB0:telosb "
  - Registers packet listener for TestSerialMsg and source
- Send packets

```
public class TestSerial implements MessageListener {
  private MoteIF moteIF;

  public static void main(String[] args) throws Exception {
    ...
    String source = args[1]; PhoenixSource phoenix = …BuildSource.
makePhoenix(source, PrintStreamMessenger.err);
    MoteIF mif = new MoteIF(phoenix);
    TestSerial serial = new TestSerial(mif);
    serial.sendPackets();
  }

  public TestSerial(MoteIF moteIF) {
    this.moteIF = moteIF;
    this.moteIF.registerListener(new TestSerialMsg(), this); }
```

TestSerial.java: sendPackets

- Initialize counter and create TestSerialMsg payload
- While loop
  - Increment counter and sleep for some period
  - Set payload counter: payload.set_counter();
  - Send packet: moteIF.send(); with destination address 0

```
public void sendPackets() {
  int counter = 0;
  TestSerialMsg payload = new TestSerialMsg();
  ...
  while (true) {
    ...
    // increment counter and wait for some amount of time before sending
    System.out.println("Sending packet " + counter);
    payload.set_counter(counter);
    moteIF.send(0, payload);
  }
}
```

TestSerial.java: Receiving Packets

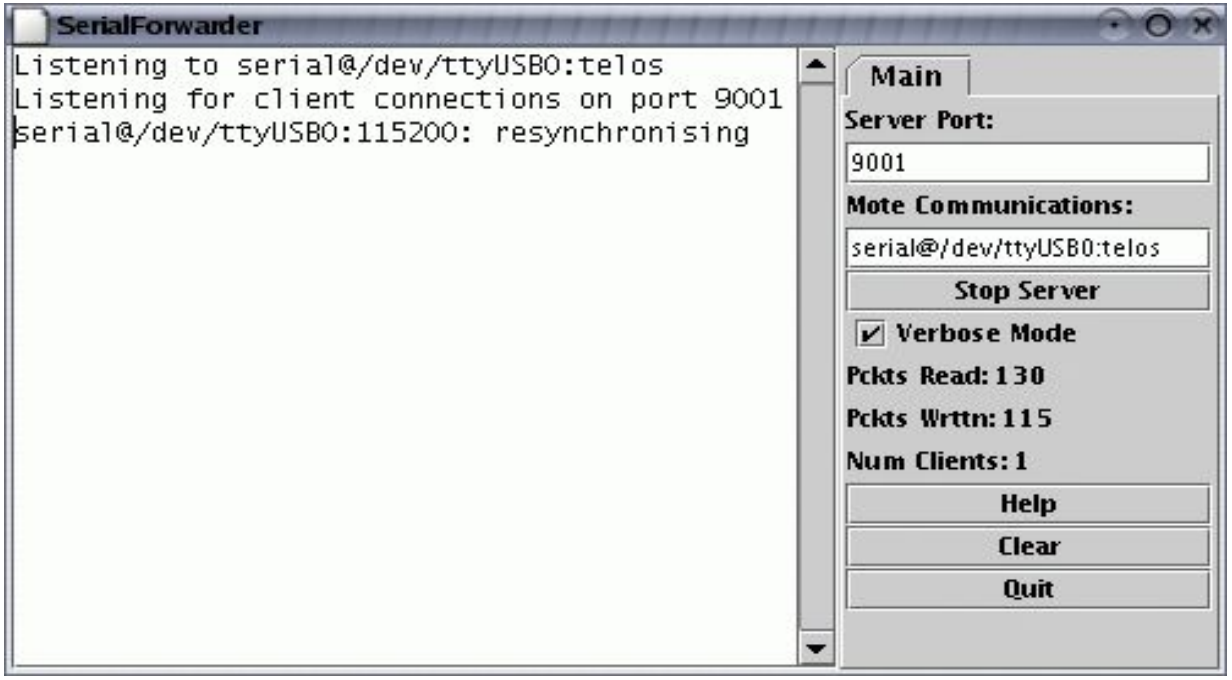
- TestSerial.messageReceived() triggered by incoming packet while listener is registered
  - Provides AM destination address and abstract class Message
- Cast message to TestSerialMsg
- Retrieve counter: msg.get_counter();

```
public void messageReceived(int to, Message message) {
  TestSerialMsg msg = (TestSerialMsg)message;
  System.out.println("Received packet sequence number " +
msg.get_counter());
}
```

Serial Forwarder

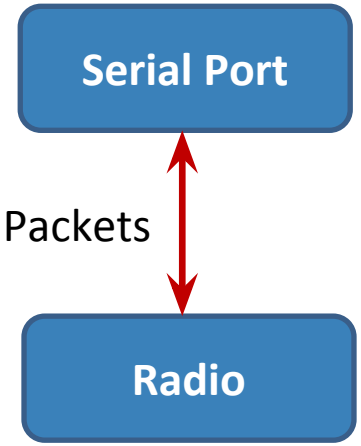
- Acts as a proxy to read and write packets
- Connection over TCP/IP => connection over the Internet
- **No "one-to-one" limitation problems!**

```
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:telosb
```



Base Station

It is a basic TinyOS utility application. It acts as a bridge between the serial port and radio network.



BlinkToRadio

- **Destination address** (2 bytes)
- **Link source address** (2 bytes)
- **Message length** (1 byte)
- **Group ID** (1 byte)
- **Active Message handler type** (1 byte)
- **Payload** (up to 28 bytes):
  - **source mote ID** (2 bytes)
  - **sample counter** (2 bytes)

```
typedef nx_struct BlinkToRadioMsg {
        nx_uint16_t nodeid;
        nx_uint16_t counter;
} BlinkToRadioMsg;
```

$ java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB0:telosb

dest addr	link source addr	msg len	groupID	handlerID	source addr	counter
ff ff	00 00	04	22	06	00 02	00 0B

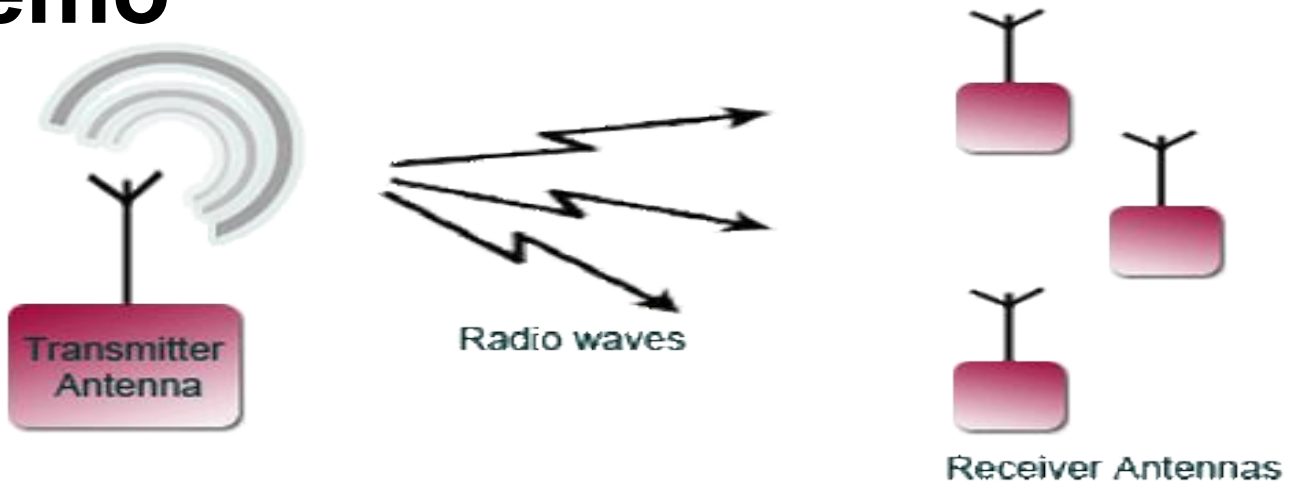
Exercise 1: Sending an integer

1. Create a java/python script that does the following:
   a. Gets an integer as input from the keyboard
   b. Sends the integer to the mote through the serial
2. The mote should receive the typed integer and display its binary value via the LEDs.

Exercise 2: Resend it over the radio

1. Modify exercise 1 in order to do the following:
   a. Once you receive an integer from the PC, forward the packet to other motes using the radio.

RSSI demo



- Received signal strength indicator (RSSI) is a measurement of the power present in a received radio signal
- Indicates the strength with which the receiving device is hearing the sending device. **Higher value = stronger signal**.

Install & run

1. Located in apps/tests/cc2420/RssiToSerial
2. make telosb install
3. java SpecAnalyzer -comm serial@/dev/ttyUSB0:telosb

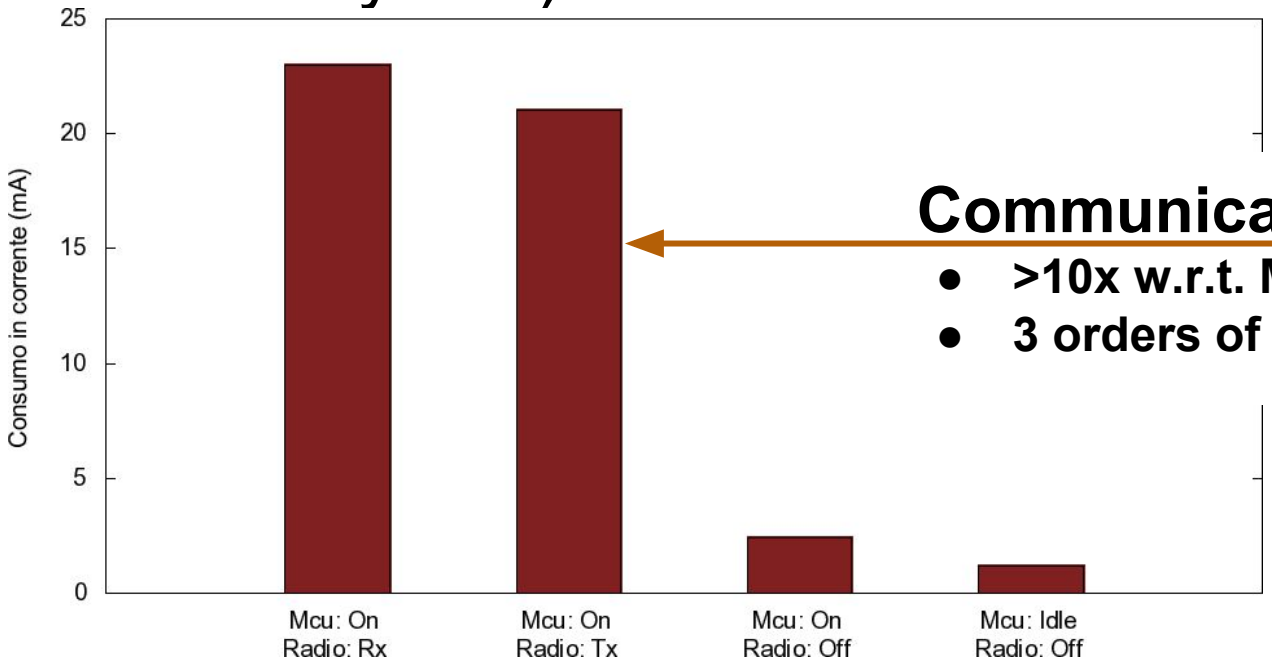
```
$ java SpecAnalyzer -comm serial@COM33:telosb
serial@COM33:115200: resynchronising
Connecting to serial forwarder...
[++++++++++++++++                                    ]
```

```
$ java SpecAnalyzer -comm serial@COM33:telosb
serial@COM33:115200: resynchronising
Connecting to serial forwarder...
[++++++++++++++++++++++++++++++++++++                 ]
```

Duty cycling

Energy Consumption

- In many applications (e.g., SHM) the network is required to run for **decades**
- Nodes are powered by batteries
  - Limited lifetime (a few days on 2xAA batteries if always on)

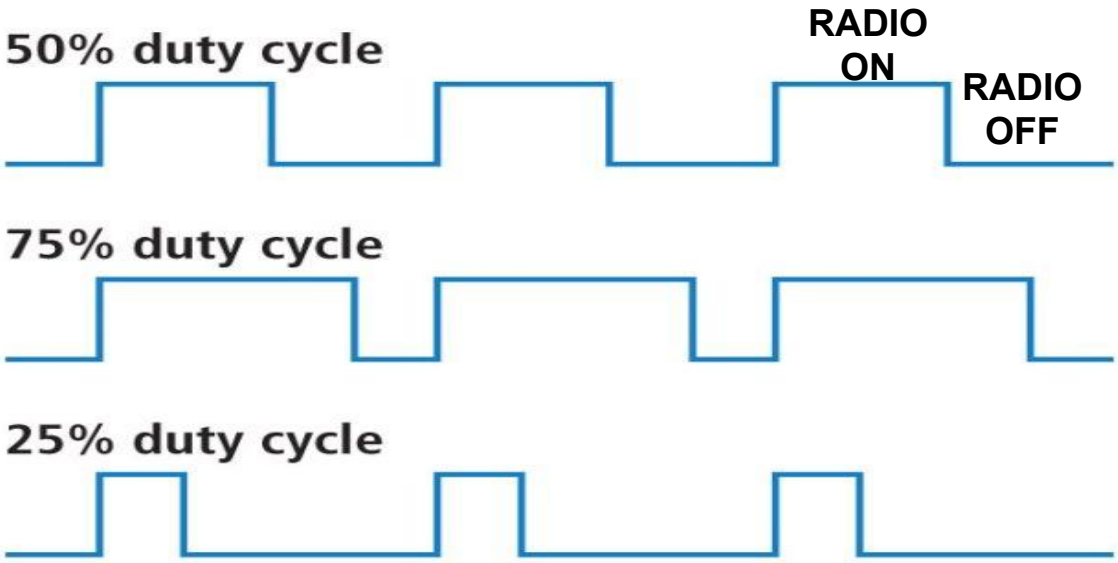


Communication is expensive!

- **>10x w.r.t. MCU on**
- **3 orders of magnitude w.r.t. sleep**

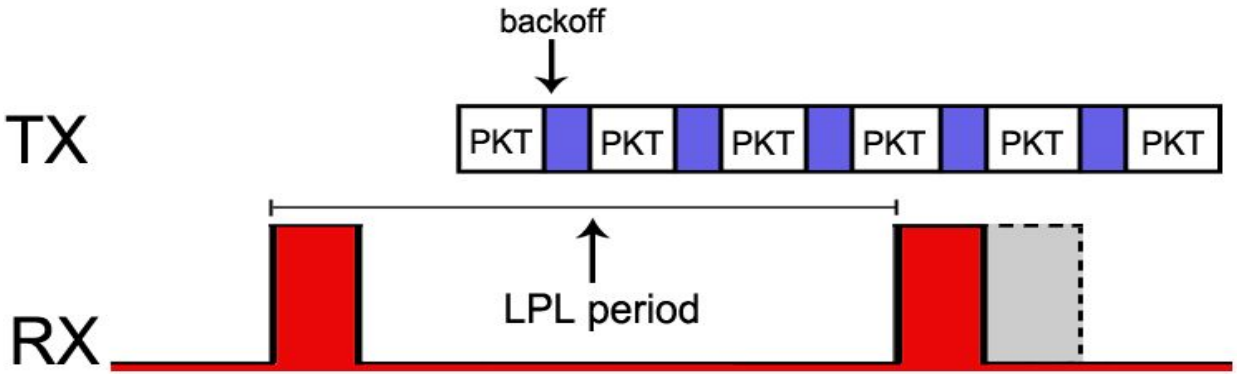
Standard Approach: Duty Cycling

- Periodically cycle the radio between ON/OFF states
  - OFF = save energy, but **no communication**
  - ON = **high energy**, but data can be transmitted and received



Low Power Listening

- Goal: periodically turn on the radio to check for traffic
- LPL period **T** fixed (e.g., 500ms)
- Listen period ~ 5ms
- Transmitter repeatedly sends the packet for **T** ms
- The receiver wakes up at some point and downloads the packet
- Advantages:
  - Very low power
- Drawback:
  - Higher traffic
  - Higher latencies
  - Higher collisions



Latency vs. Energy Trade-off