OFDM, Mobile Software Development Framework

9/27/2012

Y. Richard Yang
Admin.

- Homework 2 to be posted by Friday
- Start to think about project
Recap

- Inter-Symbol Interference (ISI)
  - Handle band limit ISI
  - Handle multipath ISI
    - Viterbi
      - problems: Its complexity grows exponentially with D (the number of multipaths taps relative to the symbol time)
      - Q: how to reduce D?
OFDM: Basic Idea

- Uses multiple carriers modulation (MCM)
  - each carrier (called a subcarrier) uses a low symbol rate
    - for N parallel subcarriers, the symbol time can be N times longer
  - spread symbols across multiple subcarriers
    - also gains frequency diversity
Benefit of Symbol Rate on ISI
Multiple Carrier Modulation
Multiple Carrier Modulation (MCM): Problem

- Despite wave shaping, there can be leak from one subcarrier to another subcarrier

- Conventional design: guard bands to avoid interference among subcarriers

- Guard band wastes spectrum
Objective: Avoid subcarrier interference

- Interference of subcarrier $i$ on subcarrier $j$

- Assume no pulse wave shaping, matched filter

\[
\int_0^T \sin(2\pi f_i t + \phi_i) \sin(2\pi f_j t + \phi_j) \\
= \frac{1}{2} \int_0^T \cos[2\pi(f_i - f_j) t + \phi_i - \phi_j] + \cos[2\pi(f_i + f_j) t + \phi_i + \phi_j]
\]

Condition for the interference to be always 0?
**Objective: Avoid subcarrier interference**

If integer number of cycles in \([0, T]\)

\[
\int_{0}^{T} \cos[2\pi ft + \phi] dt = 0
\]

# cycles in \(T\) is \(T \times f\) \(\Rightarrow T \times f = \text{integer}\)

**Symbol period**
OFDM Key Idea: Orthogonal Subcarriers

- Each subcarrier frequency is chosen so that an integral number of cycles in a symbol period, i.e.,
  - subcarrier freq = k \( \frac{1}{T} \)

They do not need to have the same phase, so long integral number of cycles in symbol time T!
OFDM Modulation
Orthogonal Frequency Division Multiplexing

- OFDM allows overlapping subcarriers frequencies

Figure 2: Spectra of OFDM Subchannels

http://www1.linksys.com/products/images/ofdm.gif
802.11a
OFDM Implementation

- Take N symbols and place one symbol on each subcarrier (freq.)

- Q: complexity of the implementation strategy?

\[
\begin{align*}
&\text{Freq}_0 & d_0 e^{j2\pi f_{sc} 0} & d_0 e^{j2\pi f_{sc} T_s} & d_0 e^{j2\pi f_{sc} 2T_s} \\
&\text{Freq}_{N-1} & d_{N-1} e^{j2\pi (N-1)f_{sc} 0} & d_{N-1} e^{j2\pi (N-1)f_{sc} T_s} & d_{N-1} e^{j2\pi (N-1)f_{sc} 2T_s}
\end{align*}
\]
OFDM: Implementation Issue

- Hardware implementation can be expensive if we use one oscillator for each subcarrier
- Software implementation requires $N$ multiplications per time output \(\Rightarrow N^2\) multi. per $N$ outputs

\[
\begin{align*}
\text{Freq}_0 & \quad d_0 e^{j2\pi 0 f_{sc} T_s} \\
\text{Freq}_{N-1} & \quad d_{N-1} e^{j2\pi (N-1) f_{sc} T_s} \\
\end{align*}
\]
OFDM: Key Idea 2

Consider data as coefficients in the frequency domain, use inverse Fourier transform to generate time-domain sequence.

Assume N outputs per symbol time T, \( f_{sc} = 1/T \)

\[
out_k = \sum_{n=0}^{N-1} d_n e^{j2\pi (nf_{sc})kT_s}
\]

\[
X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}
\quad k = 0, \ldots, N - 1
\]

\[
x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn}
\quad n = 0, \ldots, N - 1.
\]
OFDM Implementation: FFT
OFDM Implementation

- Parallel data streams are used as inputs to an IFFT
- IFFT does multiplexing and modulation in one step!
Guard Interval: Removing ISI

- Orthogonal subcarriers remove inter-carrier interference
- Slow symbol rate reduces inter-symbol interference, but may still have ISI

Basic idea of GI: skip the first part “damaged” signal

More details: Chap. 13.1.4 Gast
OFDM Guard Interval

OFDM Implementation

http://proquest.safaribooksonline.com/0596100523?tocview=true
OFDM in 802.11a

- Subcarrier frequency spacing 312.5KHz
  - $1/312.5\text{KHz} = 3.2\text{us}$
  - 64 samples FFT
  - 16 samples
    - Guard Interval

Other Multipath Techniques

- There are other techniques to handle multipath such as Rake Receiver
- See backup slides for some details
**Summary of PHY**

(a) IEEE 802.11b 2Mbps

Transmitter:
- Bits @2Mbps ➔ Scramble ➔ DQPSK Mod ➔ Direct Sequence Spread Spectrum ➔ Symbol Wave Shaping ➔ Samples @1.4Gbps ➔ To RF

Receiver:
- Samples @1.4Gbps ➔ Decimation ➔ Despreading ➔ DQPSK Demod ➔ Descramble ➔ Bits @2Mbps ➔ To MAC

(b) IEEE 802.11a/g 24Mbps

Transmitter:
- Bits @24Mbps ➔ Scramble ➔ Convolutional encoder ➔ Interleaving ➔ QAM Mod ➔ IFFT ➔ GI Addition ➔ Symbol Wave Shaping ➔ Bits @24Mbps ➔ To RF

Receiver:
- Bits @24Mbps ➔ Decimation ➔ Remove GI ➔ FFT ➔ Demod + Interleaving ➔ Viterbi decoding ➔ Descramble ➔ Samples @24Mbps ➔ To MAC
Wireless PHY

Big Picture

Applications

Wireless/Mobile Application Development Framework

Foundational Services: Communications, Location, Service Discovery, UI/Media, Power Management, Security
Overview

- Mobile/Wireless software development framework for mobile wireless applications is a quite large topic
- We have already seen Gnuradio as an example framework
- We will cover more examples
  - TinyOS, J2ME, Android, IOS
- Approach for designing/evaluating each software development framework:
  - Focus on the key concepts introduced by each framework
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
GNURadio: Design Objective

- A software development toolkit that provides signal processing blocks to implement software-defined radio systems.
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
    - Hardware setting
**GNURadio Hardware Arch**

- **Hardware Frontend**
  - RF Frontend (Daughterboard)
  - ADC/DAC and Digital Frontend (USRP)
- **Host Computer**
  - GNU Radio Software

[Link to GNURadio Lecture](http://mobiledevices.kom.aau.dk/fileadmin/mobiledevices/teaching/software_testing/Gnu_radio_lecture.pdf)
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
    - Hardware setting
    - Software concepts
Basic Software Concepts

- **Block**

- **Flow graph**

![Diagram of Basic Software Concepts](image-url)
Basic Software Concepts


- `gr_basic_block` (name, in/out signature, msg queue)
  - `gr_block` (Leaf block; key functions forecast/g general_work)

- `gr_hier_block2` (container block; key functions: connect/disconnect/lock/unlock)
  - `gr_top_block` (flow graph; start/stop/wait)
Software/Execution Model

- **Software model**
  - **Python**
    - Application management (e.g., GUI)
    - Flow graph construction
    - Non-streaming code (e.g., MAC-layer)
  - **C++**
    - Signal processing blocks
    - Certain routines also coded in assembly

- **Execution model**
  - Python thread for each top_block

Discussion: benefits/issues of the hybrid software structure?
Summary: GNURadio

- Interesting/key software design techniques you learned from GNURadio?
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNU Radio
    - Hardware setting
    - Software concepts
  - TinyOS
Design Goal

- A free and open source component based operating system and platform targeting wireless sensor networks (WSNs)

- Example app
  - Environment monitoring, e.g.,
    - measure temperature, lighting values/events
    - periodically transmit measurements/events to a base station
    - forward data for other nodes that are out of range of the base station

http://www.tinyos.net/tinyos-1.x/doc/tutorial/
Hardware

- Assembled from off-the-shelf components
- 4Mhz, 8bit MCU (ATMEL)
  - 512 bytes RAM, 8KB ROM

Devices
- serial Port
- temperature sensor & light sensor
- 900Mhz Radio (RF monolithics)
  - 10-100 ft. range
- LED outputs

1.5” x 1.5”
Schematic Diagram of a Mote

AT 90LS8535

- 8bit data bus
- SRAM
- PC
- Pgm. mem. (flash)
- Inst. Register
- Inst. Decoder
- Ctrl lines
- Regs
- ALU
- SR
- SP
- EEPROM

- Reference Voltage
- 4 MHz clock
- 32.768 MHz clock

- SPI
- Coprocessor AT90L2313
- EEPROM

- UART
- TX
- RX
- Serial Port
- Pwr
- data

- IO pins
- ADC
- Light Sensor
- Temp AD7418

- IO pins
- Ctrl
- TX
- RX

- RFM TR100
- 916 MHz transceiver

- IO pins
- Timer Unit
- LEDs

- AT 90LS8535

Reference Voltage
4 MHz clock
32.768 MHz clock
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
    - Hardware setting
    - Software concepts
  - TinyOS
    - Hardware setting
    - Software concepts
Requirements on Software Dev. Framework

- Flexible configuration of attached devices

- Small footprint
  - Devices have limited memory and power resources
TinyOS: Software Concept

- TinyOS: Generate customized OS + application for each given scenario
  - support one application at a time but flexible reprogramming
Schematic Diagram
TinyOS: Software Concepts

- A TinyOS consists of one or more components linked together
  - software components motivated by hardware component

- Each component specifies that
  - it provides some interfaces
    - allows other components to control it
  - also uses some interfaces
    - control other components
interface

- An interface declares
  - a set of functions called **commands** that the provider must implement
  - another set of functions called **events** that the interface user must implement

---

A uses interfaces I1 and I2

- B provides I1
- C provides I2
- C provides I3
**Interface: Examples**

---

**StdControl.nc**

```plaintext
interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

**Timer.nc**

```plaintext
interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event result_t fired();
}
```

**ADC.nc**

```plaintext
interface ADC {
    async command result_t getdata();
    async command result_t getContinuousData();
    event result_t dataReady(uint16_t data);
}
```
Backup Slides
Rake Receiver
Multipath Diversity: Rake Receiver

Instead of considering delay spread as an issue, use multipath signals to recover the original signal.

Used in IS-95 CDMA, 3G CDMA, and 802.11.

Invented by Price and Green in 1958.

Multipath Diversity: Rake Receiver

- Use several "sub-receivers" each delayed slightly to tune in to the individual multipath components
- Each component is decoded independently, but at a later stage combined
  - this could very well result in higher SNR in a multipath environment than in a "clean" environment
Rake Receiver: Matched Filter

- Impulse response measurement
- Tracks and monitors peaks with a measurement rate depending on speeds of mobile station and on propagation environment
- Allocate fingers: largest peaks to RAKE fingers
Rake Receiver: Combiner

- The weighting coefficients are based on the power or the SNR from each correlator output.

- If the power or SNR is small out of a particular finger, it will be assigned a smaller weight:

\[ \alpha_m = \frac{Z_m^2}{\sum_{i=1}^{M} Z_i^2} \]

Multipath Returns
Comparison [PAH95]

MCM is OFDM