CS434/534: Topics in Networked (Networking) Systems

Improve Wireless Capacity; Programmable Wireless Networks

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
    - wireless access problem and taxonomy
    - wireless resource partitioning dimensions
    - media access protocols
  - From single-hop MAC to multi-hop mesh
    - wireless mesh network capacity
    - maximize mesh capacity
Admin.

- PS2 due: Tuesday at 11:55 pm
- Please stop by Geng’s office at any time between now and Tuesday deadline

- Project first check point:
  - A meeting w/ instructor by Monday, Apr. 3
Recap: Capacity of Mesh Networks

- The question: how much traffic can a mesh wireless network carry, assuming an oracle to avoid the potential overhead of distributed synchronization (MAC)?

- Why study this question: learn the fundamental limits of mesh wireless networks, separate the spatial reuse perspective and system design perspective, gain insight for designing effective wireless protocols.
Wireless Capacity & Capacity Improvements

Radio interface constraint:
- a single half-duplex transceiver at each node

Interference constraint:
- transmission successful if there are no other transmitters within a distance \((1+\Delta)r\) of the receiver

Multiple transceivers

Reduce L

\[ \sum_{b=1}^{\lambda T} h(b) \leq WT \frac{n}{2} \]

Approx. optimal

Increase W

\[ \sum_{b=1}^{\lambda T} \sum_{h=1}^{h(b)} (r_b^h)^2 \leq \frac{16 WT}{\pi \Delta^2} \]

rate*distance capacity:

\[ \lambda \frac{L}{\pi} \leq \sqrt{\frac{8 W}{\pi} \sqrt{n}} \]
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
    - understanding: wireless mesh network capacity
    - design: maximize mesh capacity
      - reduce L
Change Traffic Pattern: Reduce L

- Reduce L => make communications local
  - node placement: change the demand patterns (thus L)
    - e.g. base stations/access points with high-speed backhaul
Change Traffic Pattern: Reduce L

- Reduce L => make communications local by being patient and wait in a mobile network
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
    - understanding: wireless mesh network capacity
    - design: maximize mesh capacity
      - reduce L
      - MIMO: Use multiple antennas
Reduce Interference Footprint

- Antenna design: steered/switched directional antennas

Exercise: redo capacity derivation if limited interference.
Multiple Input Multiple Output (MIMO)

- 4x4 MIMO

- LTE

- Kindle Fire HD
MIMO Basics

Solve two variables from two equations.

\[ y_1 = h_{11} x_1 + h_{21} x_2 \]
\[ y_2 = h_{12} x_1 + h_{22} x_2 \]
**Using MIMO for more Concurrency: Motivation**

Assume tx1 is sending to rx1

Can tx2 transmit in 802.11 using carrier sensing?

No Transmission in current 802.11n
MIMO Benefit: Concurrency using Interference Nulling

**tx2:** for every symbol $q$, transmits $q$ on first antenna and $aq$ on second antenna.

Interference at rx1:

$$(h_{21} + \alpha h_{31})q$$

if tx2 picks $\alpha = -\frac{h_{21}}{h_{31}}$

NO interference at rx1.
Problem

- rx2 hears p from tx1
- Can rx2 decode?
Decoding at rx2:
Observation

- for different symbols $p$ from tx1, the received signal at rx2 moves along vector $\vec{h}_{tx1}$

\[
\vec{y} = \begin{pmatrix} h_{12} \\ h_{13} \end{pmatrix} p = \vec{h}_{tx1} p
\]

- rx2 can estimate channels $h_{12}, h_{13}$ from preamble

Perp. of tx1 space
Decoding at rx2:
Removing tx1 signal by Projection

\[ \tilde{y} = \begin{pmatrix} h_{12} \\ h_{13} \end{pmatrix} p = \bar{h}_{tx1} p \]

- rx2 projects received signal orthogonal to \( \bar{h}_{tx1} \)
Decoding at rx2:

Projection Details

- rx2 picks $w_2$ and $w_3$:

\[ w_2 \cdot h_{12} + w_3 \cdot h_{13} = 0 \]

to compute

\[ w_2 \cdot y_2 + w_3 \cdot y_3 \]
Decoding at rx2: Projection Details

Summary: MIMO allows concurrent transmissions.
Problem of Only Nulling

Assume both tx1 and tx2 are transmitting.

If only nulling, tx3 cannot transmit

\[ r(h_{41} \alpha' + h_{51} \beta' + h_{61} \gamma') = 0 \]
\[ r(h_{42} \alpha' + h_{52} \beta' + h_{62} \gamma') = 0 \]
\[ r(h_{43} \alpha' + h_{53} \beta' + h_{63} \gamma') = 0. \]

Assume both tx1 and tx2 are transmitting.
Solution: MIMO using Interference Alignment

Assume both tx1 and tx2 are transmitting.

Key idea: rx2 ignores interference from tx1 by projection. If tx3 aligns tx3 -> rx2 interference along the same direction as that of tx1 -> rx2, then rx2 can remove it too.
MIMO with Nulling and Alignment

\[ y_2 = h_{12}p + (h_{22} + \alpha h_{32})q + (\alpha' h_{42} + \beta' h_{52} + \gamma' h_{62})r \]
\[ y_3 = h_{13}p + (h_{23} + \alpha h_{33})q + (\alpha' h_{43} + \beta' h_{53} + \gamma' h_{63})r. \]

tx3 picks \( \alpha', \beta', \gamma' \)

\[ \frac{(\alpha' h_{42} + \beta' h_{52} + \gamma' h_{62})}{h_{12}} = \frac{(\alpha' h_{43} + \beta' h_{53} + \gamma' h_{63})}{h_{13}} = L \]

rx2 sees:

\[ y_2 = h_{12}(p + Lr) + (h_{22} + \alpha h_{32})q \]
\[ y_3 = h_{13}(p + Lr) + (h_{23} + \alpha h_{33})q \]

Because rx2 projects to orthogonal to \( \vec{h}_{tx1} \), no interference from tx3 to rx2
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
    - understanding: wireless mesh network capacity
    - design: maximize mesh capacity
      - reduce L
      - MIMO: Use multiple antennas
      - Cognitive radio: use unlicensed spectrum
Spectrum Allocation Chart
Unlicensed Spectrum

- **Opportunity:** unlicensed spectrum is large and has low utilization
  - US unlicensed freq:
    - 2.400-2.4835 G
    - 902-928 M
    - 5.800-5.925G
    - 5.15-5.25 G (200 mw)
    - 5.25-5.35 (1 w)
    - 5.725-5.825 (4w)
Problem of Using Unlicenced

- Unlicensed spectrum may have occupants and is fragmented

- Requirement: Coexistence with dynamic and unknown narrowband devices in the unlicensed spectrum
Cognitive Aggregation

- **Cognition:** Detect unoccupied bands
- **Aggregation:** Weave all unoccupied bands into one link

[Diagram showing wideband spectrum with sections for Zigbee, 802.11a, Others, and Unlicensed Spectrum]
Research Issues

- How to detect available frequency bands?

- How to operate across chunks of non-contiguous frequencies?

- How do sender and receiver establish communication when their perceived available frequency bands differ?
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
- Wireless networking software framework
# Big Picture

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless/Mobile Application Development Framework</td>
</tr>
<tr>
<td>Foundational Services: Communications, Location, Service Discovery, UI/Media, Power Management, Security</td>
</tr>
</tbody>
</table>
Overview

- Wireless/mobile software development framework for mobile wireless applications is a quite large topic
- We will cover both examples and basic principles
  - SDR
    - GNURadio, SORA, OpenRadio
  - Software OS for sensors
    - TinyOS
  - Mobile
    - J2ME
    - Android, IOS
- Approach for designing/evaluating each software development framework:
  - Focus on the key concepts introduced by each framework
Outline

- Recap

- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh

- Wireless networking software framework
  - Software defined radio (SDR)
Software Defined Radio

- Instead of hardware-centric radio, making wireless networking systems software-centric.

- Key challenges
  - Handle large data volume
  - Provide hard deadline and accurate timing control
  - Provide easy to use, compose signal processing blocks
Outline

- Recap

- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh

- Wireless and mobile software framework
  - Software defined radio (SDR)
    - GNURadio
Basic Software Concepts

- **Block**
  - FFT
    - FFT Size: 1.024k
    - Forward/Reverse: Forward
    - Window: window.blackmanhar...
    - Shift: Yes

- **Flow graph**
  - Signal Source
    - Sample Rate: 32k
    - Waveform: Cosine
    - Frequency: 300
    - Amplitude: 5
    - Offset: 0
  - Noise Source
    - Noise Type: Gaussian
    - Amplitude: 1
    - Seed: 42
  - Audio Sink
    - Sample Rate: 32KHz
Basic Software Concepts

- `gr_basic_block` (name, in/out signature, msg queue)
  - See [http://gnuradio.org/doc/doxygen-3.4/classgr__basic__block.html](http://gnuradio.org/doc/doxygen-3.4/classgr__basic__block.html)

- `gr_block` (leaf block; key functions forecast/general_work)

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

What is the design pattern?
Software/Execution Model

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

Execution model
- Python thread for each top_block

Discussion: benefits/issues of the hybrid software structure?
Interesting/key software design techniques you learned from GNURadio?

- Composite pattern to build a hierarchy of blocks
- Define `gr_block` as a reusable base so that defining a new block is typically simple: overwrite `general_work` and `forecast`
- Internal scheduler to orchestrate data flows among blocks
- Hybrid software system (Python/C++)
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
- Wireless and mobile software framework
  - GNURadio
  - SORA
Sora: Goal

Resolving the SDR platform dilemma
- Commodity PC w/ C program
- High performance
  - sysinput: 10Gbps; ~μs latency
  - target wireless xput: 10M~1Gbps

Example: GNU Radio/USRP(v1&2)
- Interface USB/GbE: <1Gbps, >1ms
- Achievable wireless xput: ~100Kbps

Example: Rice WARP, TI SFF-SDR

Programmable hardware (FPGA)
Embedded DSP

Low-performance GPP-based SDR

Performance
Low
High

Programmability
Low
High

https://www.microsoft.com/en-us/research/project/microsoft-research-software-radio-sora/
Sora: Hardware (RF)

Multi-core CPU

APP | APP | APP | APP

Sora | Sora

Mem

Digital Samples @Multiple Gbps

RCB

PCIe bus

Sora Soft-Radio Stack

Sora Hardware

General radio front-end: 700M/1.8G/2.4G/5GHz
Sora: Hardware (RCB)

- Buffered data path: bridging the synchronous ops at RF and asynchronous processing at CPU (12.3Gbps measured)
- Low latency control path for software (0.36 μs measured)
Sora: Software

- Efficient impl of blocks using LUT/SIMD
- Utilizing multi-core for streaming processing
- Dedicated core for real-time support
Sora Software: Acceleration (LUT)

- Utilize cache (LUT)
  - Extensive use of lookup tables (LUT): trade memory for calculations; still well fit into L2 cache
  - Applicable to more than half of the common alg; speedup ranges 1.5x-22x

- Example:
Sora Software: Acceleration (SIMD)

- Utilize data para. (SIMD)
  - Modern CPUs support wide-vector SIMD ext.
    - Intel SSE 128 packed vector, allowing 8 x 16bit ops
  - Applicable to many PHY alg. w/ speedups (1.6x-50x)
- Example: FIR
## Sora Software: Acceleration

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Configuration</th>
<th>I/O Size (bit)</th>
<th>Optimization Method</th>
<th>Computation Required (Mcycles/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Input</td>
<td>Output</td>
<td>Conv. Impl.</td>
</tr>
<tr>
<td>Scramble</td>
<td>11Mbps</td>
<td>8</td>
<td>8</td>
<td>LUT</td>
</tr>
<tr>
<td>Descramble</td>
<td>11Mbps</td>
<td>8</td>
<td>8</td>
<td>LUT</td>
</tr>
<tr>
<td>Mapping and Spreading</td>
<td>2Mbps, DQPSK</td>
<td>8</td>
<td>44<em>16</em>2</td>
<td>LUT</td>
</tr>
<tr>
<td>CCK modulator</td>
<td>5Mbps, CCK</td>
<td>8</td>
<td>8<em>16</em>2</td>
<td>LUT</td>
</tr>
<tr>
<td></td>
<td>11Mbps, CCK</td>
<td>8</td>
<td>8<em>16</em>2</td>
<td>LUT</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>16-bit I/Q, 37 taps, 22MSps</td>
<td>16<em>2</em>4</td>
<td>16<em>2</em>4</td>
<td>SIMD</td>
</tr>
<tr>
<td>Decimation</td>
<td>16-bit I/Q, 4x Oversample</td>
<td>16<em>2</em>4*4</td>
<td>16<em>2</em>4</td>
<td>SIMD</td>
</tr>
</tbody>
</table>

### IEEE 802.11a

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Configuration</th>
<th>I/O Size (bit)</th>
<th>Optimization Method</th>
<th>Computation Required (Mcycles/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Input</td>
<td>Output</td>
<td>Conv. Impl.</td>
</tr>
<tr>
<td>FFT/IFFT</td>
<td>64 points</td>
<td>64<em>16</em>2</td>
<td>64<em>16</em>2</td>
<td>SIMD</td>
</tr>
<tr>
<td>Conv. Encoder</td>
<td>24Mbps, 1/2 rate</td>
<td>8</td>
<td>16</td>
<td>LUT</td>
</tr>
<tr>
<td></td>
<td>48Mbps, 2/3 rate</td>
<td>16</td>
<td>24</td>
<td>LUT</td>
</tr>
<tr>
<td></td>
<td>54Mbps, 3/4 rate</td>
<td>24</td>
<td>32</td>
<td>LUT</td>
</tr>
<tr>
<td>Viterbi</td>
<td>24Mbps, 1/2 rate</td>
<td>8*16</td>
<td>8</td>
<td>SIMD+LUT</td>
</tr>
<tr>
<td></td>
<td>48Mbps, 2/3 rate</td>
<td>8*24</td>
<td>16</td>
<td>SIMD+LUT</td>
</tr>
<tr>
<td></td>
<td>54Mbps, 3/4 rate</td>
<td>8*32</td>
<td>24</td>
<td>SIMD+LUT</td>
</tr>
<tr>
<td>Soft demapper</td>
<td>24Mbps, QAM 16</td>
<td>16*2</td>
<td>8*4</td>
<td>LUT</td>
</tr>
<tr>
<td></td>
<td>54Mbps, QAM 64</td>
<td>16*2</td>
<td>8*6</td>
<td>LUT</td>
</tr>
<tr>
<td>Scramble &amp; Descramble</td>
<td>54Mbps</td>
<td>8</td>
<td>8</td>
<td>LUT</td>
</tr>
</tbody>
</table>
Sora Software: Multicore Streaming

- Partition and schedule PHY processing across cores
  - Interconnecting sub-pipeline w/ light-weight sync. FIFOs
  - Static scheduling of processing modules in PHY pipeline
Interesting/key software design techniques you learned from Sora?
Outline

- Recap
- Wireless background
  - Frequency domain
  - Modulation and demodulation
  - Wireless channels
  - Wireless PHY design
  - Wireless MAC design (one hop)
  - From single-hop MAC to multi-hop mesh
- Wireless and mobile software framework
  - GNURadio
  - SORA
  - 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
Requirements on Software Dev. Framework

- Flexible configuration of attached devices

- Small footprint
  - Devices have limited memory and power resources