### Radio Protocols
- UHF (300-1000Hz)
  - Mote radio
- Bluetooth (2.4GHz)
  - Common in many consumer devices (PDAs, cell phones, etc.)
- Zigbee (850-930MHz)
  - Next generation radio for sensor networks and consumer devices

### Mote Radio
- ChipCon CC1000
  - Single-chip RF transceiver
  - Programmable frequency (300-1000 MHz)
  - Very low current consumption (Rx: 7.4 mA, Tx: 10.4 mA)
  - Very few external components required
  - FSK (frequency-shift-key) modulation spectrum shaping
  - Manchester encoded data
  - Low supply voltage (2.1 - 3.6 V)
  - High receiver sensitivity (-110 dBm)
  - RSSI output
  - FSK data rate up to 76.8 kBaud (motes use 38.4Kb)
  - Programmable frequency in 250 Hz steps
  - Suitable for frequency hopping protocols
  - Single-port antenna connection
  - Small 28-pin TSSOP package

### Chipcon CC1000 Block Diagram

### Radio on Motes
- Integrated CC1000 radio package
- Connection to ATmega microcontroller through SPI bus
  - CC1000 is master
  - Rx is double buffered, Tx is single buffered
- ATmega interfaces at the bit level
  - On transmit, new bits must be provided at the right rate
  - On receive, new bits must be collected at the right rate
  - CC1000 has a byte-wide interface
    - At 38.4Kbps, a new byte every ~8*25 ms = 200 ms
    - Corresponds to approximately 1600 assembly instructions at 8MHz
    - Bounds the length of interrupt service routines
    - Older versions of motes used a bit interface to the radio

### RF Frequencies & Channels
- Industrial, scientific, and medical (ISM) bands
  - 868 to 870 in Europe and Asia
  - 902 to 928 MHz in US
  - 433.1 to 434.8 MHz in US and Europe
  - 313.9 to 316.1 MHz in Asia
- Other unregulated frequency bands
  - 2.4GHz (Bluetooth, 802.11b)
  - 5.8GHz (802.11a)
- Mote is manufactured for specific band
  - Discrete components on board set operating frequency

### RF Modulation - FSK
- Frequency shift keying
  - One of many possible modulation schemes
  - 0 and 1 represented by two different frequencies slightly offset from a center carrier frequency (average)
  - At 38.4Kbps, ~10,000 periods for one bit
**Data Encoding**

- Manchester encoding
  - Every bit, whether a 0 or 1, has a transition
  - Guarantees there will never be a run of 0 or 1
  - Ensures stable clock recovery at receiver
  - Recovered clock determines sampling time of data bits
- Implemented in CC1000 hardware
  - Reduced ATMega128 overhead

**Radio Antenna**

- Simple ¼ wave monopole whip antenna is sufficient for most uses
  - 916Mhz → 3.2″ wire length
  - 433Mhz → 6.8″ wire length
- Equivalent to half wave dipole antenna

**Antennas and Radio Transmission**

- Polarization
  - Vertical orientation of all antennas in a system is best
  - 1/10th the distance if some antennas are vertical, some horizontal
- Transmission Near the Ground
  - Mica2 916Mhz, 3′ above ground
    - ~300′ line of sight, 30′ on the ground
  - Mica2 433 Mhz, 3′ above ground
    - ~500′ line of sight, 150′ on the ground

**RF Propagation**

- Line of sight
  - Direct path from transmitter to receiver
  - Free space attenuation 1/d^2
    - To double distance, it needs 4x power
- Reflection
  - Off objects large compared to wavelength
    - walls, buildings
- Scattering
  - Off objects smaller than wavelength
    - foliage, chairs

**Multi-path Effects**

- Path length variations
  - Delayed version of signal arrives at receiver
- Path Attenuation
  - Various signal strengths
- Result looks like distortion or interference at receiver
- Out of phase signal interference can create nulls
  - Zero or severely reduced signal strength
  - Can exist even very close

**Indoor Propagation**

- Rapid signal attenuation closer to 1/d^3
- People moving around cuts range by 1/3
- Concrete/steel flooring by 1/4
- Metallic tinted windows by 1/3
Dynamic Fading Effects

- People moving, doors opening and closing (esp. if metal)
  - E.g., closing doors in lab changes strong (green) RF regions to weak (blue) RF regions

RF Link Metrics

- Packet loss
  - Determined at application layer (your code)
- Bit error rate
  - Determined at the link layer – incorrect checksums (TinyOS)
- Low RSSI – received signal strength indicator
  - Determined at the physical layer (CC1000)

Data transport – packetized data

- 18 byte preamble – alternating 1010… pattern for clock recovery
- 2 byte frame sync – indicates start of data packet
- 36 bytes of TinyOS packet – data payload including CRC

Common RF Link Problems

- Signal strength
  - Weak, overload
- Collisions
  - Other motes (independent of GroupID)
- Interference from other sources
  - Cross-talk from adjacent RF channels
  - Other devices on same frequency
    - E.g., cordless phones at 900MHz
- Multi-path
  - Nulls are especially problematic as they can shift location

Data Packets on the Mica2 Platform
TinyOS Message Structure

- **Header**
  - Address (2 bytes)
  - Active Message Type (1 byte)
    - Indicates which handler to use to process message
  - Group ID (1 byte)
    - Adds to address space but provides a way to broadcast to a group
  - Payload Length (1 byte)
- **Payload**
  - 29 bytes user/application defined data
- **CRC**
  - 2 bytes

TinyOS Radio Stack

- **Stack Layers**
  - Application
  - Routing
  - Active Message
  - Packet
  - Byte
- **Lowest-level radio interface**
  - Data/noise streams in from CC1000 at 19.2Kb/s rate
  - SPI Input Interrupt every 416 μs (8 bits)
  - CC1000 handles the serialization and physical layers

TinyOS Radio Stack (cont’d)

- **GenericComm**
- **ActiveMessages (AM)**
- **CC1000RadioC**
- **CC1000RadioHM**
- **SPIByteFIFO**
- **RandomLFSR**
- **ADC**

TinyOS Packet Reception

- RadioIntM.nc SPI Port Interrupt Handler
  - Search for preamble pattern (10101…) 
  - Wait for frame sync word (two bytes)
  - Assemble packet
  - Check CRC – reject if bad
  - Route to active message handler
  - Check Group ID – reject if not member
  - Signal application with ReceivedMsg event

TinyOS Packet Transmission

- Packet is routed
  - GenericComm
  - AM Handler – RF or UART
  - TinyOS CC1000RadioIntM
  - Random delay (0-15 packet times)
  - Check for carrier
  - Turn on transmitter
  - Send
    - 18-byte preamble (10101… pattern) & frame SYNC
    - Packet (34 bytes) – address, Group ID, type, length, and data payload
    - CRC (2 bytes)
  - Turn off transmitter
  - Signal TxDone event to application

CC1000Radio States
SYNC State

- Shift RX Byte bit-wise into Word Buffer
  - Word Buffer == SYNC PATTERN?
    - Byte Align = Bit Shift Count
    - Next State = RX
  - RX Byte Count > MAX LENGTH?
    - Next State = IDLE

RX State

- RXBuffer[RXCount] = RX Data
- RXCount++
- Compute CRC(RXByte)
- RXCount == RXBuffer[Length]+Header?
  - CRC = RXBuffer[CRC]?
    - Post PACKETRECEIVED
    - Next state = IDLE
  - Error?
    - Next state = IDLE

TinyOS Radio Controls

- Frequency
  - Frequency Band / RF Channel Choices
    - #define CC1K_DEFAULT_FREQ 0x00
  - Specify CC1K_DEFAULT_FREQ in makefile
    - CFLAGS -d:CC1K_DEFAULT_FREQ CC1K_433_002_MHZ
- Power on/off
  - Sleep -2µA
  - Radio signal strength (RSSI valid) -20µsec
  - Receiver packet acquire time -3ms/8sec
  - Re-tune radio after a power off/on cycle
    - command result_t Tune(uint8_t freq);
- RF Power Level
  - 0xFF is 5dBm
  - 0x80 is 0 dBm (1mW)
  - 0x09 is -10dBm
    - command result_t SetRFPower(uint8_t power);

Important RF Issues

- Re-tune after sleep or temperature changes
- Remember multi-path effects can occur
- Different GroupIDs do NOT prevent RF interference
- Radio Debugging Hints
  - Correct Radio Frequency?
    - CC1K_DEFAULT_FREQ
  - Correct GroupID?
  - GenericBase Hangup
    - Press RESET button
  - RF Null Location?
    - Move mote to different location (+/- 1m)
  - RF Overload
    - Separation >2m

Example

- CntToLedsAndRfm
  - Display a number on LEDs
  - Send it to another mote over the radio

```c
configuration CntToLedsAndRfm {
  Implementation {
    components Main, Counter, IntToLeds, IntToRfm, TimerC;
    Main.StdControl -> Counter.StdControl;
    Main.StdControl -> IntToLeds.StdControl;
    Main.StdControl -> IntToRfm.StdControl;
    Counter.Timer -> TimerC.Timer[unique("Timer")].Timer;
    Counter.IntOutput -> IntToLeds.IntOutput;
    Counter.IntOutput -> IntToRfm.IntOutput;
    CntToLedsAndRfm.range[0..4] -> CntToLedsAndRfm.range[0..4];
  }
}
```

CntToLedsAndRfm Components

- Configuration Outline:
  - Provides interface (interface); provides interface (interface);
  - Implementation:
    - Components [components, (components)];
    - Components [components, (components)];
    - Components [components, (components)];.

Another Application

- Surge
  - Forms a multi-hop ad-hoc network of nodes
  - Each node takes light readings and sends them to a base station
  - Each node also forwards messages of other nodes
  - Designed to be used in conjunction with the Surge java tool
  - The node also responds to broadcast commands from the base
SurgeM.nc

```c
command result_t StdControl.init() {
    timer_rate = INITIAL_TIMER_RATE;
    atomic gfSendBusy = FALSE;
    sleeping = FALSE;
    rebroadcast_adc_packet = FALSE;
    focused = FALSE;
    return SUCCESS;
}

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, timer_rate);
    return SUCCESS;
}

command result_t StdControl.stop() {
    return call Timer.stop();
}

event result_t Timer.fired() {
    timer_ticks++;
    if (timer_ticks % TIMER_GETADC_COUNT == 0) {
        call ADC.getData();
    } // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
        call Sounder.start();
    } // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
        call Sounder.stop();
    } // If we're the focused node, chirp
    return SUCCESS;
}

async event result_t ADC.dataReady(uint16_t data) {
    atomic {
        if (!gfSendBusy) {
            gfSendBusy = TRUE;
            gSensorData = data;
            post SendData();
        }
    }
    return SUCCESS;
}

task void SendData() {
    SurgeMsg *pReading;
    uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Sending sensor reading\n" );
    if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
        pReading->type = SURGE_TYPE_SENSORREADING;
        pReading->parentaddr = call RouteControl.getParent();
        pReading->reading = gSensorData;
        if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
            atomic gfSendBusy = FALSE;
    }
}

event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen) {
    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *) payload;
    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer rate
        timer_rate = pCmdMsg->args.newrate;
        call Timer.stop(); call Timer.start(TIMER_REPEAT, timer_rate);
    } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
        sleeping = TRUE;
        call Timer.stop();
        call Leds.greenOff(); call Leds.yellowOff();
    } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
        if (sleeping) {
            initialize();
            call Timer.start(TIMER_REPEAT, timer_rate);
            sleeping = FALSE;
        }
    } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
        if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
            focused = TRUE;
            call Sounder.init();
            call Timer.stop(); call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
        } else {
            call Timer.stop(); call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
        }
    } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
        focused = FALSE;
        call Sounder.stop();
        call Timer.stop(); call Timer.start(TIMER_REPEAT, timer_rate);
    }
    return pMsg;
}
```

TinyOS Active Messages (Sending)

- **Sending using AMStandard**
  - Get a region in memory for packet buffer
  - Form packet in the buffer
  - Assign active message type for proper handling
  - Request transmission
  - Handle completion signal

```c
command result_t SendMsg.send(uint8_t id, uint16_t addr, uint8_t length, TOS_MsgPtr data) {
    if (!state)
        state = TRUE;
    if (length > DATA_LENGTH) {
        dbg(DBG_AM, "AM: Send length too long: %i. Fail.\n", (int)length);
        state = FALSE;
        return FAIL;
    }
    buffer->group = TOS_AM_GROUP;
    dbg(DBG_AM, "Sending message: %hx, %hhx\n", addr, id);
    if (!state)
        return FAIL;
    buffer = data;
    data->length = length;
    data->addr = addr;
    data->type = id;
    buffer->group = TOS_AM_GROUP;
    data->length = length;
    data->addr = addr;
    data->type = id;
    buffer->group = TOS_AM_GROUP;
    return SUCCESS;
}

command result_t SendMsg.send(TOS_MsgPtr pMsg, result_t success) {
    TOS_MsgPtr buf;
    buf = pMsg;
    return SUCCESS;
}
```

TinyOS Active Messages (Receiving)

- **Receiving using AMStandard**
  - Declare a handler to perform action on message event
  - Active message automatically dispatched to associated handler
    - Known format
    - No run-time parsing
  - Buffer management
    - Must return free buffer to the system for the next packet reception
    - Typically the incoming buffer once processing is complete

```c
AMStandard.c (send)
```
TOS_MsgPtr received(TOS_MsgPtr packet) {
    uint16_t addr = TOSLOCAL_ADDRESS;
    counter++;
    if (packet->crc == 1 && packet->group == TOS_AM_GROUP &&
        (packet->addr == TOS_BCAST_ADDR || packet->addr == addr)) {
        uint8_t type = packet->type;
        TOS_MsgPtr tmp;
        tmp = signal ReceiveMsg.receive[type](packet);
        if (tmp) packet = tmp;
    }
    return packet;
}

default event TOS_MsgPtr ReceiveMsg.receive[TOS_MsgPtr msg] {
    return msg;
}
event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr packet) {
    packet->group = TOS_AM_GROUP;
    return received(packet);
}
event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr packet) {
    return received(packet);
}

Platform Folder

- Location of details of the Hardware Layer
  - Most files have the HPL prefix
  - Each type of platform has its own subfolder where platform specific files are pulled from.
    - (e.g. HPLUARTM, CC1000Radio, HPLDCM)
  - ‘platform’ file in platform directory
  - Lists common platforms
  - Allows compiler to pull from those platform directories second.
- ‘hardware.h’ is where the pins are mapped
- ‘avhardware.h’ is where the macro’s are defined

hardware.h

// LED assignments
TOSH_ASSIGN_PIN(RED_LED, A, 2);

// ChipCon control assignments
TOSH_ASSIGN_PIN(C1_PIN, D, 3); // ChipCon C1_PIN
TOSH_ASSIGN_PIN(C0_PIN, D, 2); // ChipCon C0_PIN
TOSH_ASSIGN_PIN(C4_PIN, D, 7); // Chipcon C4_PIN
TOSH_ASSIGN_PIN(C5_PIN, D, 5); // Chipcon C5_PIN

// PWM assignments
TOSH_ASSIGN_PIN(PWM1B, B, 6);

avhardware.h

#define TOSH_ASSIGN_PIN(name, port, bit)
static inline void TOSH_SET_##name##_PIN() {sbi(PORT##port, bit);}
static inline void TOSH_CLR_##name##_PIN() {cbi(PORT##port, bit);}
static inline int TOSH_READ_##name##_PIN() {return (inp(PIN##port) & (1 << bit)) != 0;}
static inline void TOSH_MAKE_##name##_OUTPUT() {sbi(DDR##port, bit);}
static inline void TOSH_MAKE_##name##_INPUT() {cbi(DDR##port, bit);}

For TOSH_ASSIGN_PIN(PWM1B, B, 6);
Yields:
static inline void TOSH_SET_PWM1B_PIN() {sbi(PORTB, 6);}
static inline void TOSH_CLR_PWM1B_PIN() {cbi(PORTB, 6);}
static inline int TOSH_READ_PWM1B_PIN() {return (inp(PINB) & (1 << 6)) != 0;}
static inline void TOSH_MAKE_PWM1B_OUTPUT() {sbi(DDRB, 6);}
static inline void TOSH_MAKE_PWM1B_INPUT() {cbi(DDRB, 6);}

Packet Buffers

- Don’t reuse until send completes
- Your Module
  - Send
  - Radio Stack
- TOS_Msg Buffer 1
  - Declared in Your Module
- TOS_Msg Buffer 2
  - Declared in Radio Stack

Pin Assignments

- Macros used to declare pins
  - TOSH_ASSIGN_PIN(RED_LED, A, 2);
- This gives a set of macro’s that can be called
  - TOSH_SET_RED_LED_PIN()
  - TOSH_CLR_RED_LED_PIN()
  - TOSH_MAKE_RED_LED_OUTPUT()
  - TOSH_MAKE_RED_LED_INPUT()
## Bluetooth

- Short-range radio at 2.4GHz
  - Available globally for unlicensed users
  - Low-power
  - Low-cost
  - Cable replacement
  - Devices within 10m can share up to 1Mb/sec – 700Kb/sec effective
- Universal short-range wireless capability

### Bluetooth Application Areas

- Data and voice access points
  - Real-time voice and data transmissions
  - Cordless headsets
  - Three in one phones: cell, cordless, walkie-talkie
- Cable replacement
  - Eliminates need for numerous cable attachments for connection
  - Automatic synchronization when devices within range
- Ad hoc networking
  - Can establish connections between devices in range
  - Devices can "imprint" on each other so that authentication is not required for each instance of communication
  - Support for object exchange (files, calendar entries, business cards)

## Bluetooth Standards Documents

- Core specifications
  - Details of various layers of Bluetooth protocol architecture
  - Emphasis on physical and transport layers
- Profile specifications
  - Use of Bluetooth technology to support various applications
  - Examples include point-to-point audio and local area network

## Protocol Architecture

- Bluetooth is a layered protocol architecture
  - Core protocols
  - Cable replacement and telephony control protocols
  - Adopted protocols
- Core protocols
  - Radio
  - Baseband
  - Link manager protocol (LMP)
  - Logical link control and adaptation protocol (L2CAP)
  - Service discovery protocol (SDP)

## Bluetooth Stack Overview

- Cable replacement protocol
  - RFCOMM
  - Telphery control protocol
  - Telephone control specification – binary (TCS BIN)
- Adopted protocols
  - PPP
  - TCP/UDP/IP
  - OBEX
  - WAP
- Profiles – vertical slide through the protocol stack
  - Basis of interoperability
  - Each device supports at least one profile
  - Defined based on usage models
    - e.g., headset, camera, personal server, etc.
**Piconets and Scatternets**

- **Piconet**
  - Basic unit of Bluetooth networking
  - Master and up to 7 slave devices
  - Master determines channel and phase
- **Scatternet**
  - Device in one piconet may exist as master or slave in another piconet
  - Allows many devices to share same area
  - Makes efficient use of bandwidth

**Wireless Network Configurations**

**Radio Specification**

- **Classes of transmitters**
  - Class 1: Outputs 100 mW for maximum range
    - Power control mandatory
    - Provides greatest distance
  - Class 2: Outputs 2.4 mW at maximum
    - Power control optional
  - Class 3: Nominal output is 1 mW
    - Lowest power

**Frequency Hopping in Bluetooth**

- Provides resistance to interference and multipath effects
- Provides a form of multiple access among co-located devices in different piconets

**Frequency Hopping**

- Total bandwidth divided into 1MHz physical channels
- Frequency hopping occurs by moving transmitter/receiver from one channel to another in a pseudo-random sequence
- Hopping sequence shared with all devices in the same piconet so that they can hop together and stay in communication

**Physical Links between Master - Slave**

- **Synchronous connection oriented (SCO)**
  - Allocates fixed bandwidth between point-to-point connection of master and slave
  - Master maintains link using reserved slots
  - Master can support three simultaneous links
- **Asynchronous connectionless (ACL)**
  - Point-to-multipoint link between master and all slaves
  - Only single ACL link can exist
Bluetooth Packet Fields

- Access code
  - timing synchronization, offset compensation, paging, and inquiry
- Header
  - identify packet type and carry protocol control information
- Payload
  - contains user voice or data and payload header, if present

Channel Control

- States of operation of a piconet during link establishment and maintenance
- Major states
  - Standby – default state
  - Connection – device connected

Interim substates for adding new slaves

- Page – device issued a page (used by master)
- Page scan – device is listening for a page
- Master response – master receives a page response from slave
- Slave response – slave responds to a page from master
- Inquiry – device has issued an inquiry for identity of devices within range
- Inquiry scan – device is listening for an inquiry
- Inquiry response – device receives an inquiry response

States of operation of a piconet during link establishment and maintenance

- Connection
- Inquiry
- Page
- Master response
- Slave response
- Inquiry response
- Keep-alive
- Disconnection

Scenario steps

- Master device (e.g., PDA) pages for nearby devices
  - Receives response from 0, 1, or more devices
    - Slave device (e.g., headphone) responds to page
  - Determines which it "knows" – established connections
  - L2CAP establishes Bluetooth connection assigning paging device to be master
  - Devices exchange profiles they both support
  - Agree upon profile (e.g., audio streaming)
  - Master sends audio data
    - Two devices synchronize their frequency hopping
  - Keep-alive packets used to maintain connections
  - Connections dropped if keep-alive packets are not acknowledged

Limitations/Issues

- Discovery time on the order of 10sec for unknown devices
- Interaction with user required to connect to unknown devices or if multiple masters
- Can connect 8 devices at a time, more need to be multiplexed radically lowering throughput
- Doesn’t support simple broadcast – need to be on same frequency hopping schedule
- Effective bandwidth closer to 500Kbps (within one scatternet, order of magnitude lower if between two)
Zigbee (adapted from www.zigbee.org)

- Simpler protocol
- Broadcast support
- Network support (rather than point-to-point)
- Very low power (batteries that last years)
- Consumer device networks
  - Remote monitoring and control
  - Low-cost, low-complexity
  - Support ad-hoc and mesh networking
- Industry consortium
- Builds on IEEE standard 802.15.4 physical radio standard – OQSK
  - Add logical network, security and application software
- 250Kb/sec bandwidth ~ 128Kb/sec effective, 30m range

The Wireless Market

![Graph showing the wireless market with Zigbee and Bluetooth]

Protocol Stack Features

- 8-bit microcontroller
- Compact protocol stack <32KB
- Supports even simpler slave-only stack <4KB
- Coordinator requires extra memory for storing association tables

Zigbee Networks

- 65,536 network (client) nodes
- Optimized for timing-critical applications
  - Network join time: 30 ms (typ)
  - Sleeping slave changing to active: 15 ms (typ)
  - Active slave channel access time: 15 ms (typ)
- Traffic types
  - Periodic data (e.g., sensor)
  - Intermittent data, event (e.g., light switch)
  - Low-latency, slotted (e.g., mouse)

Zigbee Networks (cont’d)

![Diagram showing Zigbee network architecture]
**Lighting Control**

- Advance Transformer
  - Wireless lighting control
  - Dimmable ballasts
  - Light switches anywhere
  - Customizable lighting schemes
  - Energy savings on bright days
  -DAL (or other) interface to BMS
- Extendable networks
  - Additional sensors
  - Other networks

**HVAC Energy Management**

- Hotel energy management
  - Major operating expense for hotels
  - Centralized HVAC management allow hotel operators to make sure empty rooms are not cooled
  - Retrofit capabilities
  - Battery operated thermostats can be placed for convenience
  - Personalized room settings at check-in

**Asset Management**

- Within each container, sensors form a mesh network.
- Multiple containers in a ship form a mesh to report sensor data
- Increased security through on-truck and on-ship tamper detection
- Faster container processing. Manifest data and sensor data are known before ship docks at port.

**Residential Control**

**Residential Example**

**Comparison of Complementary Protocols**

<table>
<thead>
<tr>
<th>Feature(s)</th>
<th>IEEE 802.11b</th>
<th>Bluetooth</th>
<th>ZigBee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>11Mbps</td>
<td>11Mbps</td>
<td>11Mbps</td>
</tr>
<tr>
<td>Latency</td>
<td>2.5ms</td>
<td>2.5ms</td>
<td>2.5ms</td>
</tr>
<tr>
<td>Range</td>
<td>1000 ft</td>
<td>1000 ft</td>
<td>1000 ft</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>11 Mbps</td>
<td>11 Mbps</td>
<td>11 Mbps</td>
</tr>
<tr>
<td>Data Rate</td>
<td>11Mbps</td>
<td>11Mbps</td>
<td>11Mbps</td>
</tr>
<tr>
<td>Authentication</td>
<td>10 Mbps (AES)</td>
<td>10 Mbps</td>
<td>10 Mbps</td>
</tr>
<tr>
<td>Encryption</td>
<td>64 bit, 128 bit</td>
<td>64 bit, 128 bit</td>
<td>64 bit, 128 bit</td>
</tr>
<tr>
<td>Protocol Security</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame Length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HVAC control in building automation**
<table>
<thead>
<tr>
<th>Wireless Network Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point to Point</strong></td>
</tr>
<tr>
<td>- Simple wire replacement</td>
</tr>
<tr>
<td>- Direct Connection between devices</td>
</tr>
<tr>
<td>- Limited communication</td>
</tr>
<tr>
<td><strong>Point to Multi-Point</strong></td>
</tr>
<tr>
<td>- Centralized routing and control point</td>
</tr>
<tr>
<td>- Examples include: Wi-Fi, GSM, Bluetooth</td>
</tr>
<tr>
<td>- All data must flow through &quot;base station&quot;</td>
</tr>
<tr>
<td><strong>Multi-hop/Mesh</strong></td>
</tr>
<tr>
<td>- Full RF redundancy, with multiple data paths</td>
</tr>
<tr>
<td>- Self Configuring / Self Healing</td>
</tr>
<tr>
<td>- Distributed Intelligence</td>
</tr>
</tbody>
</table>

---

15