Adaptive Frequency Hopping

Joel Linsky

Senior Bluetooth System Architect
Silicon Wave, Inc.
11 December 2002

jlinsky@siliconwave.com
Coexistence Working Group Status

- AFH Improvement Proposal was approved last week and released to Associate Bluetooth SIG Members for comment
  - Document available on www.bluetooth.org

- Scheduled to be in next version of the Bluetooth Core Specification

- The Coexistence WG would appreciate comments on the Improvement Proposal
  - coexist-wg-feedback@bluetooth.org
Design Philosophy

- Backwards compatible with BT1.1
- Architectural consistency with BT1.1
- Support of mixed and non-mixed piconets
- Ability to have a BT1.1 Host
- Forward compatibility with BT High Rate
What is Adaptive Frequency Hopping?

- Adaptive Frequency Hopping (AFH) means that a device can hop over a reduced set of frequencies
  - FCC allows reduction from 75 down to 15 frequencies
  - Baseline BT1.1 hopping uses 79 frequencies

- Made legal for 1MHz bandwidth devices by NPRM 01-158 this year
Why AFH?

- AFH allows Bluetooth devices to perform better in the presence of frequency ‘static’ interference
  - 802.11b/g
  - Cordless phones
  - Microwave ovens
  - Other technologies in the future

- AFH allows Bluetooth devices to interfere less with other frequency ‘static’ devices in the ISM band
The Hopping Kernel

Properties:
- Overlay
- Similar pseudo-random properties as the BT1.1 hopping kernel
- Backwards compatible with BT1.1 hopping kernel

Relatively minor additional complexity

Hopping pattern is a defined sequence specified by the master and communicated to slaves in the piconet
Enabling AFH

◆ **Connection setup**
  - AFH can be enabled at the start of a connection using LMP commands
  - Normally, AFH would be enabled on slaves that support AFH

◆ **Un-Park**
  - When devices are in park they must be unparked to update AFH parameters
  - For devices that do not want this complexity, AFH could be disabled for devices in park
Example 1: Adding AFH Capable Slaves to Piconet

Step 1 – Master Pages Slave
Example 1: Adding AFH Capable Slaves to Piconet

Step 2 – BT1.1 Connection Established
Example 1: Adding AFH Capable Slaves to Piconet

Step 3 – AFH Enabled (via LMP)
Example 1: Adding AFH Capable Slaves to Piconet

Step 4 – Master Pages 2nd Slave
Example 1: Adding AFH Capable Slaves to Piconet

Step 5 – 2nd BT1.1 Connection Established
Transient State with two Hopping Sequences
Example 1: Adding AFH Capable Slaves to Piconet

Step 6 – AFH Enabled on 2nd Slave (via LMP)
Steady State with one Hopping Sequence
Example 2: Adding one AFH Capable Slave and one BT1.1 Slave to a Piconet

Step 1 – Master Pages Slave
Example 2: Adding one AFH Capable Slave and one BT1.1 Slave to a Piconet

Step 2 – BT1.1 Connection Established
Example 2: Adding one AFH Capable Slave and one BT1.1 Slave to a Piconet

Step 3 – AFH Enabled
Example 2: Adding one AFH Capable Slave and one BT1.1 Slave to a Piconet

Step 4 – Master Pages 2nd Slave
Example 2: Adding one AFH Capable Slave and one BT1.1 Slave to a Piconet

Step 5 – 2nd BT1.1 Connection Established
Steady State with two Hopping Sequences
Updating AFH

- Interference may appear, so master may remove channels
  - Mobile device moves into a location with interference
  - Duty cycle of nearby interferer increases

- Interference may disappear, so master may add channels
  - Bluetooth device is mobile and moves away from interference
  - Interference disappears or its duty cycle reduces

- Master can update set of used channels (via LMP)

- Slaves may be on the same hop sequence or different hop sequences
Example 3: Updating AFH Capable Slaves to Piconet

Step 1 – All AFH capable slaves on Hop Sequence A
Example 3: Updating AFH Capable Slaves to Piconet

Step 2 – New AFH Parameters Sent to AFH Capable Slaves (via LMP)
Disabling AFH

- May be done at the end of a connection prior to disconnect
  - Not necessary

- May be done in a mixed piconet and the master only wants to support one hopset
  - Complexity issue

- Park
  - It can be easier to manage AFH in the rest of the piconet if parked slaves are taken out of AFH
Example 4: Disabling AFH Prior to Parking a Slave

Step 1 – Steady State with one Hopping Sequence
Example 4: Disabling AFH Prior to Parking a Slave

Step 2 – Master Disables AFH on Slave to be Parked
Example 4: Disabling AFH Prior to Parking a Slave

Step 3 – Master Parks Slave
Example 4: Disabling AFH Prior to Parking a Slave

Step 4 – Master Unparks Slave
Example 4: Disabling AFH Prior to Parking a Slave

Step 5 – Master Re-enables AFH to 2\textsuperscript{nd} Slave
Channel Assessment Sources

- **Passive assessment**
  - Devices can probe channels (e.g. RSSI) to look for interference
  - Does not interfere with other devices in ISM band
Channel Assessment Sources

- **Active assessment**
  - Devices can look at RX packets and correlate errors with channels
  - Needs packets to evaluate channel
Channel Assessment Sources

- **AFH Master**
- **AFH Capable Slave**
- **Passive Probe**
- **Host (HCI)**

Devices can receive information from Host (via HCI)
Slaves can also perform channel assessment
Channel Assessment Sources

- **AFH Master**
- **Passive Probe**
- **Host (HCI)**
- **Rx Errors**
- **Classification Report (LMP)**
- **AFH Capable Slave**
- **Passive Probe**

- Slaves periodically report channel classifications to master (via LMP)
Channel Assessment

- Evaluation methodologies will be proprietary
- There may be algorithms and performance requirements recommended by Coexistence WG

**Periodic Slave Reporting**
- Slaves can be configured to periodically send channel classification reports to the master
- Slaves only send information if there is a change, so there is little LMP traffic
- Periodicity is set by the master
Does AFH Solve Coexistence Problems?

- AFH gives Bluetooth devices the ability to select frequency ranges that avoid other sources of interference in the ISM band.

- This only works if there is enough antenna isolation.

- For some applications AFH is enough.

- For others, it must be used with other complimentary technologies.
Thank You