Bulk-transfer protocols using SP

Joe Polastre          Jonathan Hui

February 21, 2005

Problem

Build a network protocol that sends bulk data through the network when requested, or sets up an epoch schedule for waking up the network as a whole and sending data.

In this document we describe the tradeoffs and primitives needed to build a bulk transfer collection protocol; however we do not present interfaces or APIs since the discussion presented herein is more of a general approach and applicable to any number of systems.

Requirements of this network protocol

Timing information

Time stamping information is crucial for synchronized network protocols. SP must expose the SFD time stamp information to services using SP. This method seems acceptable and widely in use today (ref: TinyOS, VU, & Zigbee).

Message quantity

Notification of bulk transfer. A network service must be able to indicate to SP that it has more than one message to transmit. A simple queue/buffer method may not be appropriate because the messages may be stored elsewhere, ie external storage/log.

1. Idea: only single-message notion, but service can react quickly upon completion of message send. SP asks what to do at end of message transmission, and service may indicate that it has another message to send.
   
   Pros: Simple for SP to manage, requires minimal neighbor state.
   
   Cons: Doesn’t allow sending packets to multiple neighbors because first “expensive” packet may incur address filtering and node returning to sleep if not for it.

2. Preferred Idea: Attribute-value pairs for each service are stored in a common pool, SP may read pairs and make decisions based on service requirements.
   
   Pros: More optimization possible due to more information provided by services.
   
   Cons: What are the right attribute-value pairs? More complex service interaction.

Radio management

The network protocol may need to wake up the radio at a specified time in order to receive data. The position may be that SP only can control the radio on and off times or that a downcall from the network service can actuate the radio. We break this up into three cases for transmission:
1. No explicit synchronization, Low Power Listening.

A node in the network elects itself to wake up the network (this may be a base station) at some interval. In order to do so, the leader floods out a long preamble wakeup message with a timeout. The network service receives the message, notifies SP to stay on for the remainder of the epoch, and submits messages to transmit during this time. Similarly, these messages may be stored in a queue, and marked as “ready-to-send” by SP (if knowledge is given to SP about which neighbors are awake) or the network service during the wakeup period.

2. Explicit synchronization, LPL preamble accordingly set by SP

Network service notifies SP that neighboring nodes are now on and messages in the queue may be sent. It must also notify SP of a timeout for turning off the radio. SP knows that since neighboring nodes are awake, long preamble need not be sent. Network service maintains synchronization through sync messages and SFD time stamping.

3. Underlying synchronized protocol, such as TDMA or S-MAC

SP must also notify services when the radio is turned on or off or must gather neighbor state information. If the radio cannot be turned on or off on-demand, then SP must carefully chose which messages to send during an active period determined by the underlying link protocol. The network service must have a method for requesting time or slots to a remote node(s). SP must manage which service is responsible for which time slots in a TDMA link protocol schedule. This advocates the use of an attribute-value pair system where the network service can tell SP its message traffic requirements a priori.

Conclusion: “Timeout” for turning the radio off is a key SP primitive for enabling this network service.

Service “Hood”

This is the mechanism to share data between multiple components and SP in order for SP to make intelligent data transfer decisions. The interface to this cross-layer communication/information-sharing service is sets of attribute-value pairs defined by each service. These pairs must be link-agnostic and define the service’s requirements. Some examples include: Pointers to messages to send, quantity of total messages, tolerable latency per message, grouping of messages, neighbor active and sleep times, minimum radio awake time in the next/current active period (“timeout”), etc.