1 Introduction

This is a proposed SP interface based on the needs of a simple converge-cast tree based protocol. We assume the protocol does not establish any communication schedule. Nodes sample data and send it at a regular interval $S$. We assume $S$ is a reasonably large value, so the network operates at only a tiny fraction of its capacity (e.g., $S = 1$ minute). The phase of these sense-and-sends is undetermined: all nodes may be synchronized (a testbed was rebooted through an ethernet or USB connection), completely unsynchronized (nodes were manually turned on or have drifted a lot), or something in between.

Each reading must reach the base station (as measured from time of sampling) within an interval $S \cdot d$, where $d$ is the depth of the node in the converge-cast tree.

We present an possible interface to SP, describe how SP could implement it on a low power listen-CSMA MAC as well as a TDMA MAC, then show how the protocol would use this interface to achieve its bounds.

For a cost model, we use the idea presented of three levels: low, medium, and high.

We do not consider reliability metrics in this interface. The assumption is that SP provides some kind of mechanism to improve reliability, but does not guarantee anything except some form of best effort.

We assume that unicast traffic can be snooped on by other nodes. This may involve disabling link-layer acks. I think the opposite approach could be taken (no snooping, with acks), but I’d need to think through it more. It would involve a much heavier-weight SP (as it would need to keep neighbor tables, etc.).

2 Interface

There are two interfaces: ScheduleSend and SendSP. The former allows a component to schedule regular communication. SP can use this information to combine communications into a more energy efficient schedule. The latter is what a component uses to send a packet. There’s also a basic Send interface that just sends a packet. This is so higher-level time schedulers can bypass the SP logic (more on that in the SP implementation section).

2.1 ScheduleSend

ScheduleSend has this interface. It’s essentially a communication aware Timer.

interface ScheduleSend {
    command error_t schedule(addr, period, jitter, lead);
    command void stop();
    event void sendTime(cost, jitter, lead);
}

1
The schedule() command allows a component to tell SP that it needs to send a packet at a regular interval. There could be an additional parameter, `num`, for sending more than one packet. I don’t know if that’s necessary, but it’s something to consider. The command has four parameters: the destination address, the period between transmissions (in some time base, e.g., ms), the acceptable jitter from this period, and the amount of lead time the component wants for notification that it’s time to send.

The addr parameter is necessary because depending on the MAC, SP might need to schedule differently based on the address. For example, in a TDMA MAC based on address, SP can’t tell the component when it can send unless it knows where it’s sending to.

The period parameter is self explanatory. The jitter parameter specifies how accurate the schedule must be. The timer may fire within jitter time units of the regular period (plus or minus).

The lead parameter specifies far before the actual time to transmit the timer should fire. This is in case the component needs to sample sensors or do other processing to generate its packet. If the component will be ready to immediately send a packet, it can set lead to 0 and just call send() in the event handler. SP can signal well before it thinks it can send the packet, to have it ready for opportunistic transmission.

The event handler tells the component how expensive it will be to send the packet, how much jitter was introduced and how much actual lead time there is. If it turns out that the cost is greater than the component cares to pay, then it can just not send. Or, it could keep track of some sense of how much it’s consuming (e.g., only send at cost High every fifth chance, etc.). SP does not promise that sending the packet will cost that much or less: the cost parameter merely expresses its best guess as to how high the cost will be.

SP can’t promise due to issues in CSMA protocols, unless cost will always be high in those circumstances. E.g., what if SP thinks it can send to another node at a low cost because it believes that node to be awake, but then CSMA causes transmission to be delayed for a long enough time that the other node may have gone to sleep?

### 2.2 SendSP

The SendSP interface is this:

```c
interface SendSP {
    command error_t send(addr, timeout, msg, cost);
    command void sendDone(msg, cost, error);
}
```

The first parameter, addr, is self explanatory. Note that if you request a schedule to one address then send to another, you will probably spend a lot more energy than you want to. The timeout parameter specifies the how long SP can wait before sending (if it can’t send before timeout, then it will not send). timeout is intended to be determined from terms such as the maximum acceptable jitter and the introduced jitter, lead time, etc.

The cost parameter is a pointer to the maximum cost the component is willing to pay in order to send the packet. If the actual cost is greater than this, SP will not send it.

The sendDone event notifies whether the packet was sent (error denotes the reason if not), and the actual cost spent. The cost sendDone states must be less than or equal to the cost specified by send.

### 2.3 Send

In addition to all of these cost aware SP sends, there is also a basic send, which tells SP to just send a packet. This is so higher level synchronization (e.g., Epochs, FPS) can work. This is just another instance of the SendSP interface, except that it does not use SP’s scheduling logic. Calling the basic send means “put this on the send queue now,” while calling the SendSP send means “please send this when you get a chance.”
3 Implementation

We describe how two different MACs could implement this SP interface.

3.1 CSMA/LPL

A mote is normally in a low power listening (LPL) state. It defers transmissions for as long as possible, until it thinks it can do so cheaply. The costs are as follows:

- **High**: The receiver is in LPL mode. The node has to transmit a long preamble to wake the receiver up.
- **Medium**: The receiver is in LPL mode, but there are multiple packets waiting to be sent to it. The cost of the long preamble can therefore be amortized over several transmissions, giving all of them a cost of medium.
- **Low**: The receiver is fully awake. The node can transmit a short preamble.

Note that in this model, a regular transmission that must be precisely scheduled can very well have a cost of high, but other transmissions that can stand large jitter can just wait for this high transmission and have a cost of low. The distinction between all of them being medium and high/low is whether the low packets are promised to be transmitted at that time: SP could delay them further.

SP maintains guesses on the listening state of a finite set of neighbors. It bases its guesses on two things: the destination address of packets it overhears, and, when possible, the source address. When a node receives or transmits a packet, it remains awake for a short period of time (e.g., one and a half packet times) to receive packets. When SP thinks another node is awake (e.g., overhears a packet sent to it), it tries to opportunistically send any packets destined to that node. Those packets have a cost of low.

If a deadline is approaching and a send request has a maximum allowable cost of high, then SP will resort to sending a long preamble. Note that this can then lead to a bunch of other nodes sending low cost packets.

3.2 TDMA

In a TDMA scheme, SP is aware of the slotting and can try to schedule it based on application requirements. Note that basic send requests with low timeouts will almost always fail. If the TDMA has some form of broadcast slots, then broadcast slots are considered high cost for non-broadcast traffic (you can send a unicast then, but it is wasteful). If the TDMA does not have some form of broadcast slot, then sending broadcast traffic is considered high cost as it has to be sent to all \( n \) neighbors.

4 Collection Routing

There are two kinds of traffic: control and data. Control traffic is broadcast. Data is unicast. The data traffic period is DATA_PERIOD, the control traffic period is CONTROL_PERIOD. CONTROL_PERIOD = \( C \times \) DATA_PERIOD, where \( C \) is a small integer constant greater than 1.

To schedule its control traffic, the routing layer calls

```c
ScheduleSend.schedule(BCAST, CONTROL_PERIOD, CONTROL_PERIOD / 2, 0);
```

This tells SP to schedule broadcast packets at an interval of CONTROL_TRAFFIC, but with jitter so that it can be sent up to a half interval early or late.
It also wants to send data traffic. The samples are taken regularly, with a phase offset of \( \text{DATA\_PERIOD} / 2 \) off from the timer (the beginning of its possible firing time range). The routing layer exports the sampling timer up to the component that uses it.

```java
ScheduleSend.schedule(parent_addr, DATA_PERIOD, DATA_PERIOD / 2, 0);
...
SendSP.send(parent_addr, remaining_jitter, msg, HIGH);
```

When the parent changes, the layer calls stop() and reschedules. Forwarding traffic is sent with a call to SendSP:

```java
SendSP.send(parent_addr, DATA_PERIOD, msg, MEDIUM);
```

Note that because of the rates, forwarding traffic is MEDIUM: it will not send a large preamble for forwarding traffic, but will piggyback on data traffic if it needs to.

In a CSMA case, SP aligns the phase of the control and data traffic so both can have a cost of medium. It signals sendTime as early as possible. If it hears another packet transmitted to the desired destination, or a broadcast packet, it tries to send all outgoing packets with cost low. If CSMA prevents it from doing so, then it might have to wait again, or resort to high.

One issue that arises is if several nodes share a parent, we want to amortize the cost of who sends the initial long preamble (which all others can then follow on with short premables) across them. The deadline to send high should have some random element in it (e.g. \((\text{period}/10 - \text{random})\)).

In the TDMA with bcast slot case, SP has to randomize what broadcast slot the control traffic is sent in to reduce the chance of collision.