MLA: MAC Layer Architecture

Octav Chipara

Original slides by Chenyang Lu
Challenges

- **Power management is critical for wireless sensor networks**
  - Limited energy source
  - Lifetime from months to years

- **Gap between protocols and systems**
  - Significant advance in power management protocols
  - Significant challenges to integrate them in real systems
  - Minimum support for power management in OS

- **Need unified architectures for flexible power management!**
Diversity of MAC Protocols

- Conflicting application requirements
  - Energy
  - Latency
  - Throughput
- Radio is a major consumer of energy
- Need different MACs to meet different requirements
Current Solution

- Design a new MAC protocol as a monolithic stack
  - S-MAC
  - BMAC
  - ZMAC
  - XMAC
  - WiseMAC
  - T-MAC
  - SCP
  - Funnel-MAC
  - Crankshaft
  - 802.15.4
  - DRAND
  - ……………
Problem with Current Solution

No separation between power management & core radio functionality
Problem with Current Solution

All features jumbled into one big monolithic implementation
Problem: Monolithic Radio Stack

- Hard to develop new MAC protocols
  - No clear separation of concerns
  - Need intimate knowledge of entire stack
- Hard to maintain multiple MAC stacks as OS evolves
- Protocols not reusable across radio/processor platforms
MLA: MAC Layer Architecture

- Separation of sleep sleeping from radio core
- Components for sleep scheduling protocols
  - Reusable ➔ ease development & maintenance of protocols
  - Platform independent ➔ reduce porting effort
MLA: MAC Layer Architecture

• Components implement common features of MAC protocols
  • Hardware-independent: portable across platforms
  • Hardware-dependent: portable interfaces, platform specific implementations

• Simplifies porting to a new platform
  • Re-implement hardware-dependent components
    • Once per platform
  • Hardware independent components stay the same

• Support diverse MAC protocols
  • CSMA (contention-based), TDMA (scheduling-based), Hybrid

• Comparable efficiency to monolithic implementations
dependent components is inherently platform specific, they provide abstract, platform independent interfaces to features to be composed together in a platform independent high-level component architecture. MLA identifies these techniques — such as periodic channel polling and time synchronization — and encapsulates them inside a set of optimized, reusable components. These components are aimed at formulating new MAC protocols that send data with very short preambles. Z-MAC employs a TDMA-style slot allocation for all nodes, but allows nodes to send data with very short preambles. Funneling MAC alleviates contention with channel polling's high throughput. Finally, TDMA in regions close to sink nodes, where nodes experience high contention.

The high-level, artifact to Section 3.3.3.

We define two types of components for use in MLA. Though MLA's architectural design is not inherently tied to the MAC layer. We defer a more detailed discussion of this decision does not generally affect MLA's design, with the exception of radios (e.g., the CC1000 radio used on Mica2 motes). We choose to focus on packet radios, since packet radios (e.g., the CC2420 radio used on TelosB and MicaZ motes). We define two types of components for use in MLA.

We achieve both of these goals by exposing all MAC interfaces (described later in Section 3.4) and produces corresponding interfaces provided by the radio, and exposing a set of (partially) more general interfaces. The following sections elaborate on these various interfaces, as well as provide detailed descriptions of the components that provide them. We discuss how we have met these challenges and present the MLA architecture.
B-MAC: An Example Protocol

Sender:
- Preamble
- Data

Receiver:
- Sleep
- Data
- Sleep

Check the Channel
Check the Channel and receive
Check the Channel
B-MAC

Sender:
- Preamble
- Data
- Sleep

Receiver:
- Sleep
- Data
- Check the Channel
- Check the Channel and receive
- Check the Channel

Receiver performs periodic CCA check
B-MAC

**Sender:**
- Sends preambles equal to CCA check period
- Sleep

**Receiver:**
- Performs periodic CCA check
- Sleep
- Check the Channel
- Check the Channel and receive
- Check the Channel

**Data**
B-MAC

Sender sends preambles equal to CCA check period

Receiver performs periodic CCA check

Sender: Preamble
Receiver: Data

Sleep

Check the Channel
Check the Channel and receive
Check the Channel
B-MAC: What Does It Need?

- Method of turning the radio on and off
- Method of checking the channel for radio activity (CCA)
- Periodic Timer to listen for radio activity
- A way of sending / receiving preambles
- A way of sending / receiving data
Breakdown of B-MAC

- **What does it need?**
  - Method of turning the radio on and off
  - Method of checking the channel for radio activity (CCA)
Breakdown of B-MAC

- **What does it need?**
  - Method of turning the radio on and off
  - Method of checking the channel for radio activity (CCA)
  - Periodic Timer to listen for radio activity

---

[Diagram: Channel Poller, Timers, Radio Core]
Breakdown of B-MAC

- **What does it need?**
  - Method of turning the radio on and off
  - Method of checking the channel for radio activity (CCA)
  - Periodic Timer to listen for radio activity
  - A way of sending preambles and data
Breakdown of B-MAC

**What does it need?**
- Method of turning the radio on and off
- Method of checking the channel for radio activity (CCA)
- Periodic Timer to listen for radio activity
- A way of sending preambles and data
- A way of receiving data and filtering out preambles
BMAC - details

4.1 B-MAC

RadioPowerControl

SendPreamble

AsyncSend

LowPowerListening

4.2 X-MAC

reader back to Section 3.4.5 for details on this component.

4.3 SCP

specification [6]; the first is taken from X-MAC, and the sec-
ond is unique to the monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
we implement both behaviors for consistency with the exist-
ning monolithic stack. Nevertheless,
Low-latency IO

- **Low-latency is essential for TDMA and contention-based protocols**
  - expose the async receive and sends from the radio layer
  - provides hooks for low-latency operation
  - the usual warnings about asynchronous context still apply
## Component Library

### CSMA Protocols

<table>
<thead>
<tr>
<th>Hardware Independent</th>
<th>Hardware Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble Sender</td>
<td>Radio Core</td>
</tr>
<tr>
<td>LPL Listener</td>
<td>Local Time</td>
</tr>
<tr>
<td>Channel Poller</td>
<td>Alarm</td>
</tr>
<tr>
<td>Slot Handlers (TDMA/CSMA)</td>
<td></td>
</tr>
<tr>
<td>Time Synchronization</td>
<td></td>
</tr>
<tr>
<td>Low Level Dispatcher</td>
<td></td>
</tr>
<tr>
<td>Asynchronous I/O Adapter</td>
<td></td>
</tr>
</tbody>
</table>
## Component Library

### TDMA Protocols

<table>
<thead>
<tr>
<th>Hardware Independent</th>
<th>Hardware Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble Sender</td>
<td>Radio Core</td>
</tr>
<tr>
<td>LPL Listener</td>
<td>Local Time</td>
</tr>
<tr>
<td>Channel Poller</td>
<td>Alarm</td>
</tr>
<tr>
<td>Slot Handlers (TDMA/CSMA)</td>
<td></td>
</tr>
<tr>
<td>Time Synchronization</td>
<td></td>
</tr>
<tr>
<td>Low Level Dispatcher</td>
<td></td>
</tr>
<tr>
<td>Asynchronous I/O Adapter</td>
<td></td>
</tr>
</tbody>
</table>
Component Library

Hybrid Protocols

<table>
<thead>
<tr>
<th>Hardware Independent</th>
<th>Hardware Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble Sender</td>
<td>Radio Core</td>
</tr>
<tr>
<td>LPL Listener</td>
<td>Local Time</td>
</tr>
<tr>
<td>Channel Poller</td>
<td>Alarm</td>
</tr>
<tr>
<td>Slot Handlers (TDMA/CSMA)</td>
<td></td>
</tr>
<tr>
<td>Time Synchronization</td>
<td></td>
</tr>
<tr>
<td>Low Level Dispatcher</td>
<td></td>
</tr>
<tr>
<td>Asynchronous I/O Adapter</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

- All evaluations performed on TelosB motes in TinyOS 2.0.1
- Implemented 5 MAC protocols
  - B-MAC, X-MAC, SCP-Wustl, Pure TDMA, SS-TDMA
- Measure
  - Reusability of components among protocols
  - Memory footprint compared to monolithic implementations
  - Throughput
  - Latency
  - Energy Consumption
## Reusability of Components

<table>
<thead>
<tr>
<th>Component</th>
<th>B-MAC</th>
<th>X-MAC</th>
<th>SCP-Wustl</th>
<th>Pure-TDMA</th>
<th>SS-TDMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Poller</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPL Listener</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Preamble Sender</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Time Synchronization</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>TDMA Slot Handler</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CSMA Slot Handler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Low Level Dispatcher</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Async I/O Adapter</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Alarm</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Local Time</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Radio Core</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Other Components</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Reused Components</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Memory Footprint (TelosB)

ROM Overhead

RAM Overhead

Monolithic
MLA

Monolithic
MLA
Throughput (BMAC)

Throughput (kbits/s)

Number of sending nodes

B-MAC (MLA)

B-MAC (Monolithic)
Throughput

Throughput (kbits/s) vs Number of sending nodes for X-MAC (MLA) and X-MAC (Monolithic) protocols.
Latency

![Graph showing latency over number of hops for different MAC protocols: B-MAC (MLA), B-MAC (mono., orig. CCA), B-MAC (mono., new CCA), X-MAC (MLA), X-MAC (mono., orig. CCA), X-MAC (mono., new CCA), SCP (MLA). The x-axis represents the number of hops, and the y-axis represents latency in ms. The graph illustrates that SCP (MLA) has lower latency compared to other protocols as the number of hops increases.](image-url)
The figure shows the duty cycle for various MAC protocols and implementations. The X-MAC implementation outperformed the MLA implementation in terms of duty cycle. The difference between the two stacks becomes insignificant. Specifically, the portable CCA routine allows developers to achieve optimal energy efficiency on specific sensor platforms, while still permitting platform-independent tuning on other platforms. The figure also highlights the importance of efficient CCA routines and their impact on the duty cycle.
MLA: Summary

• Component-based, low-power MAC architecture
  • Increases flexibility
  • Simplifies development
  • Reduces porting effort

• Provides evidence contrary to the existing philosophy that radio stacks must be monolithic to be efficient
  • Bridge the gap between algorithms/protocols and systems.

• Code: tinyos-2.x-contrib/wustl/upma
Solve the Real Problems

- **Hard to develop new MAC protocols?**
  - RI-MAC (SenSys’08) built on top of MLA
  - More built on MLA
- **Hard to maintain multiple MAC stacks as OS evolves?**
  - Upgrading MLA for TinyOS 2.0.1->2.0.2->2.1 took several hours
  - Multiple MAC protocols survived upgrade without any change!
- **Protocols not reusable across radio/processor platforms?**
  - Supports both Telos and MicaZ
- **TinyOS 2.1 version available from TinyOS “contrib” CVS**
References