TinyOS Installation

TOS Programming Environment
TinyOS Installation on MS-Windows
Tool Description
Other Topics
Conclusion – TOS Installed!
TinyOS Development Tools

- Host Application Tools – Java
- USB Device Programmer
- AVR GNU Compiler, Assembler, & Linker
- Nesc Pre-Compiler
- (Configurations
- TinyOS Components (Modules &
- CYGWIN Unix Shell Environment
- TinyOS Development Tools

- Run & Debug Application
- Load Application Executable onto Platform
- Build Application Executable for a Specific
  Platform
- Design & Create Application Source Files

TinyOS Development Flow
What We Need

- MB520 Programmer, 2 ea. MIGA2, Power & Batteries
- Mote Hardware
- 50 minutes – Time
- Disk Space – 1GB+Bytes
- Acrobat PDF Viewer
- WINZIP
- MS Windows (XP, 2K, 98, NT)
- Crossbow TOS Training CD-ROM

Installation Flow

© Problems - Notify a Crossbow Trainer

- ITTOS already installed - just do UPDATES
- Test all Tools
- Update Utilities - nisp
- Install NESC - 1.1beta
- Update TinyOS - v1.1.4july03
- Update Compiler Tools – avr-gcc v3.3
- Install TinyOS 1.0
Run SETUP.EXE
Run INSTALL
Run IMMUNISHM.EXE
Go to CD/DVD/TOYOS

TIVO Setup

Login with ADMINISTRATOR PRIVILEGES

Disk

105 Installation from Xbow
Run Isscript.msi

Run Install Services

cdrom:/Tnysos-Install/Install/Install.exe
D:\thumbs\install\dkinstallation.exe
It is on the CD-ROM

Please enter information in the field below.

InstallShield Wizard

Specify install\exe path

..."Preparing to Install" takes a little while...

Preparing to Install...

TOS 1.0 Baseline Install
TOS BaseLine Installed

- Utilities
- Java Developer Kit
- Graphviz
- TOS 1.0 BaseLine
- AVGcc
- Cygwin - UNIX Shell

TOS Install Setup Status
Cygwin Install from Local

Cygwin Update
Select Packages

Cygwin Update

Select Local Package Directory

Cygwin Local Package Dir
TynQOS Install Complete

Cygwin Updating
Avrcc Update

WinAver – avrcc Update
Setting the Cygwin Environment

Cygwin Profile Update

- Go to Cygwin
- Copy profile
- Rename profile
- Or profile.d:xbow if TFTP installed on D:
- Profile xbow from CDROM
- Rename profile:xbow to profile

AvrGcc Update

Installing Files
TivOS 1.1 Beta Update

- Requires WINZIP to install
- Rename c:\tivos-1.1 to tivos-1.x.old

Click on CYGWIN Icon

* All commands are CASE sensitive
  - $ exit
  - $ cd
  - $ err-log
  - $ apt-get
  - $ apt-get
  - $ apt-get
  - $ apt-get

Cygwin Test
CLOSE WINZIP WHEN FINISHED

TINYOS 14JULY03 UPDATE

UNZIP TINYOS-14JULY03.ZIP

Press Extract
Specify c:\ (or d:\) as "Extract to:"
Select EXTRACT Panel
Programmer Extra Flags: `-dirt` 

```
# The default here is to use `make` instead of `cd` and `make`.
```

- Use Programmer Notification (conf on desktop) 

Group ID: Make sure you are logged in as root.

Make Resolution Update

```
exet $ 
make $ 
cd $ 
plink $ 
cdapps $ 
make install 
cd nesc-1. beta 
cd cistньs $ 
Open CYGWIN
```

Nesc-1. Beta Update
- Displays ATmega CPU configuration
- Version 2003-09-01
- Plug-in Power
- Plug MICA2 in to MIB500
- Plug MIB500 in to LPT Port

isp Test

make $ •
cd $ •
make hostinstall $ •
cd $ •
make $ •
cd $ •
make config $ •
cd $ •
cd $ •
cd $ •
Open CYGWIN

Programmer Build
isp Universal In System
TinysOS Document Generator

Close the SerialForwarder Window

$ java net.tinys.fst.SerialForwarder

$ cd java

$ cd tools

$ Cygwin
Debug Application using TEDS and/or JTAG
  - Or simulate on PC 'make pc'
  - cable

Install executable on Target via Programming
  - AVRT-GNU links into an executable for specific target
  - AVRT-EC GNU compiler compiles/produces modules for
  - nesc locates & pre-compiles all required components

Build Application via 'make'

Create Application 'wire' interfaces

TOS Development Flow

Novity a Crossbow TOS Trainer

Monitor Tivos-1.x, Cygnus, AVRcc, uisp, Java are running

Problem

Installed

Tivos 1.1 Beta

Install Complete
Use ModuleID = 10 (decimal)
make install [module] misc
ModuleID set during install or exe into target
Specific Module within a Group
Module ID
Each your lineno/apps/makefiles NOW using ID # on Name Tag
- GroupID is SET in makefiles
- Messages are local to a Group
- Collection of Modules form a Group

TARGET IDS

DETAILS WILL BE COVERED IN A LATER SESSION
- Debugging options
- Platform options
- System specific options
- make/variables in /apps directory
- Application specific options & SensorBoard
- make in application's directory
- Build is controlled by 2 script files

make
<table>
<thead>
<tr>
<th>TinyOS Directory Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
</tr>
<tr>
<td>Tiny OS modules and Programs</td>
</tr>
<tr>
<td>Development utilities and Tools</td>
</tr>
<tr>
<td>Nesc pre-compiler source Nesc</td>
</tr>
<tr>
<td>Documentation &amp; Tutorials Doc</td>
</tr>
<tr>
<td>User contributions Contrib</td>
</tr>
<tr>
<td>and Test Programs Apps</td>
</tr>
</tbody>
</table>

TinyOS make options

- Docs - documentation
- Primary into target
- Install [module] <platform> - build and download
- Clean - erase all build files, clean up
- All - all platforms
- Platforms: micaz, nisea2dot, legacy renez, mica2
MICAZ Platform Topics

<table>
<thead>
<tr>
<th>Special type definitions</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEPROM, UART</td>
<td>System</td>
</tr>
<tr>
<td>Mote System drivers</td>
<td></td>
</tr>
<tr>
<td>Sensor Board drivers</td>
<td>Sensorsboards</td>
</tr>
<tr>
<td>Mote Platform specific drivers</td>
<td>Platform</td>
</tr>
<tr>
<td>Libraries – incl. TinyDB, Route</td>
<td>Lib</td>
</tr>
<tr>
<td>TOS component interfaces</td>
<td>Interfaces</td>
</tr>
</tbody>
</table>

TOS Subdirectory
$ fuse428 & $ fuse103
$ fuse428 & $ fuse204

Alias Scripts

udsp -dp provoke=depage --wx fuse-e=0x7ff
udsp -dp provoke=depage --wx fuse-h=0x649

To Set Fuses

TAG OFF (ADC Channels 4 thru 7 work)
TAG OFF (ADC Channels 4 thru 7 work)

ATMega 128 Native (not 103 compatible)

Standard ATMega-128 Fuse

[BROWN-OUT Detect [OFF]
[ON-COMP Mode [OFF]
[TAG [OFF

Shipped with this fuse ENABLED

-Disable Legacy (MICA), note older MCAs were

AT103 Compatibility Mode [OFF]

ATMega Fuses
Using CVS

Retrieving Current TOS Versions

To set fuses for JTAG debug

- Low power SLEEP mode draws 2mA!
- JTAG
- ADC channels 4, 5, 6, 7 reserved for
- JTAG & ON-Chip Debug Enabled
- JTAG ATMega-128 Fuse
Conclusion

- Tool Description
- Tool Installation for MS-Windows
- Tool Sequence
  - Review
  - Installation Complete
Introduction to TinyOS and nesC Programming

- TinyOS Kernel Design and Implementation
- nesC Code Lab
- TinyOS Packet Networking and PC Base Station
- nesC Software Concepts and Basic Syntax

DSP Labs, Livermore, CA, USA
maurer@dsplabs.com
Bill Maurer
- Energy Hog
  - Slow
  - More than an order of magnitude too heavy
- PDAs, Cell Phones, Embedded P.C.'s
- Execution similar to Desktop Systems
  - VxWorks, QNX, Wind, PalmOS
- Microkernel Architecture
  - Can't use existing RTOS's

TinyOS Design Options

- Smaller, cheaper, lower power
- Keep scaling down
- Support Technical and Advances
- Power, sensing, communication
- Support Micro Hardware
- Bursts of events and operations
- Asleep but remain vigilant to stimuli
- Support Networked Embedded Systems

TinyOS Design Goals
Implies only need a single stack.
Run to completion WRT other Tasks.
E.g., Computing an average on an array.
Tasks - Larger amount of processing.
Not Time Critical.

Tasks -
Can interrupt/longer running tasks.
E.g., Timer, ADC interrupts.
Small amount of processing.

Events -

TinyOS Kernel: 2 Level Scheduling Structure.

TinyOS Kernel Design

Extremely Simple.
Extremely Efficient.

Conclusion: Need a Multi-Threaded Engine.
Add: Managing Application Data Processing.
Manage large # of Outstanding Events.
Manage large # of Concurrent Data Flows.
Data Driven Execution.
Similar to Building Networking Interfaces.

TinyOS Design Conclusion.
TivOS Kernel Design (Events and Tasks)
neSC Programming Language Supports the
Details of neSC Forthcoming
Application is created in the neSC Language

TivOS Applications Under The Hood

(neSC Compiler C Code Output)
Robustness Improved
- Development Made Easier
- Function Inlining
- Data Race Conditions
- nesc Compiler Analysis
- No Heap
- No Function Pointers
- No Dynamic Memory (no malloc)
- Everything is Static

**tnivos/nesc Application Notes**

---

### Code and Data Size of the Tnivos Kernel

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost(nexes) Time(s)</th>
<th>Normalized to Byte Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call a Command</td>
<td>2.5 2</td>
<td>0.10</td>
</tr>
<tr>
<td>Schedule a Task</td>
<td>2.5 1.5</td>
<td>0.10</td>
</tr>
<tr>
<td>Context Switch</td>
<td>2.5 1.75</td>
<td>0.15</td>
</tr>
<tr>
<td>Hardware Interrupt (any)</td>
<td>2.5 1</td>
<td>0.10</td>
</tr>
</tbody>
</table>

---

### Code Size

<table>
<thead>
<tr>
<th>Code Size(bytes)</th>
<th>Code Size(bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

---

### Data Size

<table>
<thead>
<tr>
<th>Data Size(bytes)</th>
<th>Data Size(bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
</tr>
</tbody>
</table>
Events, Commands
TinyOS Concepts Embodied by nesC - Tasks

- Can't Signal
- Function call to another Component
- Command
  - Receiver gets/accepts an Event
  - Initiator gives a Signal
  - External Interrupts
  - Time Critical Events
    - Critical
    - Background computation, non-time
    - Tasks

Events, Commands

- Stack - Grows Down in the Free Space
- Free Space - Fixed (No Dynamic Memory)
- In the 4K SRAM
  - Variables
  - In the 128K Program Flash
  - Data - Program Constants
  - In the 128K Program Flash
  - Text/Code - Executable Code

Application Memory Map
Component used to form an executable

BlinkMac Application - A top level configuration SW

Concepts of SW Components

- Configuration (xxx,nc)
  - Module (xxx,M,nc)
    - Software Components
      - Tell outside world
        - Special functionality to outside world
      - Tell outside world
        - Commands can be called
      - When events need handling
      - (No code - xxx,nc)
  - Linking w/ interface
  - When top level app, drop C from
  - Optional Module
    - When top level app, drop C from
  - Linking w/ interface
  - Interface implementation
  - C:
    - x:
      - Interf:ce:ne
  - Interface:ne
  - Main:ne

Note: The interface includes a signal from a Bluetooth command. Use code to interface with a Bluetooth interface.

1. When interface and main are application.
Our First Application

The Blink Configuration

- At the top level, it is implied

Our First Application

- Why isn't this BlinkCone?  

The Blink Configuration

- A single top-level configuration block
- Each configuration block is
- Every providing a set of components

A configuration is a

A configuration is a

Another configuration

components in the
configuration that

configuration

components, together

building an application

configuration, where

component

configuration

configuration

configuration

configuration

configuration

configuration

configuration

configuration

configuration
10s Interfaces/Interface
Result:

- Definition of a command or event prefixed with "command"
- Event Signal prefixed with "signal"
- Command Calls prefixed with "call"
- Definition and Invocation Name is "I"
- For a Command Or Event For Interface "I"

nesc Command and Event Syntax
Distribution is Transparent

Some are Thin Wrappers Around Hardware

Most Components Are Software Implementations

- Clock => BinkMacClock
- Clock => BinkMacClock
- Clock => BinkMacClock
- BinkMacClock => Clock
- BinkMacClock => Clock

Equivalent Wiring Syntax

TinyOS/nesc General Notes

```
{ return Call(Core::Singleton::TOS_TCP, TCP_Send());
  if (result)
    return SUCCESS;
}

BinkMacClock::Tick()
```

```
module BinkMac
  uses
    Interface Clock
    Interface Singleton
  provides
    module BinkMac
```

A module is a

BinkMac Implemenation
Resolving Components / Interfaces

Building Applications: Blink

- E.3, TAG, Debug
  - Other
- How to program (e.g., which usip and usip flags)
- Install Specific instructions
- Target hardware and sensor board information
- Platform Specific build instructions
- Group ID, Code Optimization
- System level build instructions
  - Makefile in apps directory, global to all apps
  - Debug options
  - Radio frequencies
- COMPONENT=Appname (Blink)
- Makefile – local to Blink (application specific build instructions
  - BlinkMac
  - BlinkMac
  -
Exercise 2

Exercise 1

Change Blink Rate of Blink to 2 Blinks per Second

Look for IPS

Edit Blinky.mc

- Change to TOS, 12PS, TOS, 22PS

Make Install 1 miscal

File is at doc/nesdoc/miscal/index.html
nesc Terminology Review

documentation

In this context, the nesc coding and naming conventions is described in
the nesc code. These are several other naming conventions used
corresponding module, which you would name an application's target
module and associated top-level configuration and an interface
together. When this is the case, the corresponding module used in

Sometimes (as is the case with Blink and Blink++) you will have a configuration and a

modules that can be used in a range of applications.

The reason for the distribution between modules and configuration is to allow a system designer to quickly

developer can provide a new set of library modules, none of which are included. Libraries, another developer can provide a new set of

configuration. For example, a designer could provide a configuration that simply wires together one or more

components, a module called Blink++, and a configuration, called

Blink++ consists of two components: a module called Blink++ and a configuration, called

apps/Blink Application Notes
Implies only need a single stack
- Run to completion wrt other tasks
- Larger amount of processing tasks
- Can interrupt longer running tasks
- Events - Small amount of processing

TinyOS Kernel: 2 Level Scheduling Structure

Review of the TinyOS Kernel

Fan in & Fan Out Components Wiring
- Parameterized Interfaces
- Split Phase Operations

What's Left?
Snippets from apps/SheetsTask/SenseTask.mc

So let's look at an example Task

- Back to Sleep
- Event - Clock Fire, toggle led.
- Sleeps
- There were no Tasks in Blink
- Blink Again
(between Components) Split Phase Operations (aka Handshaking)

...{task 1 } {task 2} {post task3} ....

Break it up into a cascade of Tasks

- Executing
  - Long Latency Tasks prevenient other Tasks from
  - Short Latency Tasks are Nice
  - Tasks do not Preempt Other Tasks

Tasks Characteristics
Uses ADC

- How does TimeQOS handle this?
- One ADC component, but 8 ADC channels
- E.G. ADC (Analog-Digital Converters)

Parameterized Interfaces

Split Phase Defined

C1 must handle this event
C2 Task finishes event at completion
Else C2.COMMAND Posts Task and returns OKAY immediately
- C1 Error Handling
- C2 Error Handling
- Call C2.COMMAND (dual command)
- C1 initializes the Split Phase Operation
- Provides the Interface
- Lower Component C2
- Uses the Interface
- Upper Component C1

2 Components (C1,C2) Wired Together
The Timer: los/system/Timer, nc
•
The LED Display: los/lb/IntToLeds, nc
•
A Counter: los/lb/Counter, nc
•
Wires Up
•
CountToLeds, nc – Display Count on LEDs
•

Configuration Only Application

Sniffer Pair Lab

- Remote
- Front
- Control Switch
- Timer[x]
- IntToLeds
- Counter
- CountToLeds

(Module)

Configuration Only Apps (No

(Grok, Inc.) Lab

nesc Code
module LogClockThread (ios of code, ...

	...LogClockThread's methods...

	...LogClockThread's classes...

...The Time and SICClock

implementation

_configuration.TimeM

implementation

Configuration TimeM

trigger The Counter

Triggered when SICClock is initialized.

implementation

_configuration.Counter

// This application is built by putting the TimeM, InitialTime

configured from Existing Components

Counter
What is strange about

```c
return SUCCESS;
post(outbound);
else if (calls < 1) {
    return SUCCESS;
} else if (calls >= 2) {
    // Call 2nd callback;
    call 2nd callback;
    // Call 3rd callback;
    call 3rd callback;
} else {
    // Call 1st callback;
    call 1st callback;
} command result 1: inbound=0, callback(1)
```
- Make install 10 mica2
- Plug-in MICA2 unit #1 to Programmer
  - DEFAULT LOCAL GROUP: = use # on your Name Tag
  - Makelius
- Check GROUP
  - CLASSES: [collect DEFAULT FREQ=CHIRK 433.002 MHz]
  - Makelius
- Set the Radio Frequency
- Go to apps/RmtToleds

RmtToleds
Build Receiver Side

- Verify LEDs show count
- Make install 10 mica2
- Plug-in MICA2 unit #1 to Programmer
  - DEFAULT LOCAL GROUP: = use # on your Name Tag
  - Makelius
- Check GROUP
  - CLASSES: [collect DEFAULT FREQ=CHIRK 433.002 MHz]
  - Makelius
- Set the Radio Frequency
  - Go to ../apps/RmtToledsAndRm

RmtToledsAndRm
Build Transmitter Side
Public VS. Secure
Addresses, AM#, GroupID and Payload
TinyOS Packet Structure

AM Handler using Parameterized Interface
- AM Handler using Parameterized Interface
- Active Messages Similar to a Port on a Server
- Inter-Process Communication Mechanism
- Appropriate Component in the Application
- System/AM standard message delivered to
  - C.G. Route Update message vs Data Payloads
  - Establishes a Connection between Sender and a
    Component in an Application
    "Active Messages" (AM) Messaging System
display Raw Tynios Packet Data on PC

Terminal via UART

Application sends/receives Tynios packets via UART

Hardware Connections

- Oscilloscope
- MIB
- MIB to COM Port
- Mote to MIB

AM Routine

GroupID and ADDRESS Filter
Note the -mtca switch sets baud rate

- Java net.tools.Interframe -mtca
- Java net.tools.Interframe -p
- "Java net.tools.Interframe -p" will list the available ports

- Java net.tools.Interframe -mtca
- List PC Com Ports
- cd tools/java

The Java ListenRaw Application

- make install 1.0 micas
- Go to ../apps/Oscilloscope
- Plug-In Mica2 #1 into Programmer
- Oscilloscope

Install TivoOS Application
End of Morning Session

---

Time Programming Part II

Radio Basics

Hardware Overview

Coming up this afternoon

---

So we can express the data packet as follows:

Now what are you seeing?
Outline

Lab
- Modified Oscilloscope for Temperature
- Oscilloscope
- Serial Forwarder
- Generic Base App

DSP Labs, Livermore, CA, USA
maurer@dsplabs.com
Bill Maurer

nesc Programming II
Basestation Mode runs CernetBasic
- Mode Network Interface via Mode Basestation
- Client Interface via Sockets
- Java Server Application Running on PC
  Implemented using Serial Forwarder
  TCP/IP
  MOTES TinyOS Packets to Client Applications via
  "Active Messages" Encapsulated in

"Active Messages" Server
Building a Server Application

Oscilloscope

Generic Base Layer

More Base Station

Issues

- No Packet Framing on UART side
- Payload collisions between TX and RX packets cause lookup

- Packet Framing on UART
- Robust
- UART Baudrate

- loss of information

- GenericBase: nc
- Radio-UART Bridge
- apps/GenericBase: nc
- apps/XGenericBase: nc

- No Packets on XGenericBase

- Packet Framing on UART
- Robust
- UART Baudrate

- loss of information

TCP/IP

- GUI
- Serial Forward
- Actieve Messages Server and Receives broadcast packets
- GenericBase
- Base Station Mode
- TI-Radio Packets
- Broadcast Oscopemessage in
- Samples Photosensor
- More Oscilloscope Application on

- net/uros, oscilloscope.Oscilloscope
- net/uros, st.SerialForward
- Client GUI on PC
- Connect MIB to PC Serial Port
- Make install mica2
- Go to /apps/GenericBase
- Plug-in Mica2 Mote#2 in to Programmer

How to Do It - Generic Base
Mote Base Station

- Make install 1.0 mica2
- Go to /apps/OscilloscopePerf
- Plug-in Mica2 Mote#1 in to Programmer

How to Do It - Mote OscilloscopePerf
Mote Network/Active Messages Server
Click the reset button

Select the "scrolling" check box

If you don't see a status (green lines)

- javanet.tcpip.yos.oscilloscope.oscilloscope

Go to ".tools/javanet/tcpip/yos/oscilloscope

---

How to Do it—Oscilloscope Client

More Network/Active Messaging Server

---

"debug"

- javanet.tcpip.yos.st.serial-forwarder-baud 57600 -

Go to ".tools/javanet/tcpip/yos

---

How to Do it—SerialForwarder

More Network/Active Messaging Server
Sake

End of First Day

FINISHED

{ 
oscilloscope, parameter = comm, sendmsg [AM, OSCOPEMSG]!

comm, recvmsg [AM, RECEIVEDMSG]!
oscilloscope, requestmeasurement
oscilloscope, command, oscilloscope, comm;

Temp

oscilloscope, sendcontrot, oscilloscope, com
oscilloscope, ident = read,
oscilloscope, clock = clock;

Main, stdio, comm

generatecomm as comm!

components, main, oscilloscope, clock, ident, Temp

} 

- implementation
- 

Modify Configuration File

Backup apps/oscilloscope

Sense the Temperature instead of Photosensor

A NEW Oscilloscope
MultiHop Networking Session

Overview:

- Characteristics of wireless mesh networks
- TinyOS Status of MultiHop Protocols
Ad-Hoc Routing
(Self configuring)

- Links are not reliable over the long term
- Links change dynamically
- Requires networking topology that also dynamically changes.
- Low energy requirements limit types of protocols. Powered networks can afford to expend a lot more energy to manage links.
- Broadcasting is energy and time inefficient
- Protocols where the motes dynamically determine the best parent are attractive.

Mote Msgs Constrain Sleep Time

Computation of required sleep time to achieve duty cycle given number of TOS msgs to rcv/xmit while awake

<table>
<thead>
<tr>
<th>Specifications</th>
<th>value</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg Size</td>
<td>40 bytes</td>
<td></td>
</tr>
<tr>
<td>Msg Preamble</td>
<td>16 bytes</td>
<td></td>
</tr>
<tr>
<td>Baud Rate</td>
<td>38400 baud</td>
<td></td>
</tr>
<tr>
<td>Duty Cycle</td>
<td>0.5 %</td>
<td></td>
</tr>
<tr>
<td># of msgs to rcv/re-xmit during</td>
<td>5 msg</td>
<td></td>
</tr>
<tr>
<td>wake time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Computed Values                 |        |       |
| Time to xmit/rcv 1 msg          | 11.7 msec |
| Time to rcv/xmit all msgs       | 116.7 msec |
| Required sleep time to maintain | 23.22 sec |
| duty cycle                      |         |       |

Motes closer to base station get more messages
Radio Link Behavior #2

- Influence quality of link
- Local null effects, people, ...
- Receive strength over time
- Static links show variability in

Radio Link Behavior #1
Reasonably good results between simulation and real world

Best way to start, prove out code before real implementation

Timos supports PC simulation of networks.

Simulation

- Large probabilistic of packet collision
- Probability ~1.3 seconds with min delay
- 156 bytes = 6.25 sec (no min delay)
- 10 more transmitted once
- 80 bytes to receive when transmitted
- 156 bytes at 19.2k baud. 100 packet, 156 modes
- Asymmetric links
- Clustering
- Shortest (no reception)
- Lone links
- Backbone links

Broadcasting (Flooding)
Power management

Timer: Period update of routing table, message...

Parent

Cycle Detection: Avoid loops. (L.E. Forwarding msg to child instead of)

Parent Selection: Decide on parent to forward message.

Sequence numbers. Estimation of differentiable sequence.

Estimator: Computation of neighbors link quality. EX: # of hops.

Table Management: Evolution/insertion of neighbors into routing table

Components:

Routine Table: List of best neighbors and routing info. Stream

Find best parent to forward messages:

Routine beacon from base station to establish route path back.

Multihop Components

Time/Power management is being added to these protocols (3-4 months)

Effort

TIMVOS will allow users to compare different protocols for reliability

No single multihop stack: application dependent

Complex time management

Complex energy management

Complex routing protocols
Simulated Results

- Quality change as link changes quickly as link quality
- Critical to good estimators
- Packet throughput over 90%

Multihop Throughput

Figure 9: Messages flow chart to illustrate the different components for routing

Multihop Components

From Alice Woo, To be Published
<table>
<thead>
<tr>
<th>Module</th>
<th>Work</th>
<th>Testing</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Management</td>
<td>**</td>
<td>***</td>
<td>In progress</td>
<td>May be released with power management implementation in progress</td>
</tr>
<tr>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td>May be best when work on estimators should be released</td>
</tr>
<tr>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td>PhD Research, a lot of work on estimators</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T'indy/fi</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td>**</td>
<td></td>
<td>Released</td>
<td>Released and stable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T'indy/fi</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Some T'indyOS Multi-hop Stacks

- **UNC**: Not released
- **UCB**: Work on protocol
- **Woo**: Released
- **Alice**: Nielsen
- **Intel**: Released and stable
- **DSDY**: Used in T'indy, GSK
- **CF**: Protocol
Crossbow In-Network Programming

XNP
processed independently of Application Service – all XNP Radio messages are implemented as a private active Message Re-program More to local Flash Memory – Download Application SPEC file from Host Two Phases

How XNP Works

(Routine (forward & reverse path) Compatible with Single and Multi-hop Host Side GUI Control Program Individual or Group More Updates to any TOS Application

What is XNP?
memory and restores ID.
Application fetches More ID from non-volatile
Motes re-program themselves and re-boot
memory
Motes store their current More ID in non-volatile
downloaded ID
Active Motes verify Program ID matches
with Program ID
Host broadcasts Re-Program / Re-Boot command

Step #2 XNP Re-Program

Image
Repeat until all Active Motes have complete
Host transmits lost capsule
Motes request "missing" capsule id
Host queries motes for any "missing/lost"
Active Motes store capsules in FLASH
record/capsule to all Motes.
Host broadcasts spec file (2 TOS packets per spec)
Host broadcasts Start Download message

Step #1 XNP Download
Example Interface to XNP

- Install Application and XNP Bootloader in Motes
- Wire xmpc mc into application
- More and Group IDs, call in INIT to restore

- add xnp::npx::set-ids
- XNP signals done when XNP downloaded ends
- Application responds with GRANT or DENY
- Application must release resources (external FLASH)
- XNP signals a request to start XNP download
- add XNP Event Support functions

How to Use XNP
Installation of XNP Bootloader

Example Component Wiring

```c
{ xtype = Xnp, xnp = xnp;
  xtest = xtest, xtest = xtest;
  test = test, xclock = xclock;
  main, stdcont, xtest = xtest, stdcont;
  components = main, generator, clock, ledsc,
} implementation
```
Demonstration Example

XNP Host User Interface
Must sleep 99% of the time or some equivalent strategy.

Bandwidth crunch (see slide from multipath session). Need to wake-up, send data, forward messages.

Multiple hop becomes hard:

Sleep and sniff more strategies (~3-4 Yr).

can’t multipath for more distance.
multiply better battery life (Axxon, 5yr). Phone vs. collision.

Wake up and transmit only strategies have resulted in:

Not hard for single hop.

Power Management

Radio Power Management

More Sleep

Battery Life Calculation

Overview

Session

Mote Power Management
management strategies:
- CUB: introducing more advance power
- contrib/xbow/apps/XTerrorsouze
- Mica2 Sleep example:
  - add >100mA of current
  - setting to any pin in incorrect state will typically
  - all pins must be in correct state
  - Amega 1/2 will sleep at 1mA.

Mote Sleep

battery capacity (mAh)

Spread sheet in on CD.

sleep most of the time.

year battery life must

To achieve multi-

Conclusion:

0.5% duty cycle

1% duty cycle

Two models:

Battery Life Computation
No implementation in TinyOS yet.

- Either go to sleep or wake up to receive a radio packet.
- Sleepy: wake up Atmega128, sample radio’s RSSI
- Chipcon radio can turn on quickly from sleep ~200usec.

**Radio Power: RSSI Read**

- Penalty for transmitter power.
- Is this scalable for a large network?

- Define preamble up to 1 second in length.
- New addition to TOS in micro platform, see cC1000con.sh.
- 100usec for 10 usec to sample the preamble (10% duty cycle)
- Example: If preamble is ~200usec then more can wake up every
  200usec and still catch the preamble.
- Sleepy: User very long preamble so that sleeping modes can wake
  up and still catch the preamble.
- Radio receiver uses this to balance (zero) receiver.
- Preamble is a series of OX5F (alternate 0.1s) set of bytes.

**Radio Power: Long Preambles**