

Programming Environments for Mobile Ad Hoc Networks

Justin Collins
CS 233A
March 10, 2008

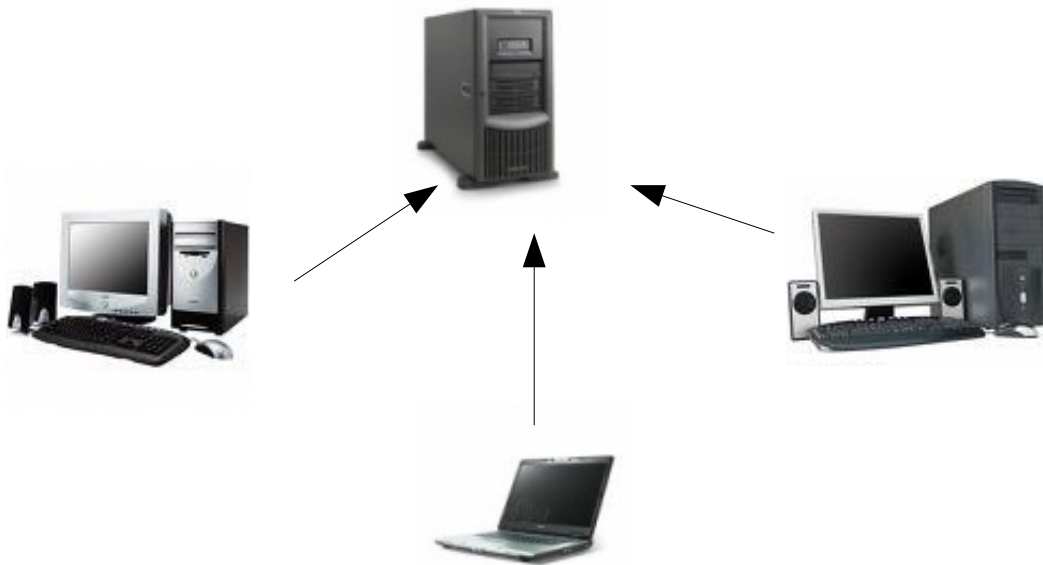
What is a MANET?

- Mobile devices
 - Cell phones, smart phones, PDAs
 - Laptops, sensors, game systems
- Ad hoc
 - No infrastructure
 - Self organizing
- Wireless Networks
 - 802.11b
 - Bluetooth



Traditional Approach

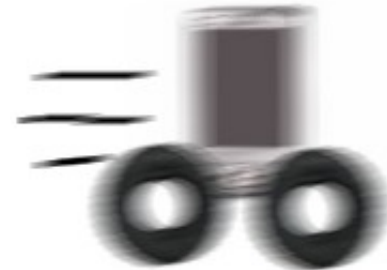
- Client-Server architecture



- Disconnection is an “exceptional” event
- Fairly stable topology
- Centralized server

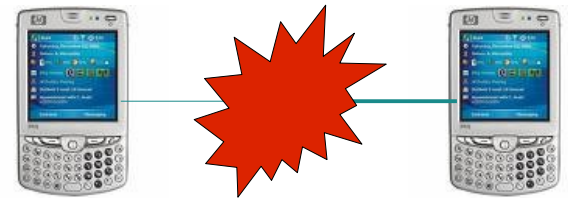
Issues in MANET

- Frequent disconnections
 - Mobility
 - Wireless channel
- Dynamic topology
 - Nodes join and leave often
 - No fixed infrastructure
 - Transient connections
- Small devices
 - Limited power, battery life



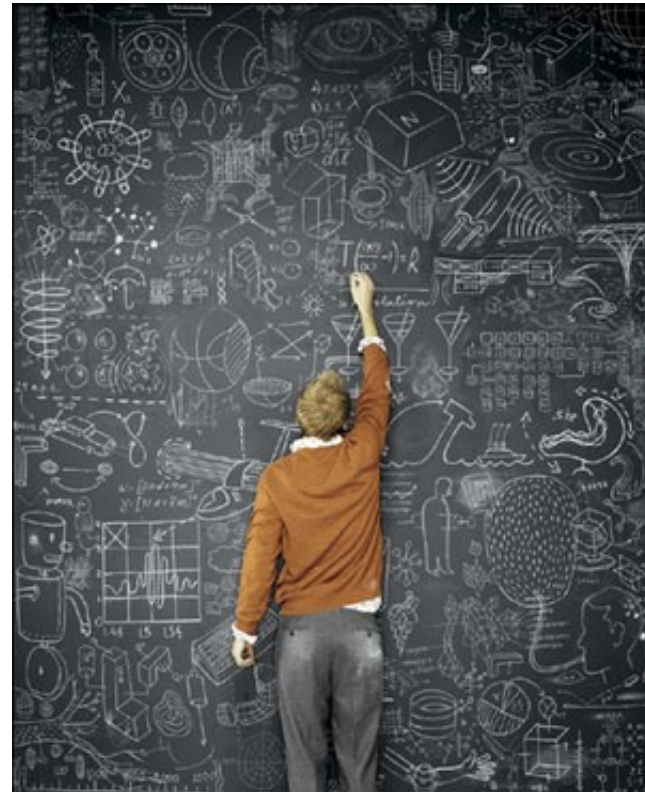
Desired Properties

- Indirect addressing
 - No IP or MAC addresses
- Disconnection handling
- Resource discovery
- One-to-one and group communication
- Decentralized
- Scalable



Ideas

- Tuplespaces
- Peer-to-Peer
- Code migration
- Broadcasting
- Service-oriented
- Remote objects
- Publish/subscribe
- Mobile agents



Project Approaches

- Middleware
 - Sits in between application and network
- Libraries
 - Adds onto existing language
- Languages
 - Programming languages designed for MANET
- Runtimes
 - Tiny language runtimes for mobile devices

SpatialViews

- Java ME language extension
- Code migration
- Iteration over objects
- Limited shared variables
- “Views” based on location and object types



Example – Count Nodes

```
public class Count {  
    public static void main(String[] args)  
    {  
        spatialview v;  
        sumreduction int n=0;  
        visiteach x : v {  
            n++;  
        }  
        System.out.println(n);  
    }  
}
```

Example – Count Nodes

Normal Java class

```
public class Count {  
    public static void main(String[] args) {  
        spatialview v;  
        sumreduction int n=0;  
        visiteach x : v {  
            n++;  
        }  
        System.out.println(n);  
    }  
}
```

Declare a view

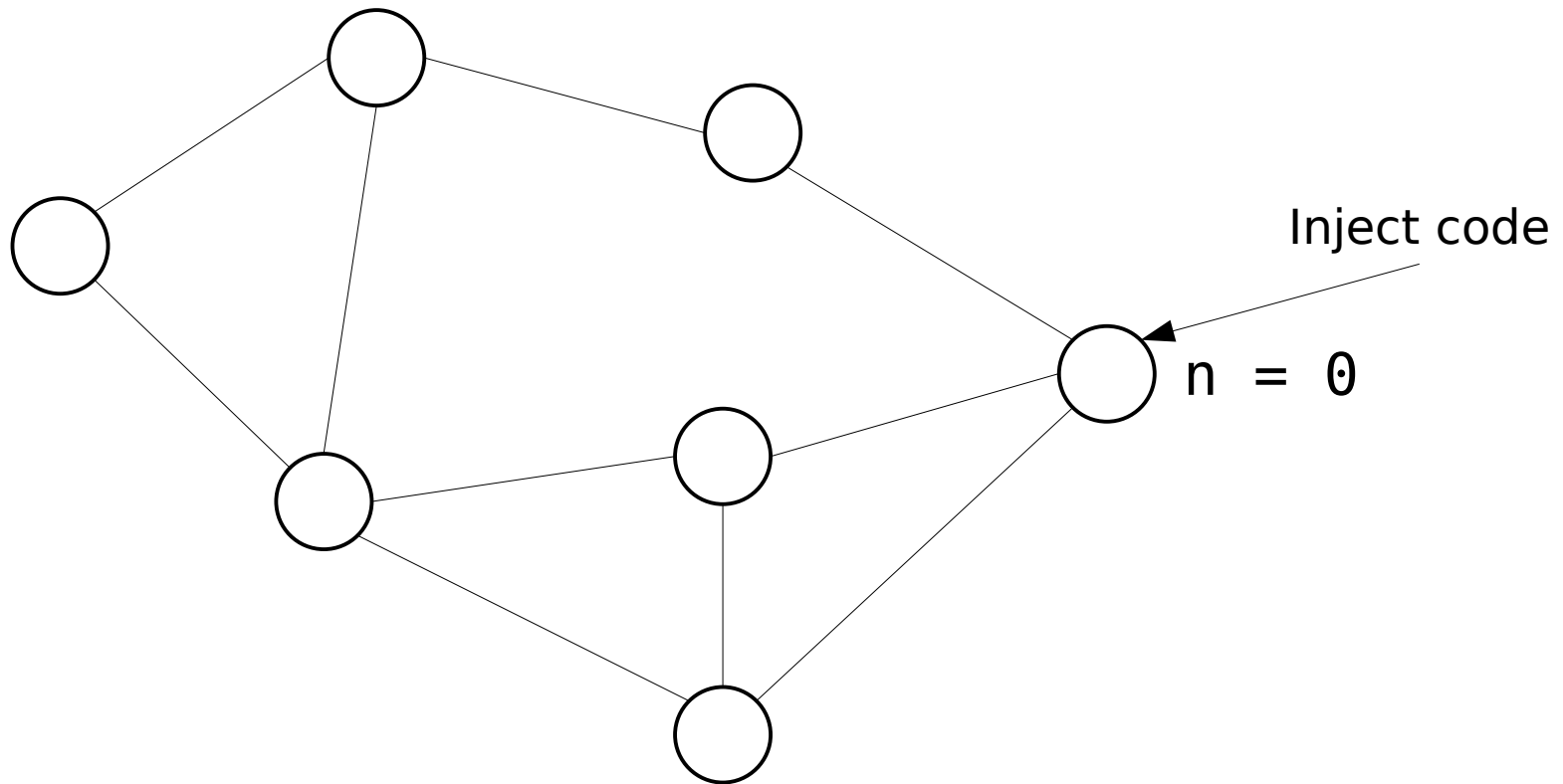
Sum reduction variable

Loop over objects in view

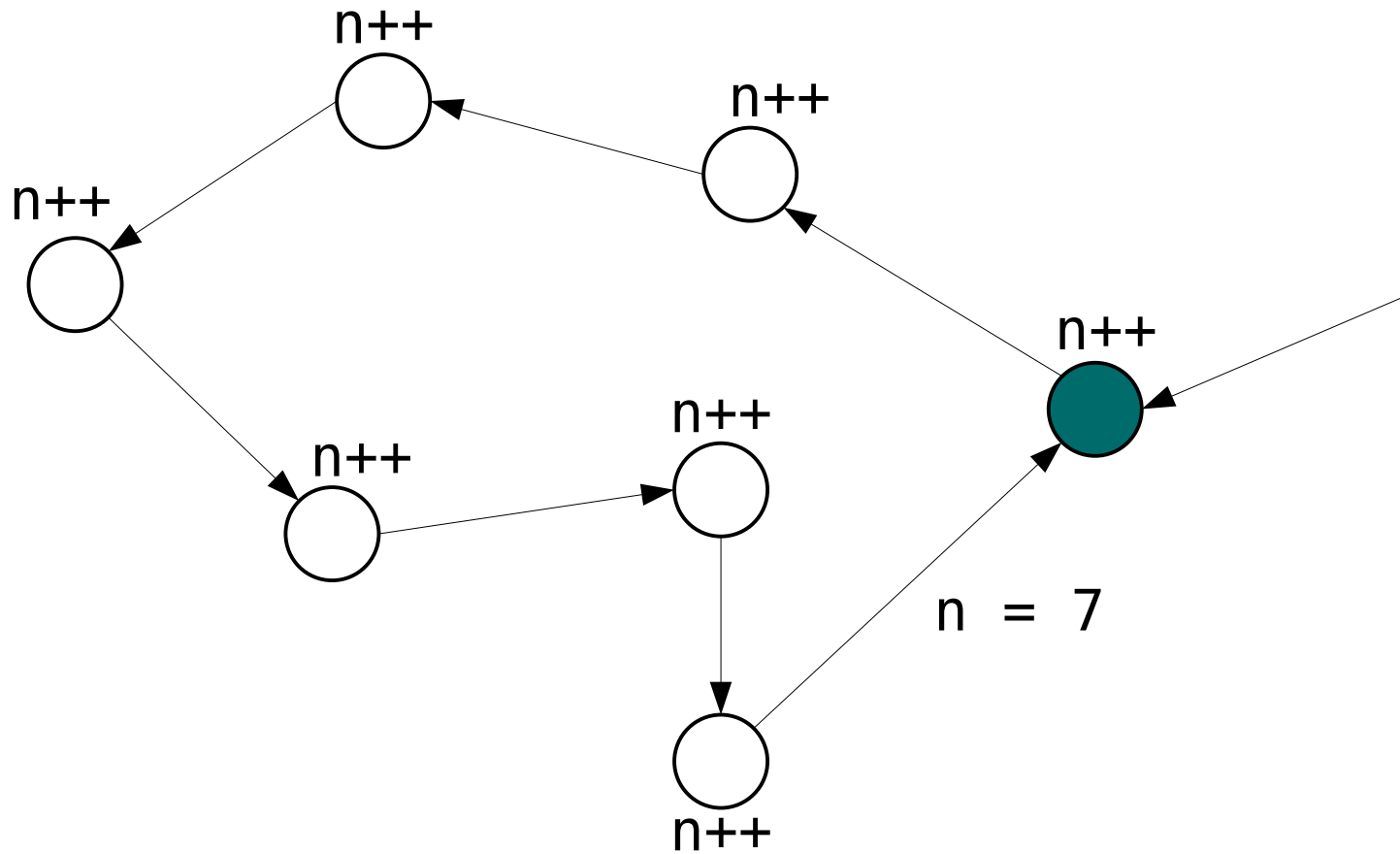
Increment reduction variable

Print out result

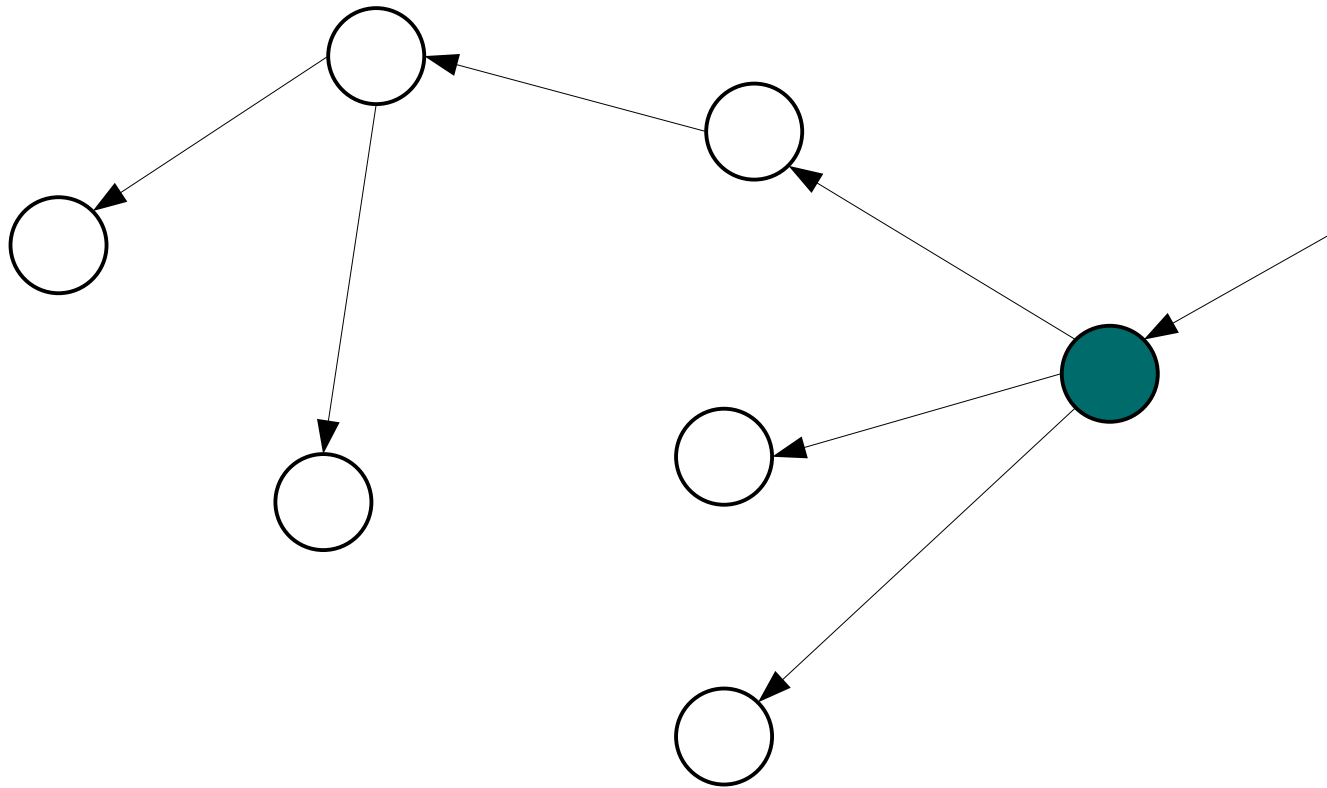
Code Migration – Count Nodes



Code Migration - Linear



Code Migration - Tree



Example – Print Document

```
public class PrintDocument {  
    public static void main(String[] args) {  
        spatialview v = Printer;  
        visiteach p : v {  
            p.print("example.doc");  
        }  
    }  
}
```

Print Document - Problem

```
public class PrintDocument {  
    public static void main(String[] args) {  
        spatialview v = Printer;  
        visiteach p : v {  
            p.print("example.doc");  
        }  
    }  
}
```

Prints at every printer!



Print Document – Print Once

```
public class PrintDocument {  
    public static void main(String[] args) {  
        Container location = new Container();  
        spatialview v = Printer;  
        visiteach p : v {  
            if( location.isEmpty() ) {  
                p.print("example.doc");  
                location.addElement(p.getName());  
            }  
        }  
        System.println("Printed at " + location.getFirstElement());  
    }  
}
```

Print Document - Another Problem

```
public class PrintDocument {  
    public static void main(String[] args) {  
        Container location = new Container();  
        spatialview v = Printer;  
        visiteach p : v {  
            if( location.isEmpty() ) {  
                p.print("example.doc");  
                location.addElement(p.getName());  
            }  
        }  
        System.println("Printed at " + location.getFirstElement());  
    }  
}
```

What if no printers were found?



Print Document – Check Result

```
public class PrintDocument {  
    public static void main(String[] args) {  
        Container location = new Container();  
        spatialview v = Printer;  
        visiteach p : v {  
            if( location.isEmpty() ) {  
                p.print("example.doc");  
                location.addElement(p.getName());  
            }  
        }  
  
        if( location.isEmpty() )  
            System.println("No printers found");  
        else  
            System.println("Printed at " +  
location.getFirstElement());  
    }  
}
```

SpatialViews - Issues

- Disconnections
 - Complete failure if executing code
- Dynamic topology
 - Each step in iteration finds neighbors dynamically
- Small devices
 - Runs on a tiny version of Java
- Indirect addressing
 - Through object types
- Resource discovery
 - Dynamic, but only at iteration time

SpatialViews – Issues Continued

- One-to-one and group communication
 - Group communication is simple
 - One-to-one is very awkward
- Decentralized
 - Mostly, though 'injection point' is like a central source
- Scalable
 - Can be, with good routing

SpatialViews Evaluation

- Advantages

- High level of abstraction
- Simple extensions to Java (many people know it)
- Good for group communication

- Disadvantages

- Breaks if nodes executing code disconnect
- Very limited Java implementation
- One-to-one communication is awkward

AmbientTalk



- Full object oriented language
- Asynchronous event handlers
- Remote method calls
- Futures
- “AmbientReferences”

Example – Print Document

```
deftype Printer;  
def print(document) {  
    when: Printer discovered: { |printer|  
        printer<-print("example.doc");  
        system.println("Printed document");  
    };  
};
```

Example – Print Document

```
Declare Printer type  deftype Printer;  
Define print method  def print(document) {  
Set up event handler      when: Printer discovered: { |printer|  
Remote method call        printer<-print("example.doc");  
                           system.println("Printed  
document");  
                           };  
};
```

Print Document - Problem

```
deftype Printer;  
def print(document) {  
    when: Printer discovered: { |printer|  
        printer<-print("example.doc");  
        system.println("Printed document");  
    };  
};
```



How do we know this is true?

Print Document – Wait for Result

```
import /.at.lang.futures;
enableFutures(true);

deftype Printer;

def print(document) {
    when: Printer discovered: { |printer|
        when: (printer<-print(document)) becomes: {|result|
            system.println("Printed document:" + result);
        }
    };
};
```

Regular Remote References

- Typical remote object handle
 - Unicast
 - Messages buffered until they can be sent
 - Refers to one specific remote object
 - Method calls can return values

For example, the printer in the previous code.

AmbientReferences

- Variable Cardinality
 - Unihandle
 - Refers to exactly one remote object
 - Multihandle
 - Refers to a set of remote objects
 - Omnihandle
 - Refers to all remote objects of a given type

AmbientReferences

- Variable Flexibility
 - Sturdy
 - Never breaks
 - Buffers messages
 - Elastic
 - Buffers messages
 - Breaks after timeout
 - Rebinds to new remote object
 - Fragile
 - Breaks immediately upon disconnection
 - Rebinds to new remote object

AmbientTalk - Issues

- Disconnections
 - Default is to buffer and resend messages
 - Use AmbientReferences for more options
 - Event handler for disconnection
 - Programmer can recall and break connection manually
- Dynamic topology
 - Remote objects can be discovered dynamically
- Small devices
 - Runs on Java, more overhead than

AmbientTalk – Issues Continued

- Indirect addressing
 - Object handles by object type
 - AmbientReferences
- Resource discovery
 - Built into language with event handlers
- One-to-one and group communication
 - One-to-one through regular object handles
 - AmbientReferences for group communication
- Decentralized
 - Completely

AmbientTalk - Evaluation

- Advantages
 - Good disconnection handling
 - Built in resource discovery
 - Communication to remove objects is simple
- Disadvantages
 - Have to think “asynchronously”
 - Completely new language, unfamiliar
 - Resource discovery through beaconing