ASPECT-ORIENTED SIMULATOR FOR DISTRIBUTED MST CONSTRUCTION ON WIRELESS NETWORKS

ABDULLAH BÜLBÜL
ÖMER FARUK UZAR
UTKU OZAN YILMAZ

CS 586 Aspect-Oriented Software Development
Course Project

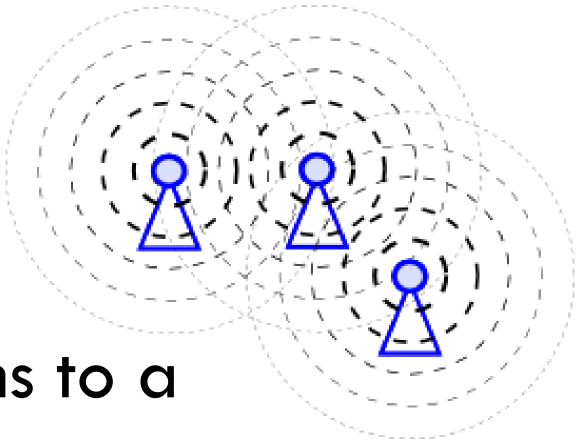
Outline

Introduction

- Wireless networks are integral parts of our lives
- Simulating the behavior of a network is a key concern
  - Real
    - Difficult to deploy
    - Costly
    - Restricted test conditions
    - More reliable
  - Simulation
    - Easy configuration
    - Cheaper
    - Broader test cases
- Distributed algorithms are very important to wireless networks
- AOSD can be used to develop elegant solutions to the problems of network simulators

Wireless Networks

- Covers our daily life inevitably
  - Mobile phones
  - Wireless internet
- Benefit from distributed algorithms to a broader extent
- Time consuming and costly to deploy
- All nodes cannot be connected directly



Distributed Algorithms

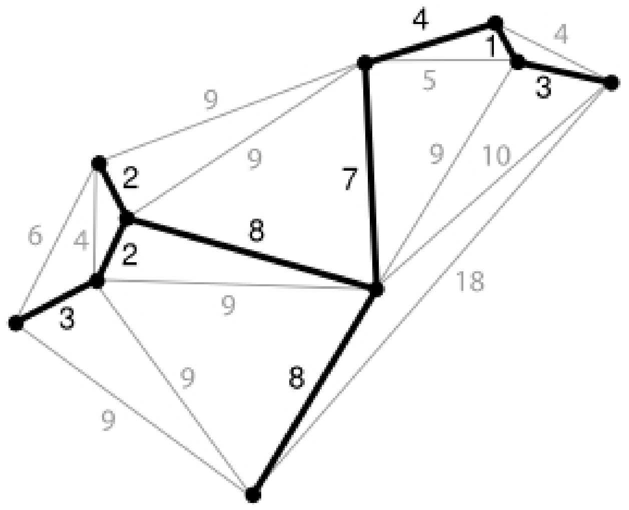
- Multiple hosts run an algorithm collectively to accomplish a goal
- Harder to design and implement than their centralized counterparts
  - Synchronization
  - Stability
  - Communication
  - Even harder if all nodes are not directly connected

Simulators for Wireless Env.

- Deploying wireless networks is time consuming and costly
  - Deploying a network which is not suitable for the needs of the situation is out of question
- Simulating the network prior to actually building it is a key concern
  - Understand the behavior of the network
  - Brings diversity to test cases
  - Can test marginal cases which are hard to reproduce

Minimum Spanning Tree

- Spanning tree: A tree that includes all nodes of a network
- Minimum spanning tree: The spanning tree on which the total weights of the tree edges are the minimum
- A fundamental problem in the fields of Graph Theory and Computer Networks
- Centralized algorithms: Kruskal's &Prim's Algorithms
- Distributed algorithm: GHS Algorithm
  - Proposed by Gallager, Humblet and Spira
  - Assumes Synchronized Environment



Legacy Work

- In a previous study, a distributed program that builds an MST for a network is implemented
  - First, creates network topology and sets up connections
  - Then, builds the MST in O (log n) rounds
  - Finally, broadcasts given messages
  - Text based outputs

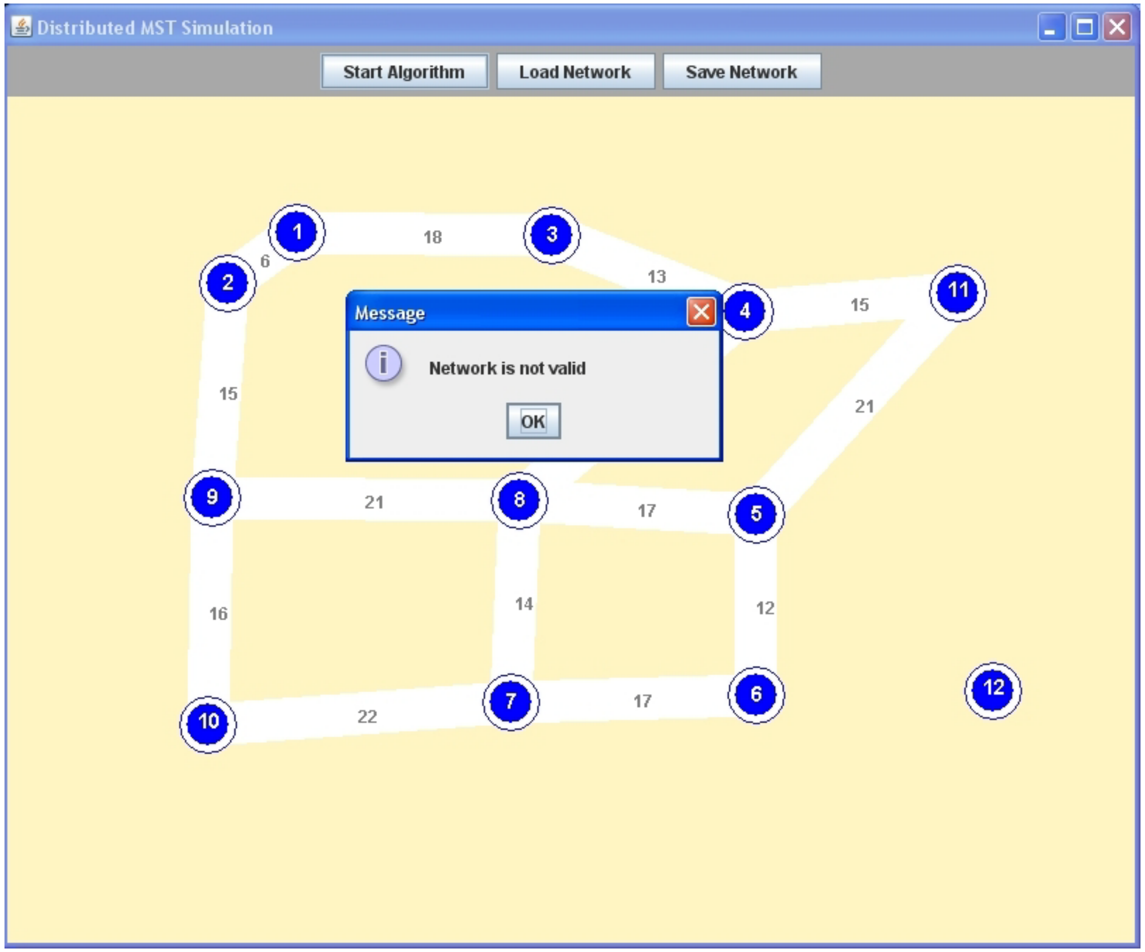
Problem Statement

- Inspired by this program, we designed and implemented:
  - An aspect oriented simulator for distributed MST construction on wireless networks
  - In a centralized manner
  - With better user interaction

Requirements Analysis

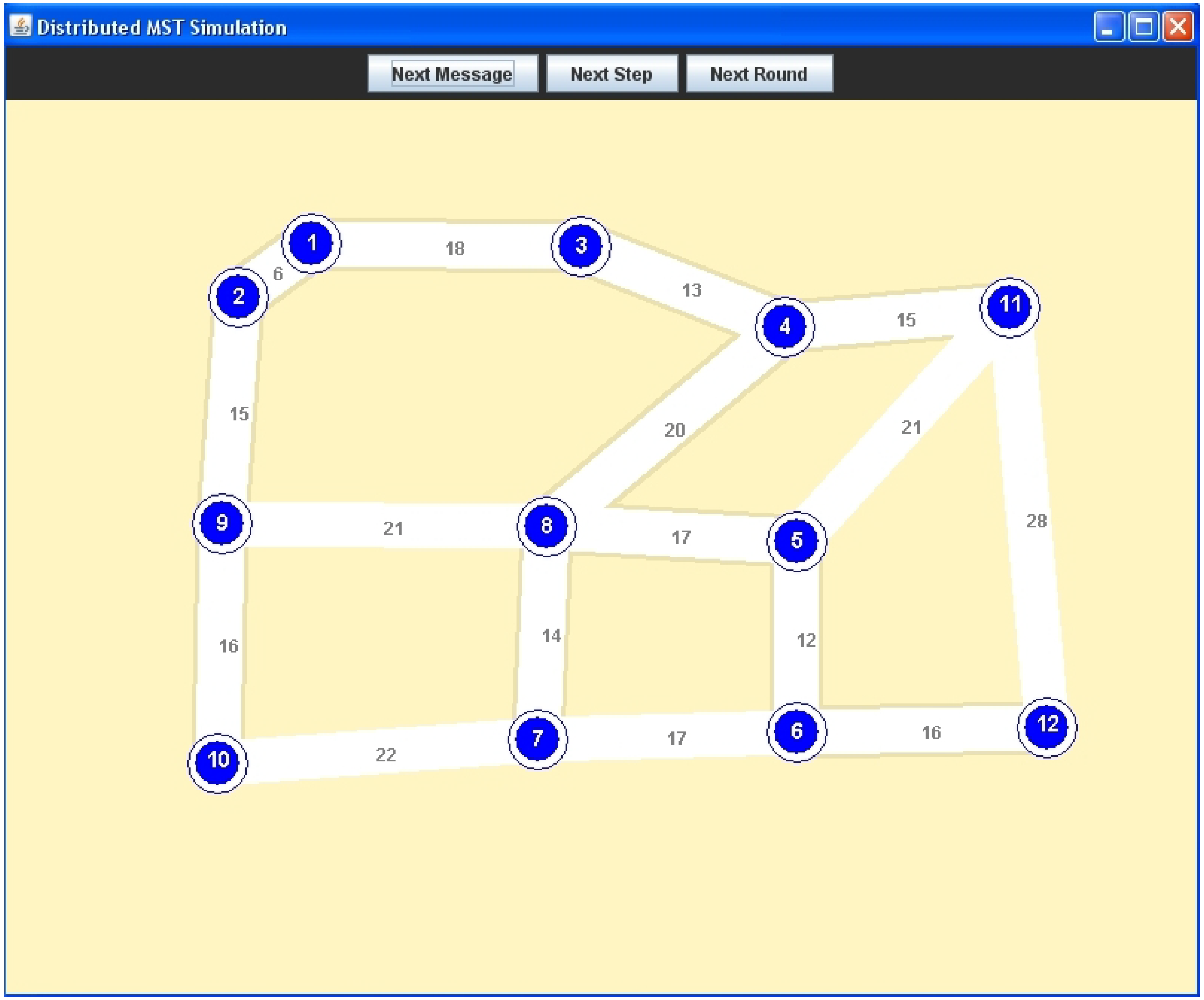
- Synchronization
  - Algorithm stability
- Step-by-step observation of algorithm
- Central visualization of different components
- Nodes oblivious to the overall structure
  - Preserve distributiveness

User Interface Prototypes

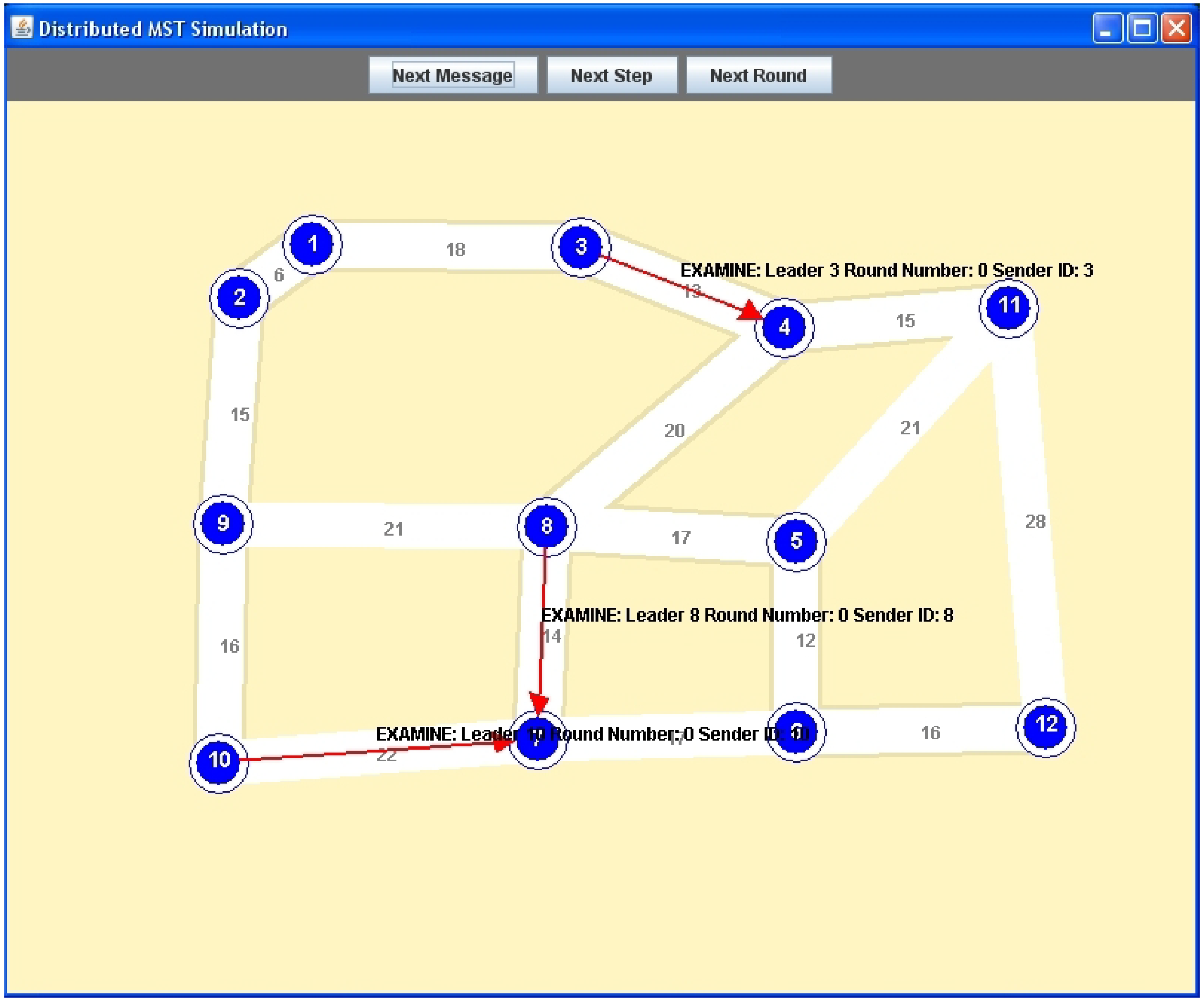


Network is not valid

User Interface Prototypes (2)



User Interface Prototypes (3)

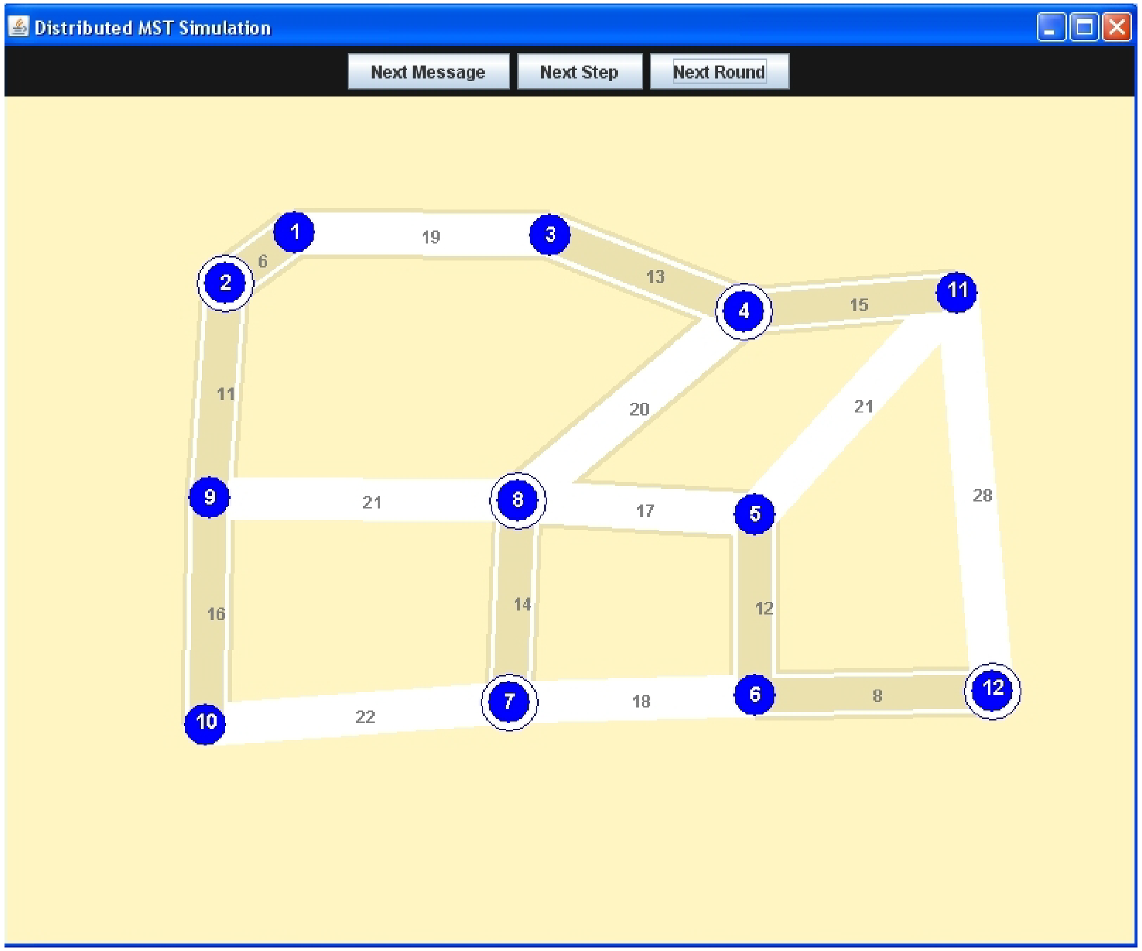


EXAMINE: Leader 3 Round Number: 0 Sender ID: 3

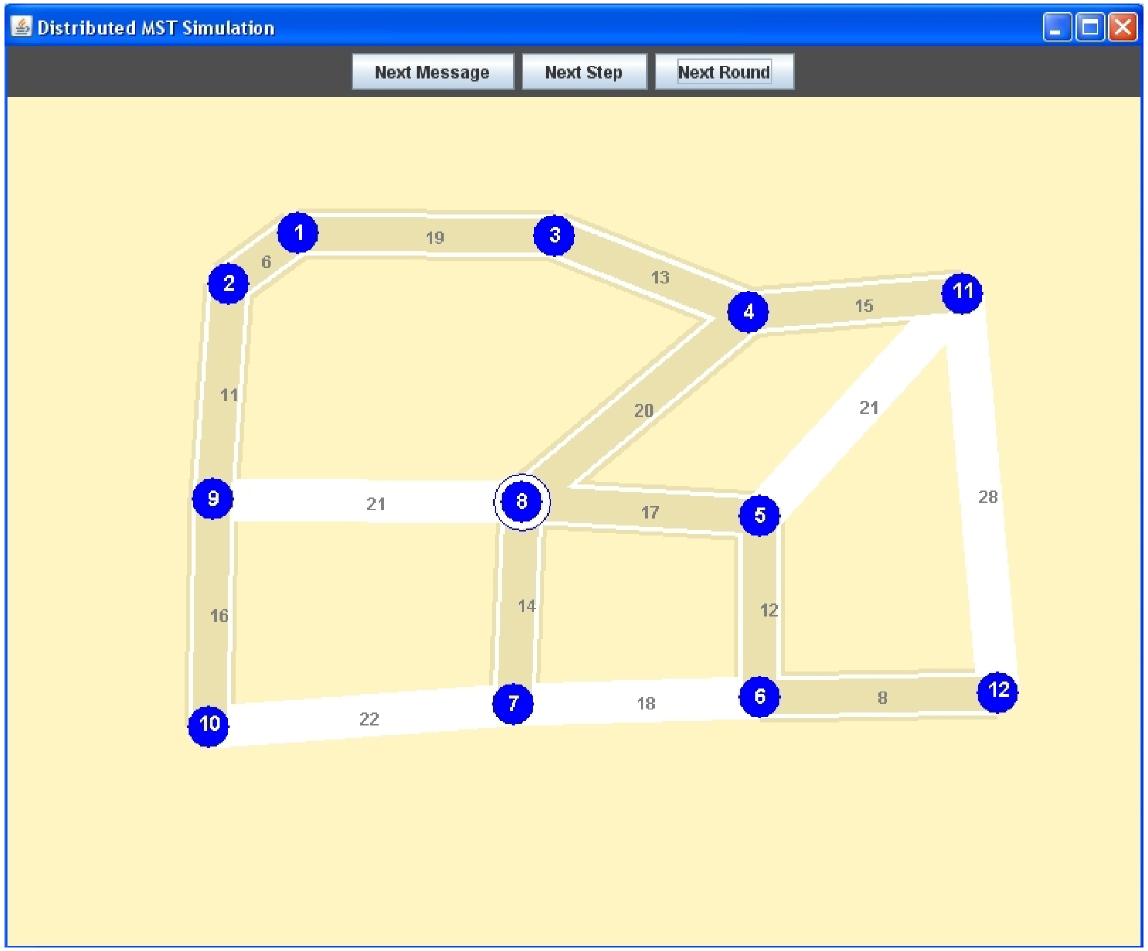
EXAMINE: Leader 8 Round Number: 0 Sender ID: 8

EXAMINE: Leader 10 Round Number: 0 Sender ID: 10

User Interface Prototypes (4)



User Interface Prototypes (5)



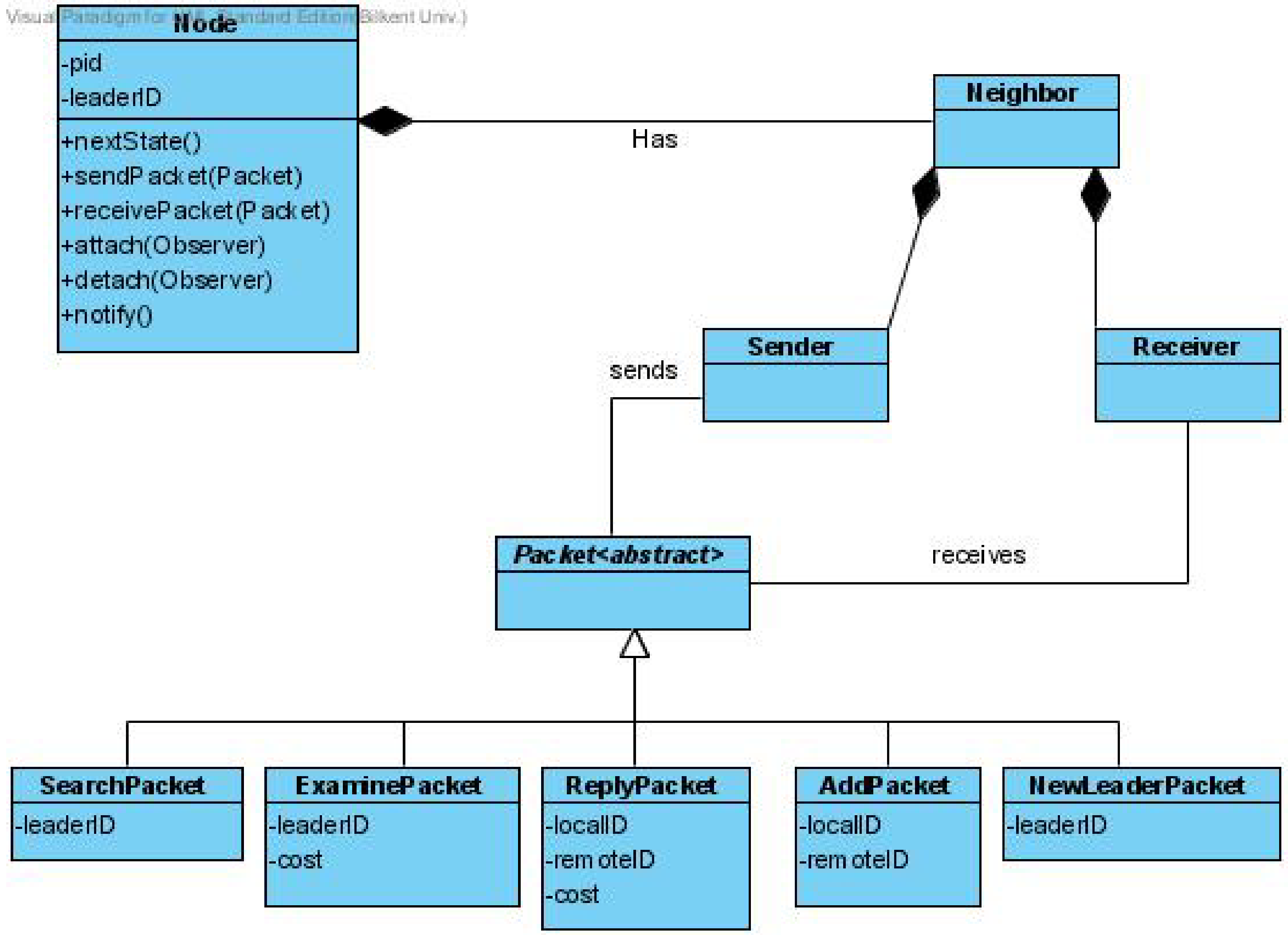
Our Approach

- Apply Object-Oriented Design
  - Use OO Design Patterns
- Identify the Crosscutting Concerns
- Define the Necessary Aspects
- Implement Aspects using AOP

Object Oriented Design

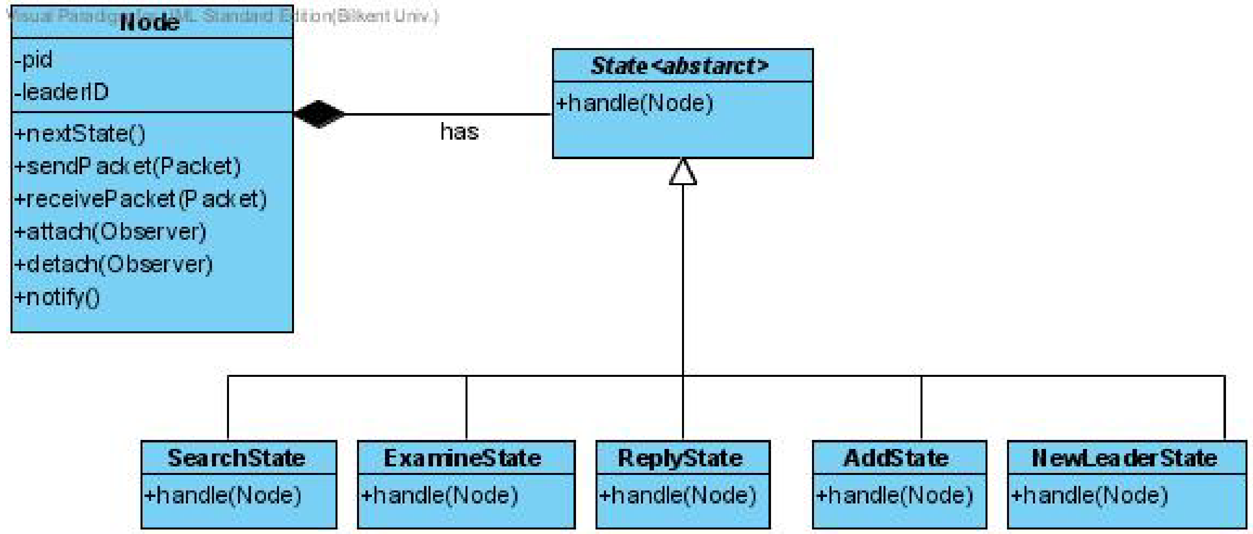
- The network is mainly represented by Node, Neighbor and Packet objects
- Includes 17 classes
- State and Observer patterns are utilized
  - State pattern is used for processing incoming/outgoing packets according to the current state
  - Observer pattern is used for updating the user interface

Object Oriented Design (2)

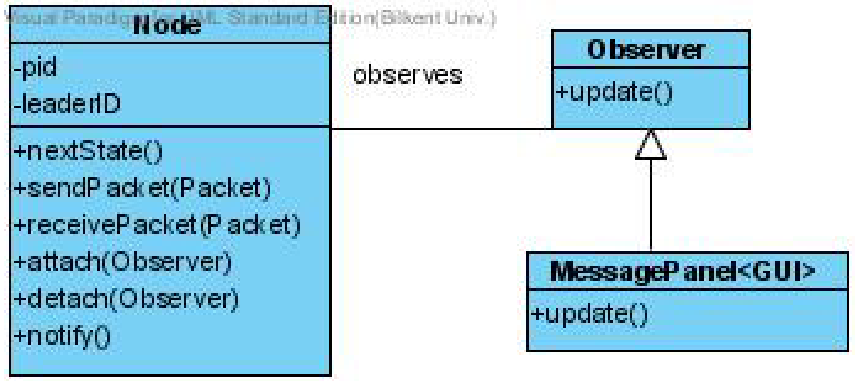


Object Oriented Design Patterns

- State pattern



- Observer Pattern



Crosscutting Concerns

- Synchronization
  - Required to make all state transitions at the same time
- Validation
  - Validates input and output
- Display
  - Enables the usage of different views according to attribute values
- Control
  - Required to view intermediate steps

Synchronization Aspect

- GHS algorithm assumes an already synchronized environment
  - Synchronization should be checked at multiple points
  - Providing synchronization inside the algorithm causes tangling
- Synchronization aspect solves the problem
  - pointcut nextState: call(* Node.nextState(..))
  - before nextState()
    - set *Node i* as ready for next state
    - suspend *Node i* until all nodes become ready for next state

Validation Aspect

- There are two points to validate:
  - Input validation
    - Separate this concern from main algorithm
    - Avoid scattering
  - Output validation
    - Used for development
    - Needs to be removed after development stage

Validation Aspect (2)

- Input validation (around advice)
- Around program execution
  - check if input is correct or not
  - run program if it is correct
  - warn user and terminate otherwise

Validation Aspect (3)

- Output validation
- After program finishes
  - check if MST is constructed correctly
  - warn developer if it is not

Display Aspect

- Display concern causes tangling
  - Network elements should not be responsible for visualizing themselves
- Network visualizer
  - Traces the changes in the network
  - Visualizes them

Display Aspect (2)

- Catch communication (messages transmitted among nodes)

```
after(Neighbor n, Packet p) returning() : target(n) && args(p) &&
                                    (call( * Neighbor.sendMessage( Packet) ) ) {
   NetworkVisualizer.getInstance().sentMessage(n.getNodeId(),(int)(n.getPid()), p);
   NetworkVisualizer.getInstance().getVisualizer().repaint();
}
```

- Catch leader changes

```
pointcut newLeader( ) : set( int Node.leaderID );
after( Node n ) : target(n) && newLeader() {
    NetworkVisualizer.getInstance().setLeaderMessage(n.getPid(),n.getLeaderID());
}
```

- Catch addition of new MST edges

```
after(int node1, int node2,int senderId ) : args(node1,node2,senderId) &&
                                             call(ADD_MWOE.new(..)) {
   NetworkVisualizer.getInstance().addMSTedge( node1, node2 );
}
```

Control Aspect

- Controlled execution of the algorithm
  - User determines the continuation at control points
    - Stop at each message
    - Stop at each state
    - Stop at each round
  - Oblivious to the algorithm
- Aspect avoids scattered code
  - Check for user input at each control point >> scattering
  - Separate the control concern from the main module

Control Aspect (2)

- Control Aspect (pseudocode)
  - After a button is clicked
  - If the button is “next message” button
    - Send one message
    - Stop and visualize
  - If the button is “next step” button
    - Continue until all nodes finish this step (finished nodes waits)
    - Stop and visualize
  - If the button is “next round” button
    - Continue until all nodes pass to the next round (passed nodes waits)
    - Stop and visualize

Advantages of AOSD

- Avoid croscutting concerns by separating
  - Synchronization,
  - Validation,
  - Display,
  - Control
  - Concerns from the distributed algorithm design
- Provide a synchronized network environment
- Easier to extend & modify the visualization component
  - Oblivious to the distributed algorithm
- Control and validation aspects ease debugging the algorithm
- Easier to develop distributed algorithms
  - Do not deal with extra concerns other than algorithm design

Conclusion

- Implemented a simulator for distributed MST construction on wireless networks
  - OOSD
  - AOSD
- Handled Croscutting Concerns with AOP
  - Synchronization
  - Validation
  - Control
  - Display

End of Presentation

Thank you

Questions?