

Deployment of Sensor Networks: Problems and Passive Inspection

Matthias Ringwald, **Kay Römer** (ETH Zurich)



Sensor Networks

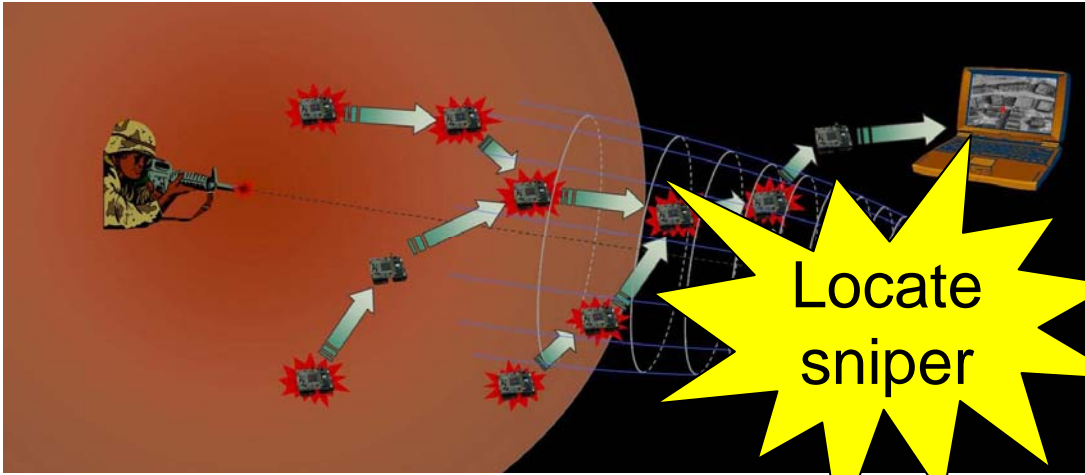
- Ad hoc network of sensor nodes
 - Perceive real world (sensors)
 - Process data (microcontroller)
 - Communicate (radio)



classify
frogs



measure
vibrations



Locate
sniper

Deployment of Sensor Networks

- Setup large network in the real world
 - Make transition from lab to reality
- Behave differently
 - Failure or poor performance
 - Reasons: radio channel, sensor input, physical strain
- Bugs are hard to find
 - Limited visibility of network state
 - Active instrumentation changes network behavior

Deployment Problems

- Single node problems
 - Death, reboot, faulty sensors, ...
- Link problems
 - No neighbors, neighbor oscillations, link congestion, ...
- Path problems
 - No route to sink, path oscillations, loops, ...
- Global problems
 - Low data yield, high latency, network partitions, ...

Passive Inspection

- Overhear network traffic and infer existence of problems
- Protocols provide indicators for the existence of problems
 - Death: no messages
 - Reboot: sequence number resets to zero
- Pro: No instrumentation of sensor network
- Con: Additional hardware, not all problems can be detected
- Main challenges
 - Incomplete information
 - No standards

WSN Deployment Problems

- Great Duck Island¹, Redwood Tree², Potato Field³, ...
 - Hardware failures: Moisture, battery depletion, etc.
 - Networking problems (link failures, no route, spanning tree construction), often only detected after deployment
- Existing tools (debugger, testbed, simulation, emulation) don't work for deployments
- No resources reserved to indicate state
=> Lack of visibility (No LEDS => “in the dark”)

[1] R. Szewcyk et. al. An analysis of a large scale habitat monitoring application. In Sensys '04.

[2] G. Tolle et. al. A macroscope in the redwoods. In SenSys '05.

[3] O. V. K. Langendoen et. al. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In WPDRTS '06.

Get Insight into Network

In-network monitoring: Sympathy⁴, Memento⁵

- “Heisenbugs”
- Waste of resources
- Monitoring suffers from network problems, too.

Active collection of network state
as part of WSN application

Passive observation: SNIF

- + No “heisenbugs”
- + No additional processing on node (load)
- + No changes in networking behavior

Passive collection of radio traffic
by separate network

[4] N. Ramanathan et al. Sympathy for the sensor network debugger. In SenSys '05.

[5] S. Rost et al. Memento: A Health Monitoring System for Wireless Sensor Networks. In SECON '06.

SNIF: Sensor Network Inspection

Framework

Example: how to detect a node reboot?



Sniffer

WSN radio communication

collects packets

Decoder

provides access to packet content

Processing

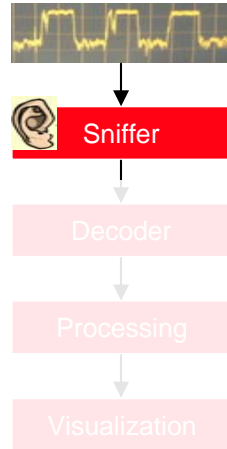
of packet contents

Visualization

to represent node and network state

Sniffer: Challenges

- Sniffer = additional node to overhear traffic in the sensor network
- No standard for WSN radio communication
 - Different settings for radio module (data rate, frequency)
 - Increasing number of Media Access (MAC) Protocols: S-MAC, B-MAC, SCP-MAC, WiseMAC, ..., DWARF, BitMAC
 - Different packet formats (preamble length, start-of-packet symbol (SOP), packet size, CRC)



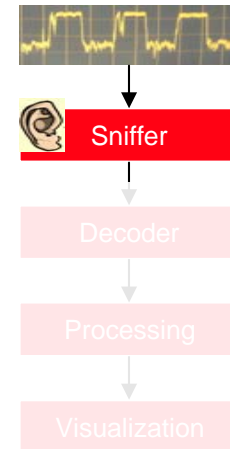
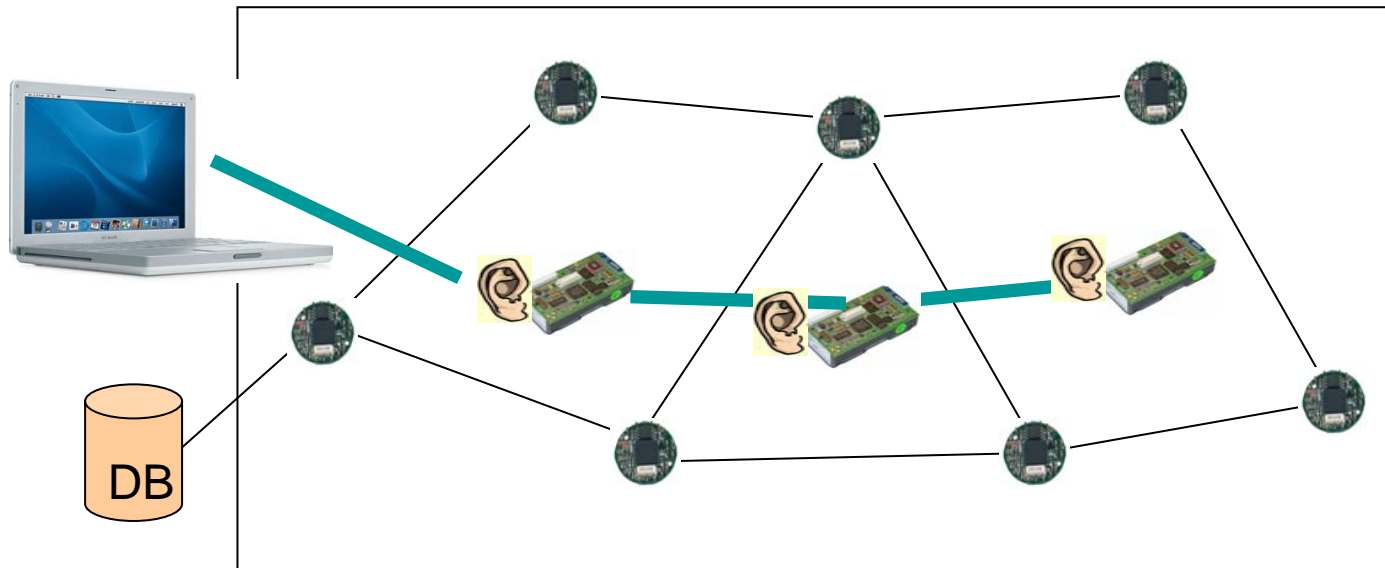
Sniffer: Solution

- Physical Layer - radio settings configurable:
 - Frequency
 - Data rate
- Different MAC protocols and packet formats
 - Always listening, waiting for **Preamble** and **Start-of-Packet(SOP)**
 - **SOP** and position of **Len** field configurable



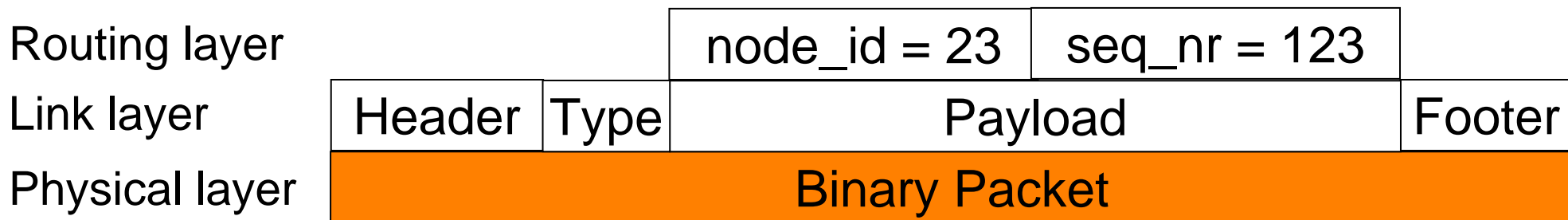
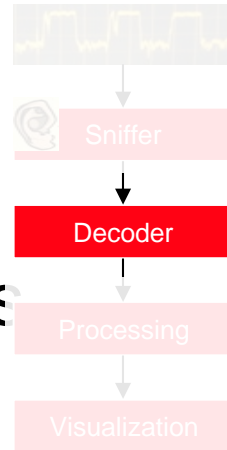
Sniffer Network

- Single sniffer cannot observe complete WSN
 - Network of sniffer nodes (synchronized)
- Sniffer nodes have two radios



Packet Decoder: Challenges

- Need access to header fields and packet contents
 - No standards!
- Common practice: C code for every protocol
 - E.g. Ethereal network analyzer



Packet Decoder: Solution

- Most sensornet apps are written in

C

- C str
layou

- Our App

C Struct

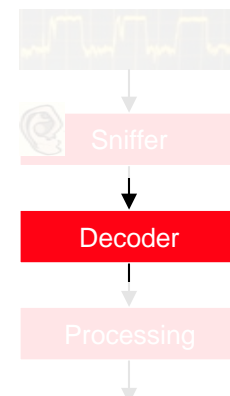
- C str
- + Pack
- + Varia

```
/** Basic TinyOS message */
struct TOS_Msg {
    uint16_t addr;
    uint8_t  type;
    uint8_t  group;
    uint8_t  length;
    int8_t   data [length];
};
```

```
/** Hallo beacon */
const int BEACON = 1;
struct Beacon_Packet :
TOS_Msg.data(type == BEACON) {
    u_int16 node_id;
    u_int16 seq_nr;
};
```

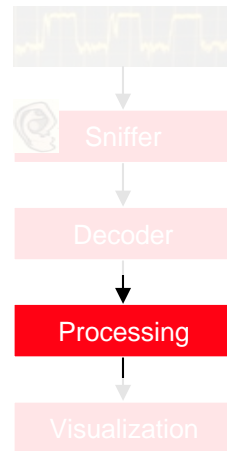
```
/** Basic TinyOS message */
struct TOS_Msg {
    uint16_t addr;
    uint8_t  type;
    uint8_t  group;
    uint8_t  length;
    int8_t   data [0];
};
```

```
/** Hallo beacon */
const int BEACON = 1;
struct Beacon_Packet
{
    u_int16 node_id;
    u_int16 seq_nr;
};
```



Packet Processing: Challenges

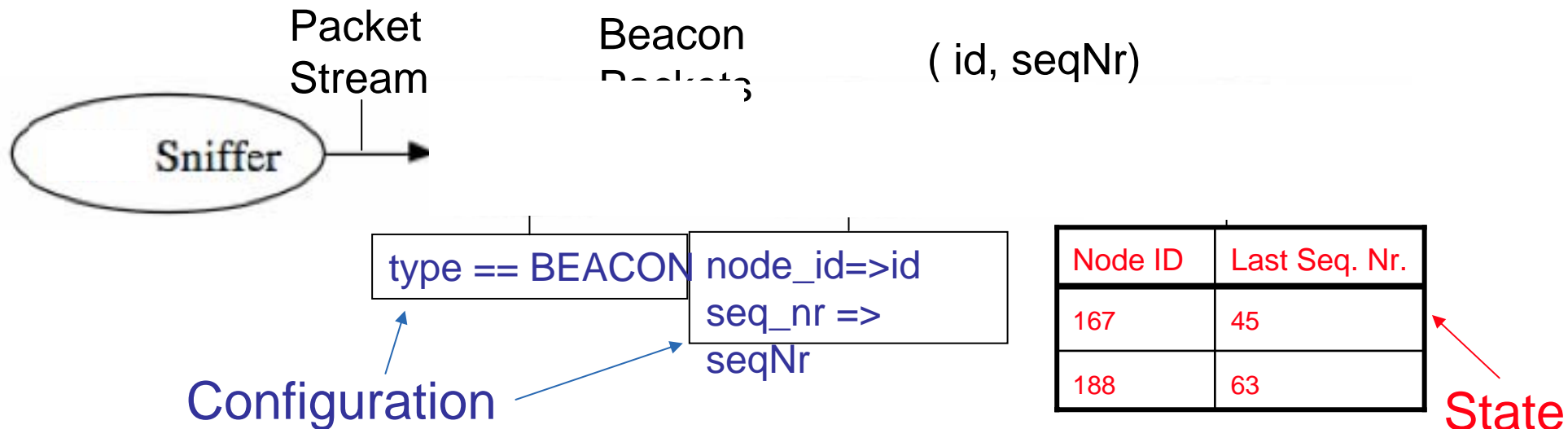
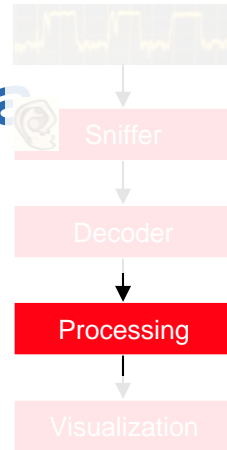
- Analyze packet trace to find indicators for problems
- Processing should be online
- Generalization (no standards!)
- Incomplete information



Online Packet Processing with Data Streams

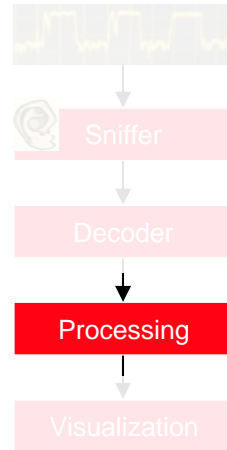
Data stream processing:

- Configurable operators
- Reusable operators and sub-graphs



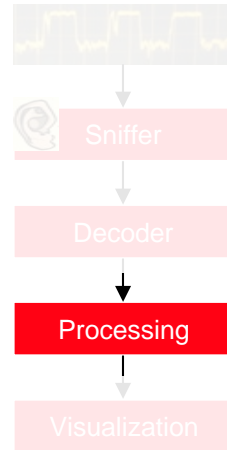
Indicator for Node Reboot

- Implemented by SeqReset operator
- Premise:
 - First beacon packet after reboot has sequence nr 0
 - Sequence numbers are increasing
- Rule: “new sequence nr \neq last sequence nr + 1 \Rightarrow reboot”



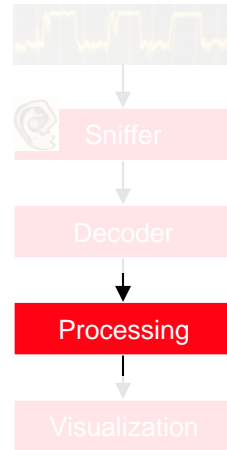
Indicator Challenges

- Incomplete information:
 - Packet loss: not all packets are received
 - Loss tolerant rule: “new number < last sequence nr => reboot”
 - Sequence counter could overrun internally: $0xffff \Rightarrow 0$
 - Internal node state not known (black box observation)
 - It is not possible to correctly distinguish overrun and reboot
 - Heuristic: use minimal time between beacons to calculate earliest expected time for received beacon packet



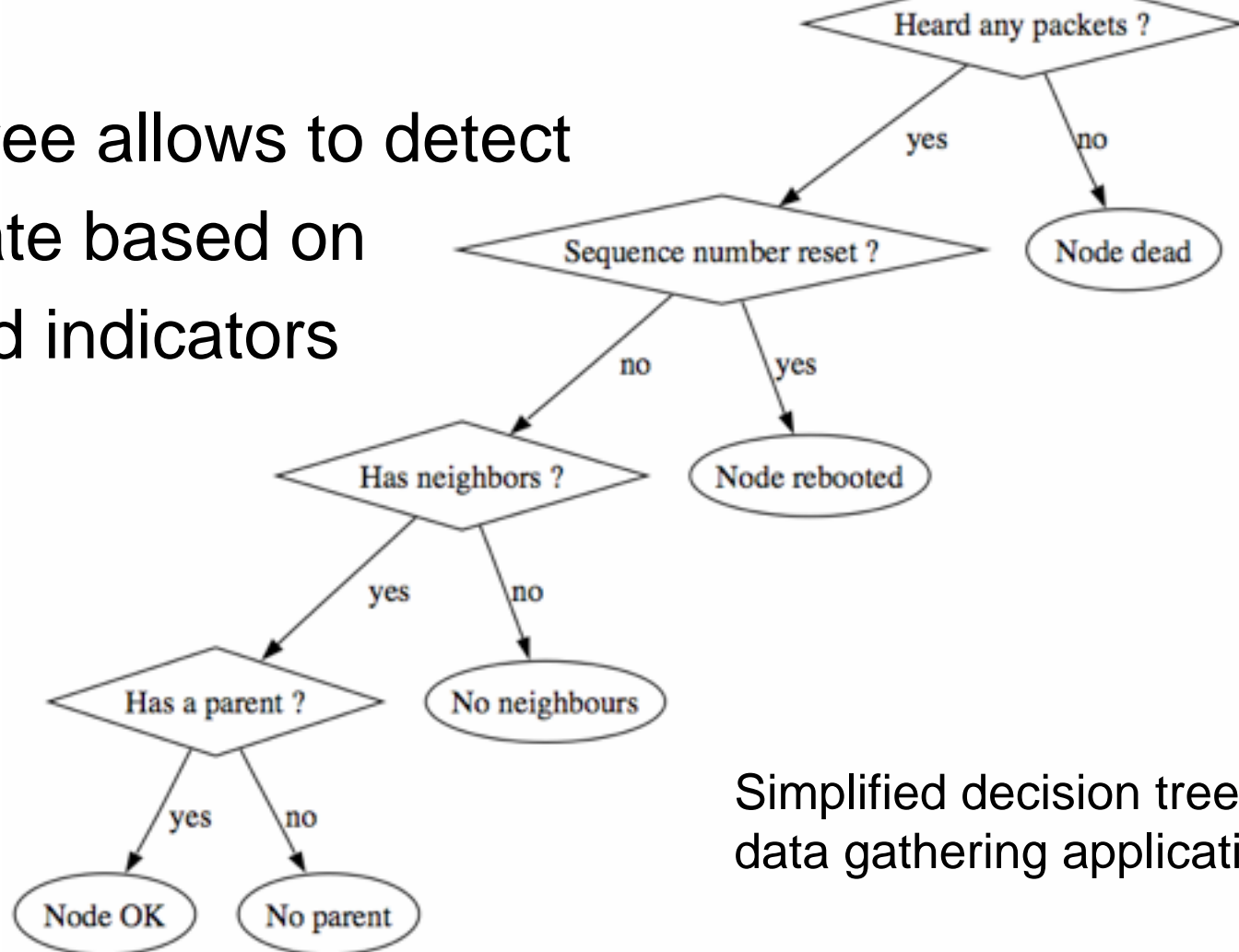
Root Cause Analysis

- Many problems to detect => different indicators
 - One data stream graph for each indicator
- Problem A can cause problem B:
 - Node has no neighbors => node cannot send data to sink
- Only root cause matters

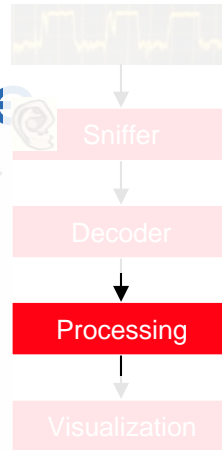


Root Cause Analysis: Decision Tree

Decision tree allows to detect node state based on observed indicators



Simplified decision tree for data gathering application

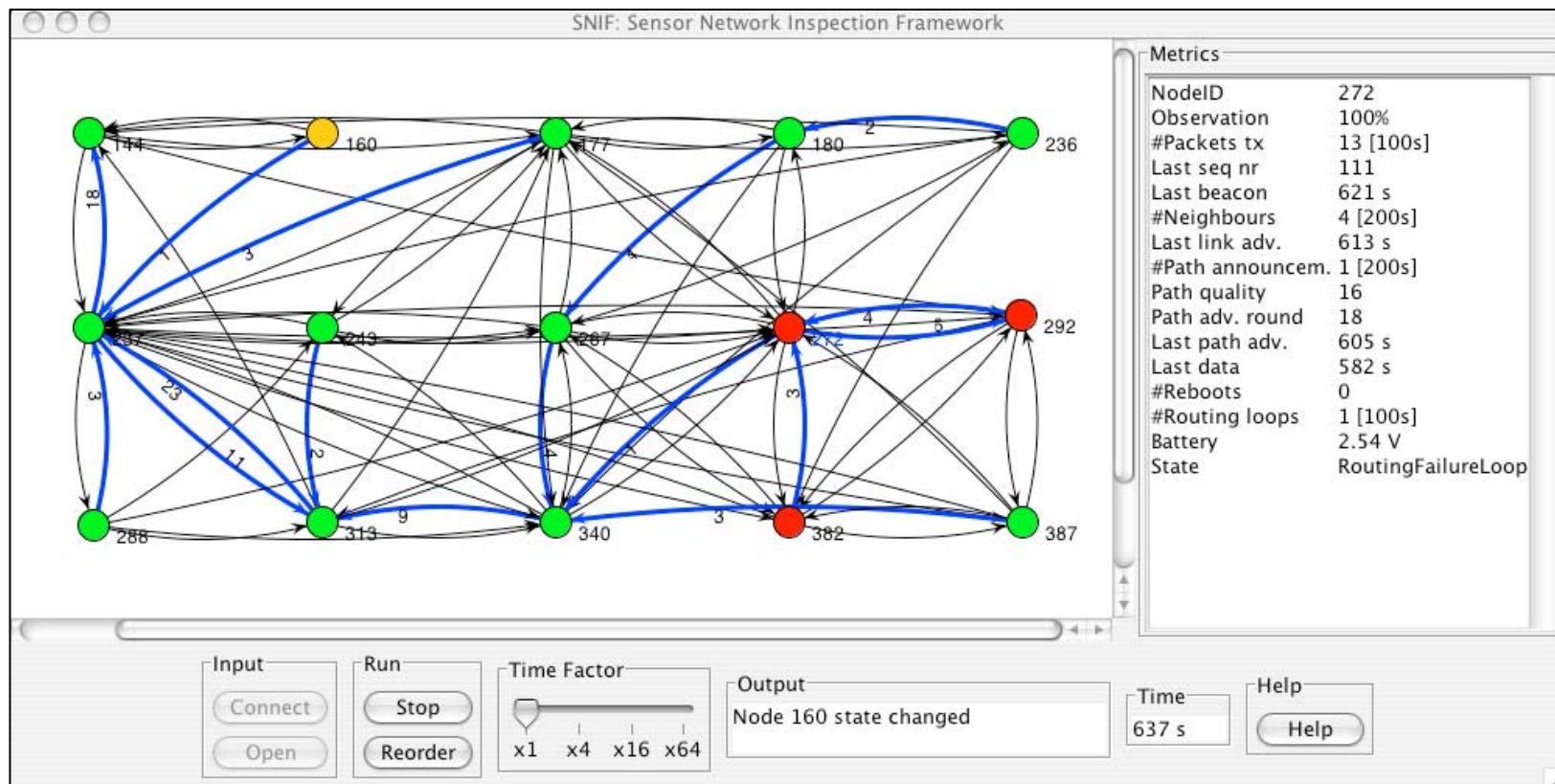
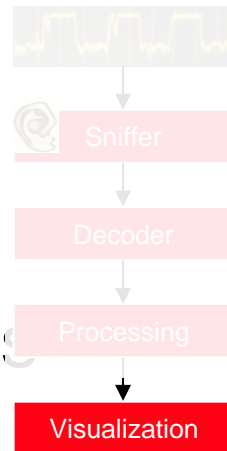


Generic Visualization

- Graphical representation of the observed network allows for intuitive inspection
- Problem: writing a special GUI for each application is cumbersome / will not happen
- SNIF provides a basic prototypical GUI to visualize node and network state - information from data stream graph can be attached to nodes and links

Generic Visualization

Indicators and node state can be inspected directly,
allows for (time-lapse) replay of recorded sessions



Other Passively Detectable Problems

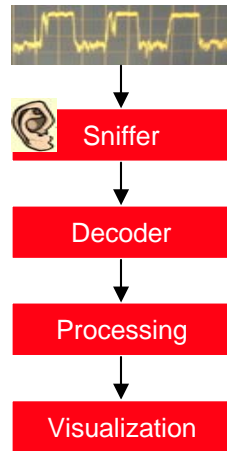
Problem	Indicator
Node crash	no packets received
No neighbors	no or empty link advertisements
No path	no route announcement
No route	data does not reach destination
Routing loop	data packets trace
Network partition	nodes on previous routes are crashed

SNIF Implementation

- Java 1.5 framework for data stream processing using template classes and prototype GUI
- Different packet sources:
 - Log files of EmStar simulator
 - DSN sniffer based on BTnodes
- Available at: code.google.com/p/snif/

Summary

- Passive inspection represents a valuable tool for deploying WSNs
- Challenges: Generalization, incomplete Information
- SNIF provides...
 - Distributed Sniffer
 - Generic Packet Decoder
 - Data Stream Processing with WSN specific operators, sub-graphs are reusable
 - Basic network and state visualization

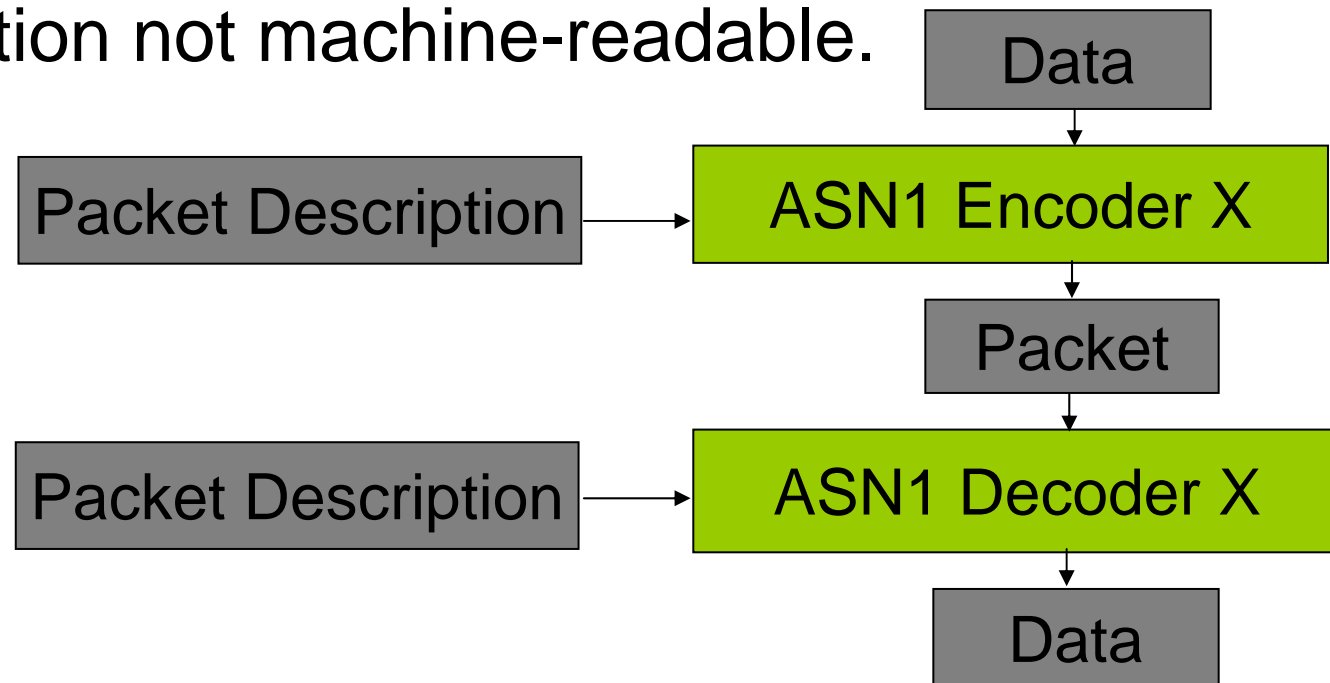


Thanks..

Why not ASN.1 ?

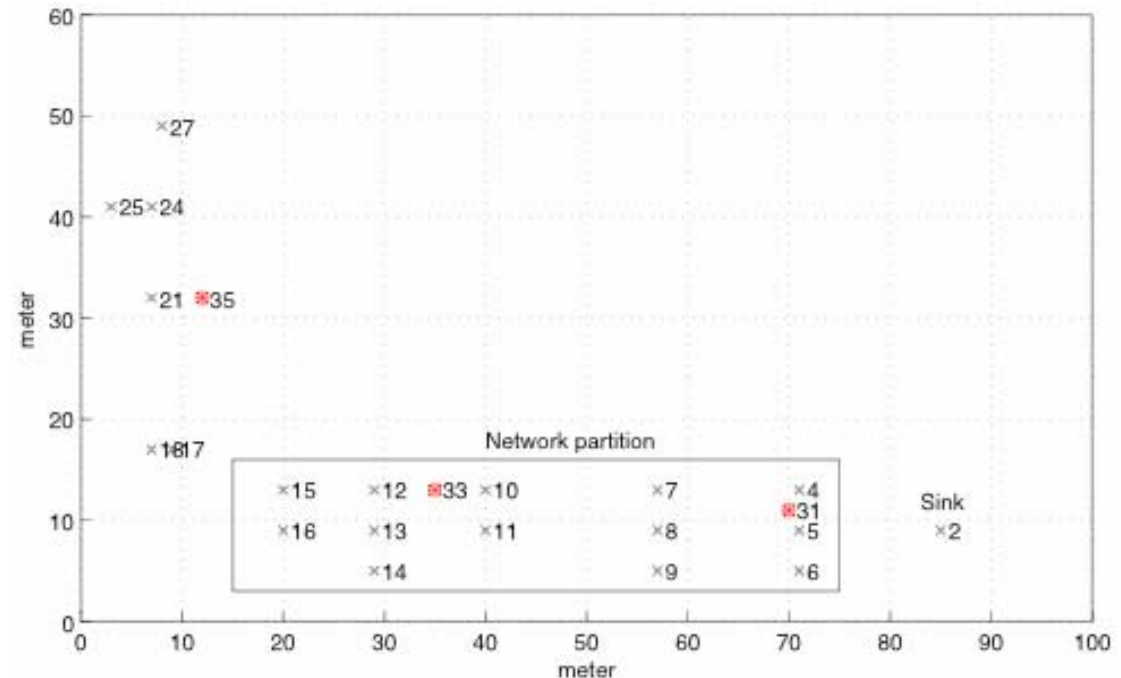
ASN.1 defines the syntax of data and provides various encoding schemes.

Scheme specification not machine-readable.



Evaluation Setup

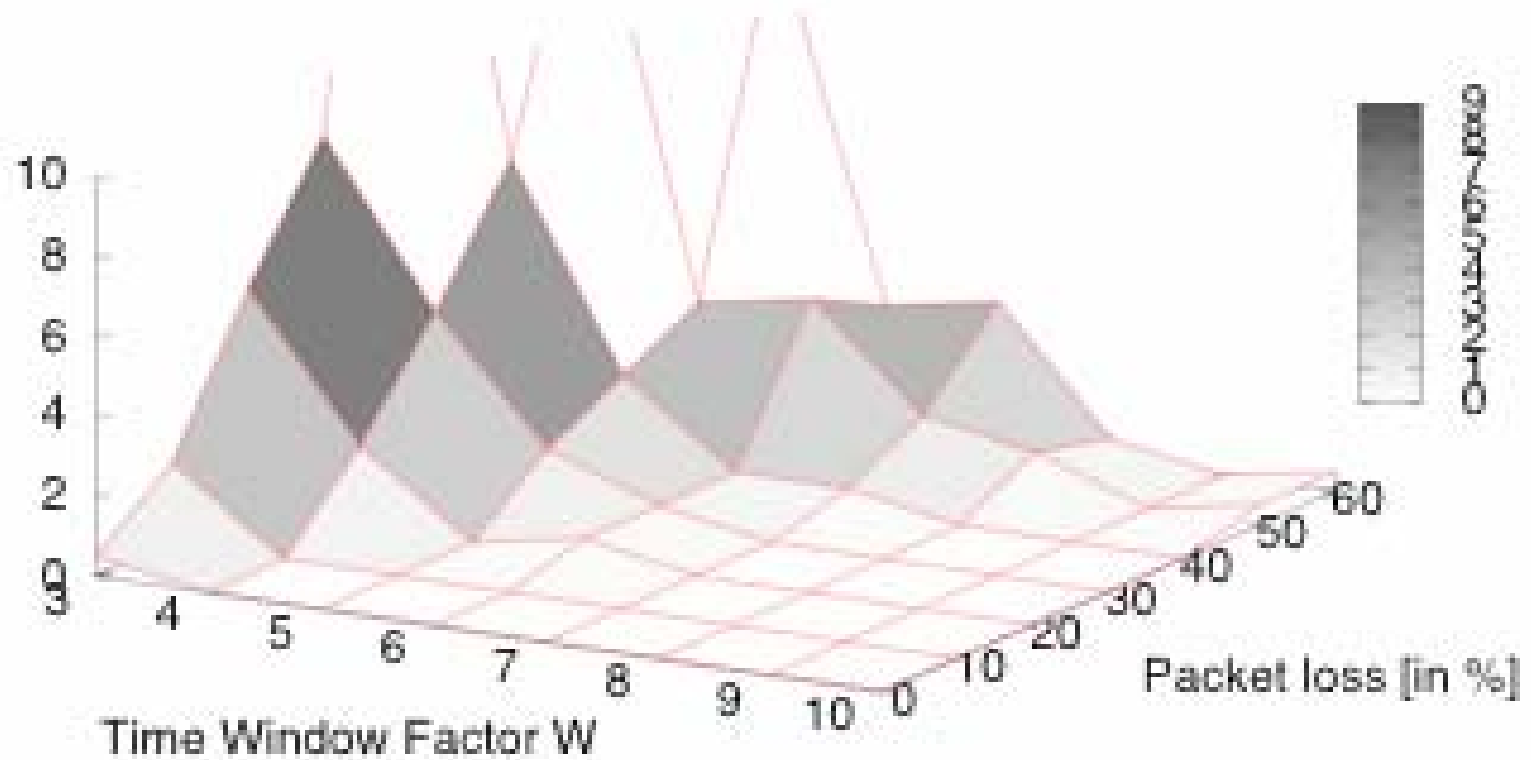
- EmStar emulation of Extending Sensing System (ESS) data gathering application
- Different faults injected: node crash, network partition, no data



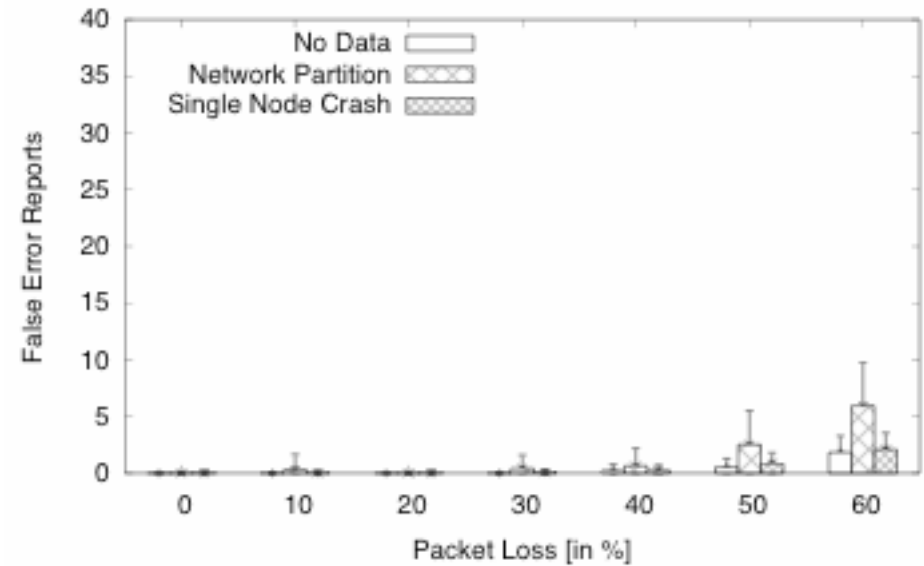
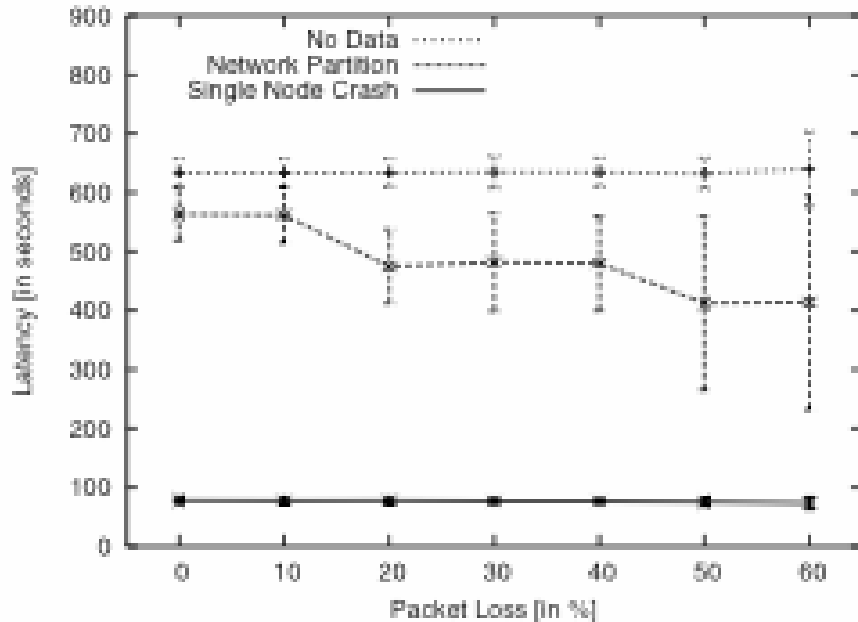
Evaluation: Metric, Parameters and Trends

- Metric:
 - detection latency of injected fault
 - and accuracy of fault detection
- Parameters:
 - time window factor
 - packet loss
- Trends:
 - wider time window => higher latency, higher accuracy
 - higher packet loss => lower latency, lower accuracy

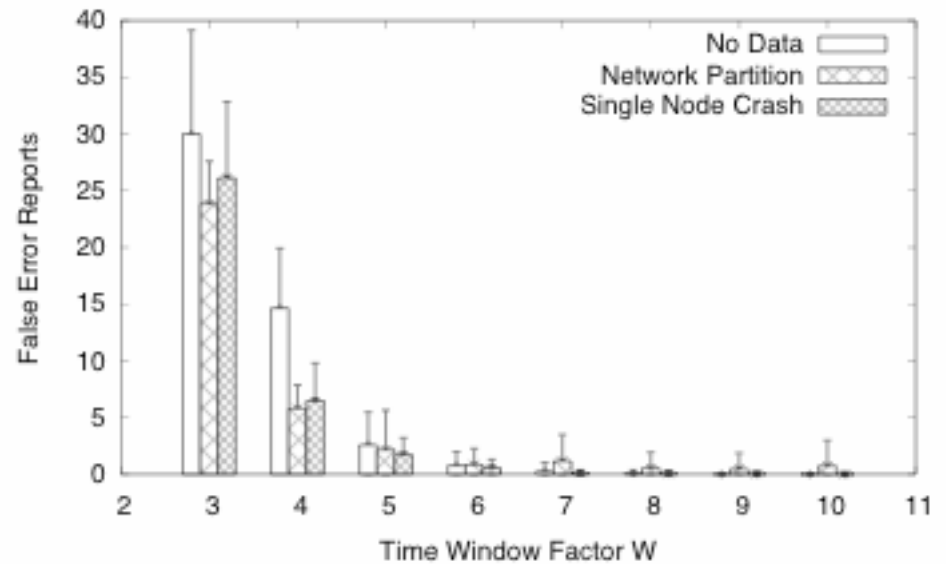
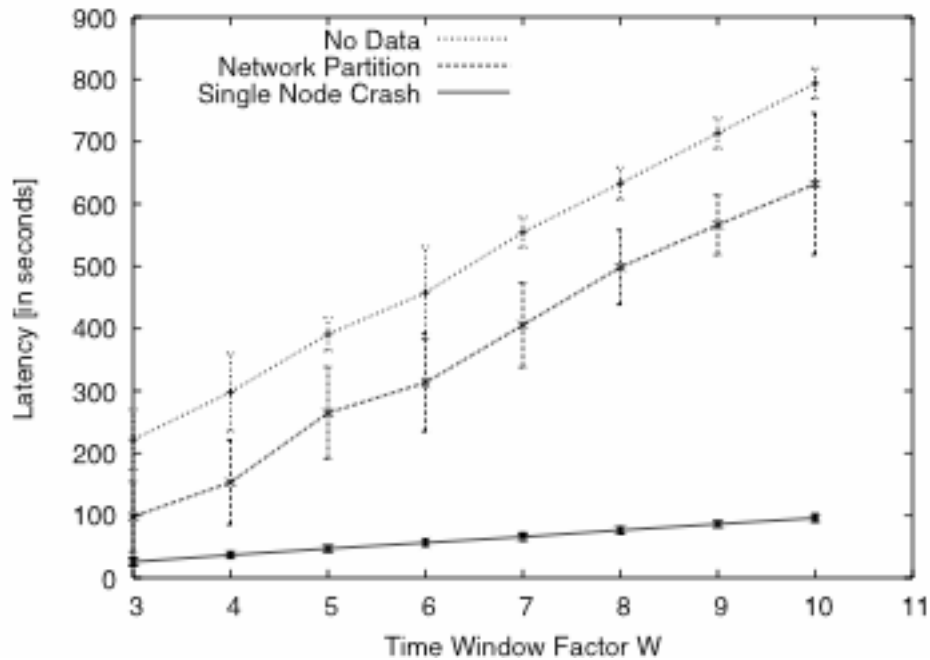
Evaluation: No Fault Injected



Evaluation: Latency and Accuracy vs. Packet Loss



Evaluation: Latency and Accuracy vs. Time Window Factor W



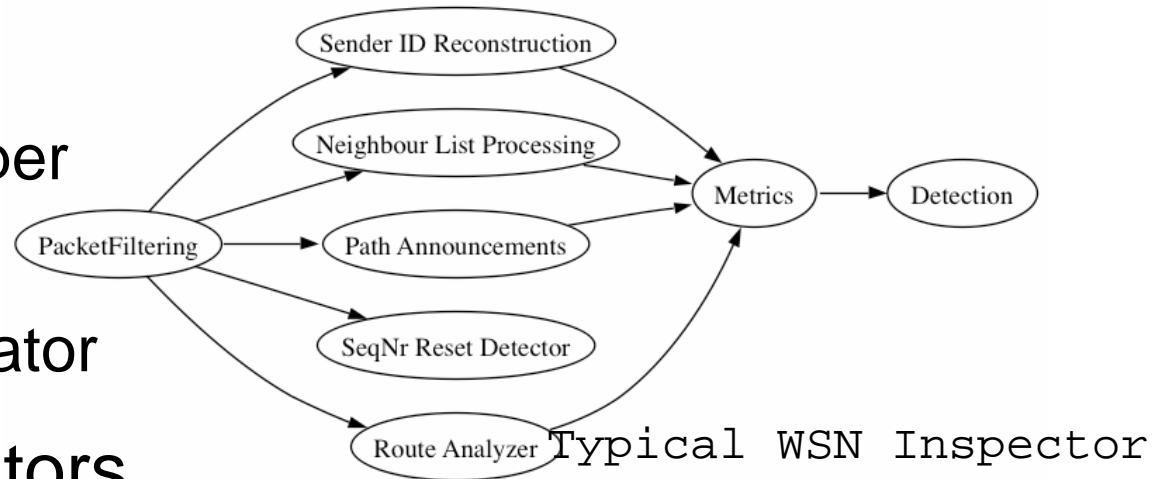
Data Stream Processing Operators

- Basic Operators

- Filter - Union - Mapper
- ArrayIterator
- TimeWindowAggregator

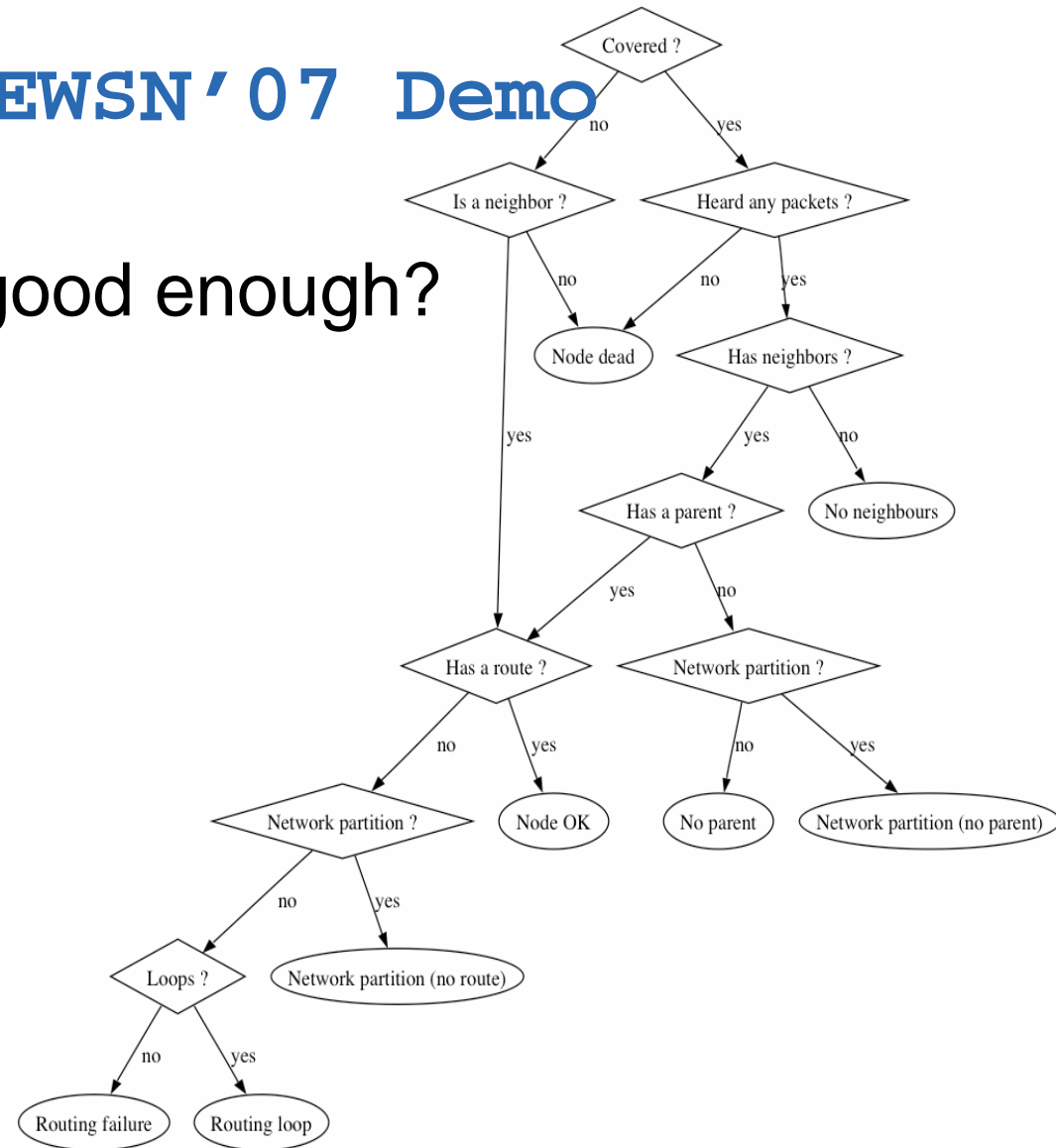
- WSN specific operators

- SeqNrReset
- PacketTracer (src addr missing in packet)
- PathAnalyzer (valid path to sink, routing loop?)
- TopologyAnalyzer (partition detection)
- Binary Decision Tree



Decision Tree of EWSN'07 Demo

- Can node be observed good enough?
- Reboot?
- Has neighbors?
- Is neighbor?
- Has parent?
- Routing loop?
- Network partition?



Dominant networking scheme for multihop WSN

- Convergecast: All generated data is routed to one or more sinks
- Basic algorithm:
 - Learn about neighbors: broadcast “hello” packets
 - Estimate link quality to neighbors: count hello packets received per neighbor and sent list of {neighbor, hello packet count}
 - Build spanning tree using some routing metric: pick best parent, calculate own cost and announce own path

Passive indicators for networking problems

- Periodic broadcast of “hello”-packets include sequence number: node crash + node reboot detection
- Periodic exchange of link quality (used for bi-directional link estimation): neighbor information + link quality checks
- Periodic announcement of route to sink: path to sink
- Periodic data packets

Missing information

- Tiny OS 1.x radio packets do not contain a source address field => Receiver does not know about sender
- Sender address is required for non-trivial broadcast packets => sender address is contained in pay-load
- Convergecast (“route to sink”) data lacks per hop source address. Non-trivial applications require address of data source, routing often uses sequence numbers for packet loss and duplicate detection

Missing information: Packet Tracer

- Assumption: routed packets can be identified based on their source address + sequence number
- Heuristic:
 - If packet is seen the first time, it was sent by the data source: store packet ID + destination address
 - If packet is seen again and sent to a new destination, consider last seen destination as source address, update destination address
 - If packet is sent to the currently stored last location, it's a re-transmission

Missing information: Topology

- Based on missing sender address, topology cannot correctly inferred.
- Weaker observation: If a data packet reaches the sink, a path exists from the node that generated the data

Missing information: Network Partition Detection

- If a connected network gets disconnected because of a link or node failure, those errors should be localized to the failed link/node.

ToDo

- Animation of data stream
- Visualization on two slides
- Deployment Problems: use deployment or real problems

Outline

- Passive observation
 - WSN deployment problems and the passive approach
- Example: passive online detection of node reboots
 - Issues and solutions
 - SNIF components
- Other detectable WSN problems
- SNIF implementation

What Problems to Detect?

- First goal: assert nodes and network are ok/working
 - Existing deployments motivate first step
 - Networking problems impede real applications
- Problem classes:
 - Node problems: node failure, node reboot, ...
 - Link problems: no neighbors, congestion, ...
 - Path problems: no parent, routing loops, ...
 - Global problems: low data yield, high reporting latency, ...

Example: How to Detect a Node

Reboot Online?

Basic idea: observe sequence number in “hallo” beacons

- issues

SNIF: Sensor Network Inspection Framework



WSN radio communication

collects packets



provides access to packet content



of packet contents



to represent node and network state