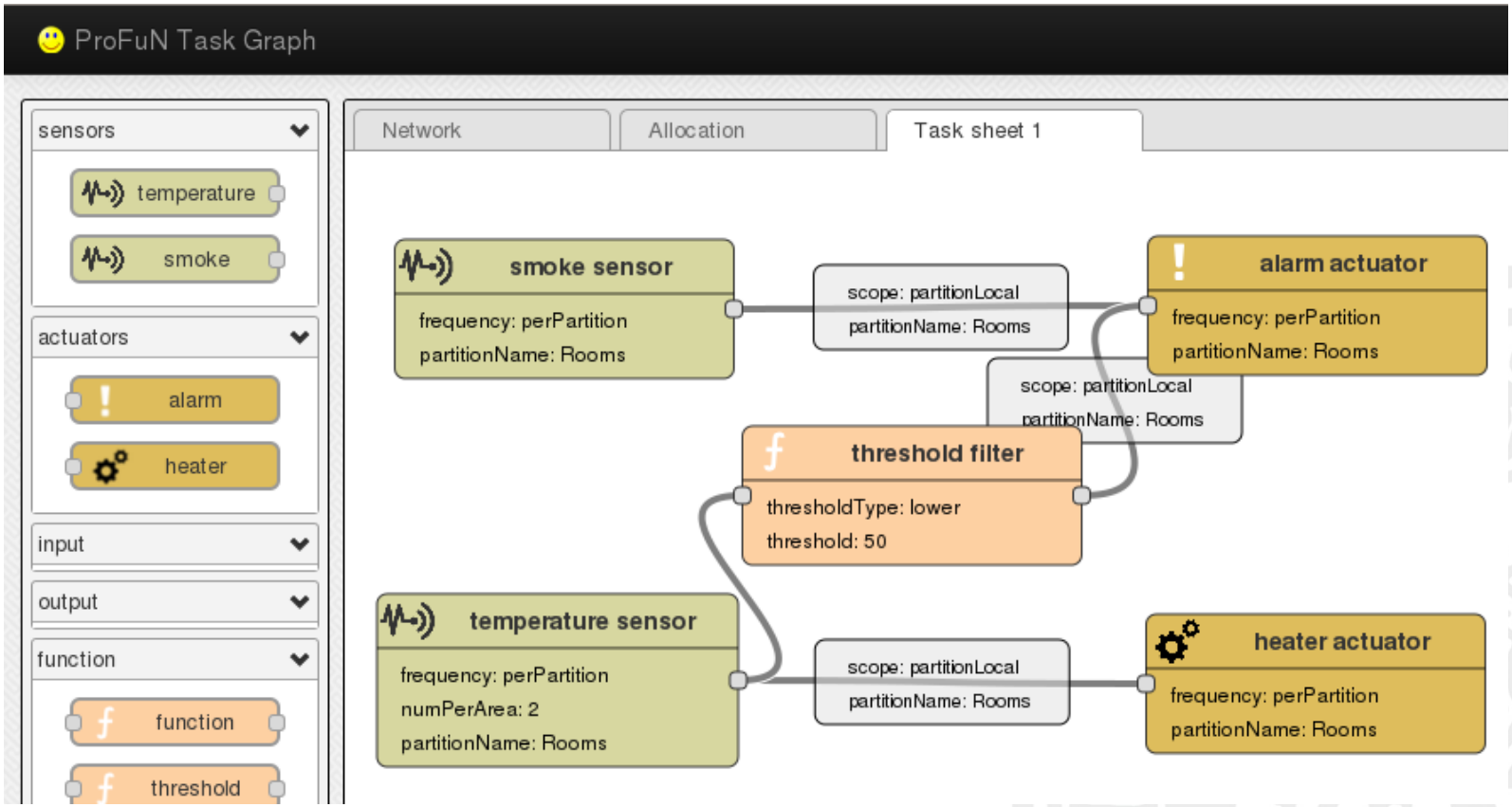
A tool for WSN application development
and maintenance



Atis Elsts
Martin Jacobsson
Farshid Hassani Bijarbooneh

ProFuN demo day, 22.10.2014

WSN application requirement examples

- Industrial monitoring
  - Requirement: reliably low delays are crucial
- HVAC (heat, ventilation, air conditioning) indoor monitoring
  - Requirement: network maintenance operations should be reduced
- Scientific data collection
  - Requirements:
    - end-to-end packet loss is not significant
    - at least *n* sampling sources should always be active

Handling these user requirements

- Allow the user to define probabilistic, end-to-end constraints for:
  - Packet delivery rate (PDR)
  - Delay
  - Energy

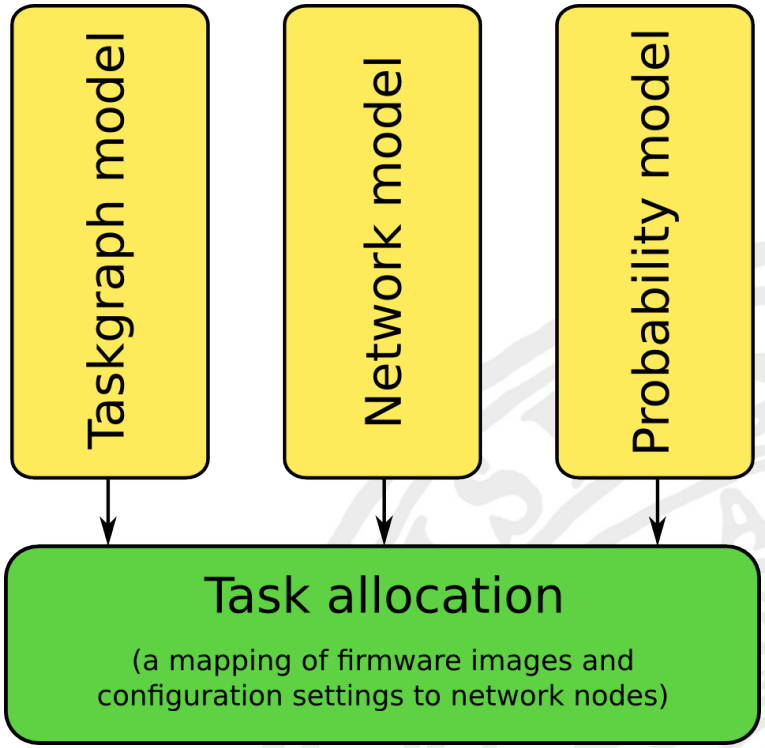
$$\mathbf{P}(Max(\text{Delay}) < 3000\,\text{ms}) \leq 0.98$$

$$\mathbf{P}(Avg(\text{PDR}) > 90\,\%) \geq 0.98$$

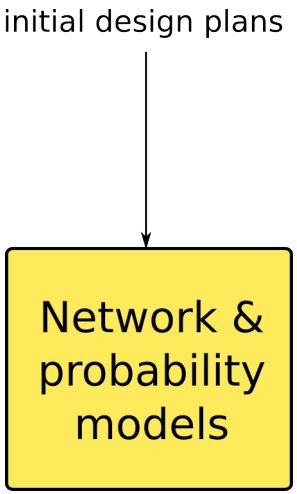
- Provide automatic or high-level mapping of functionality to network nodes
  - through WSN macroprogramming
- Perform run-time assurance of that functionality
  - through constant monitoring of network behavior
- Detect and repair faults
  - by automatically re-mapping functionality to initially redundant hardware nodes

WSN app development as modelling

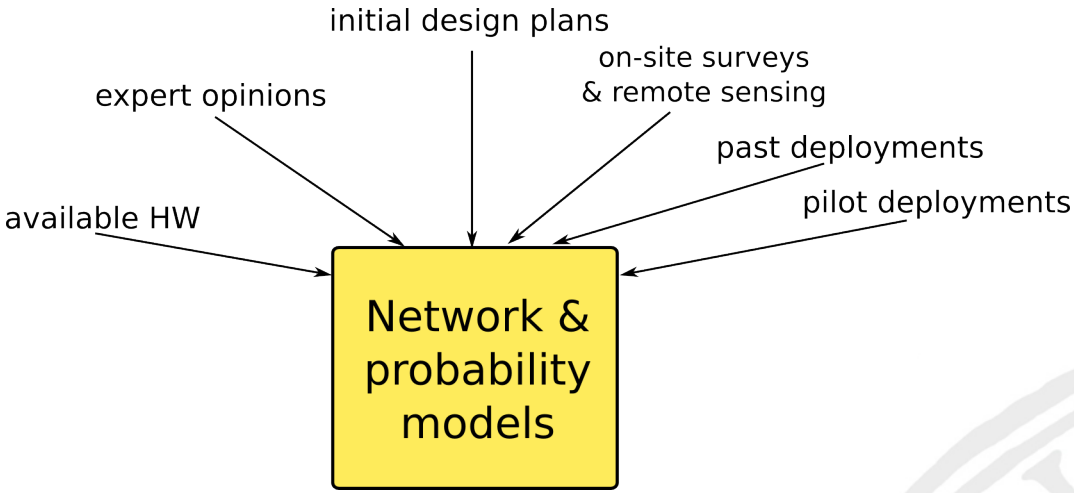
- Taskgraph model:
  - tasks
  - task firing rules
  - communication rules
  - constraints
- Network model:
  - node positions
  - node capabilities
- Probability model:
  - link qualities
  - failure rates
  - energy consumption



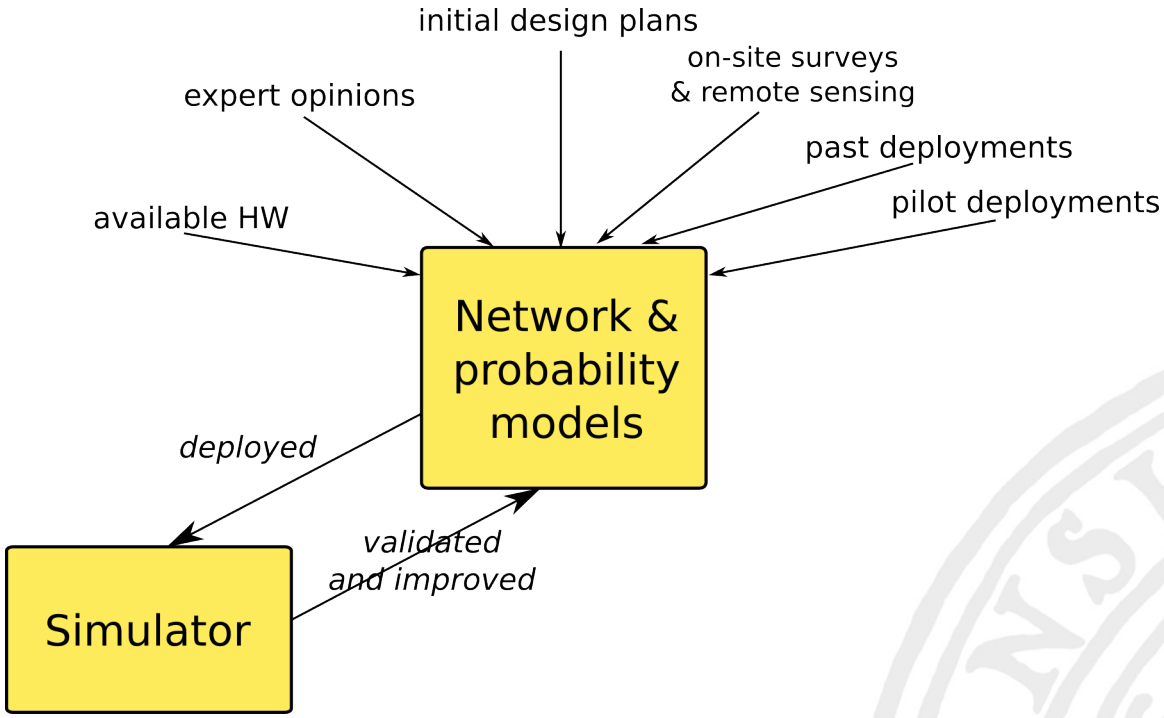
Creating and improving the models



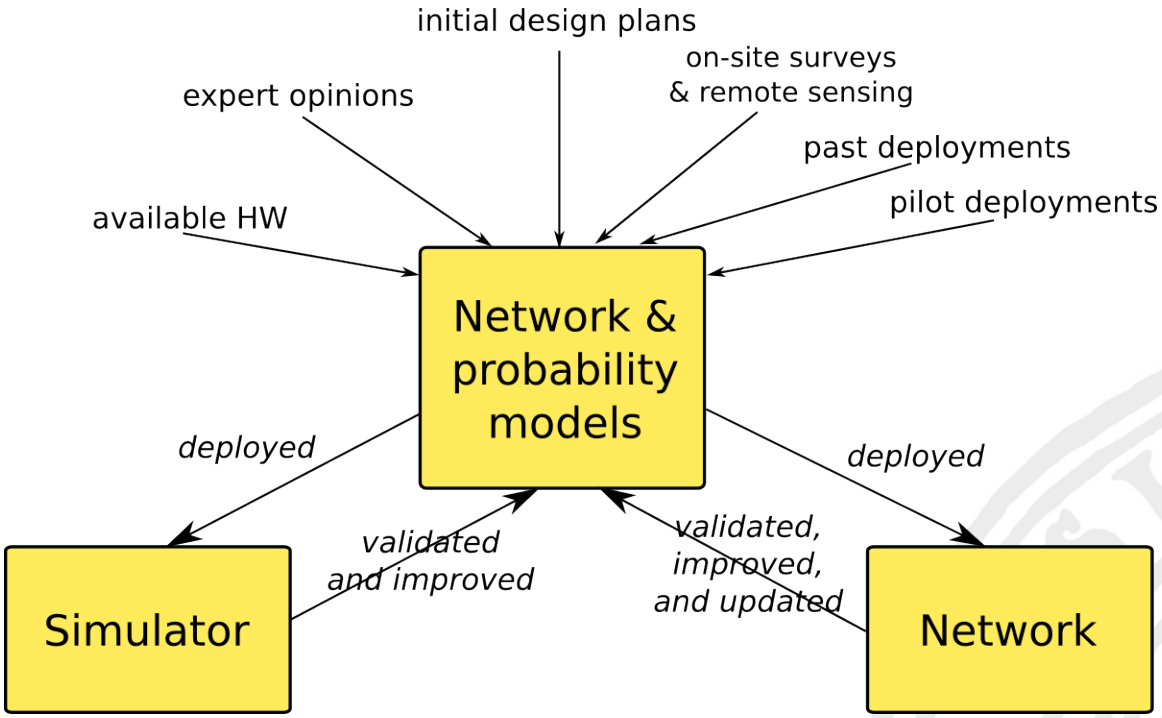
Creating and improving the models



Creating and improving the models



Creating and improving the models



Increasing the probability of success

- Task pair with a delay or PDR constraint; single sending task:

$$P(\text{success}) = P_{sender}(\text{success}) \cdot P_{link}(\text{success}) \cdot P_{receiver}(\text{success})$$

$$P(\text{fail}) \approx P_{\{sender,link\}}(\text{fail})$$

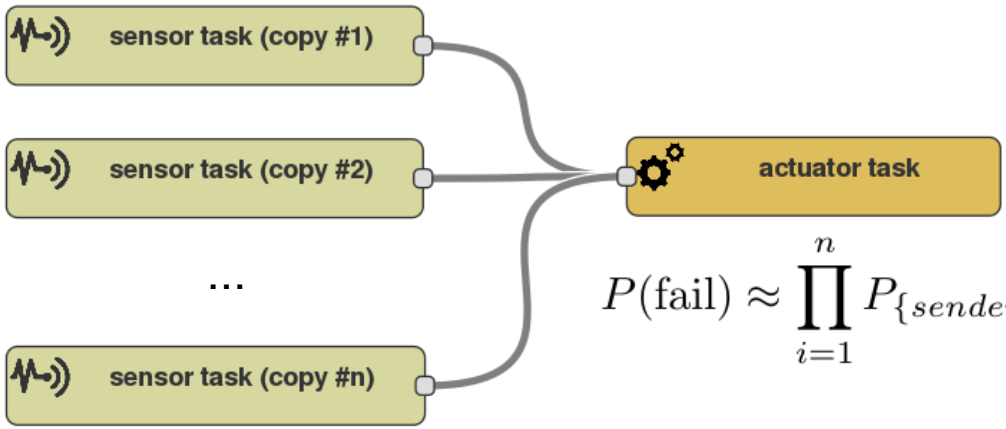
Increasing the probability of success

- Task pair with a delay or PDR constraint; single sending task:

$$P(\text{success}) = P_{sender}(\text{success}) \cdot P_{link}(\text{success}) \cdot P_{receiver}(\text{success})$$

$$P(\text{fail}) \approx P_{\{sender,link\}}(\text{fail})$$

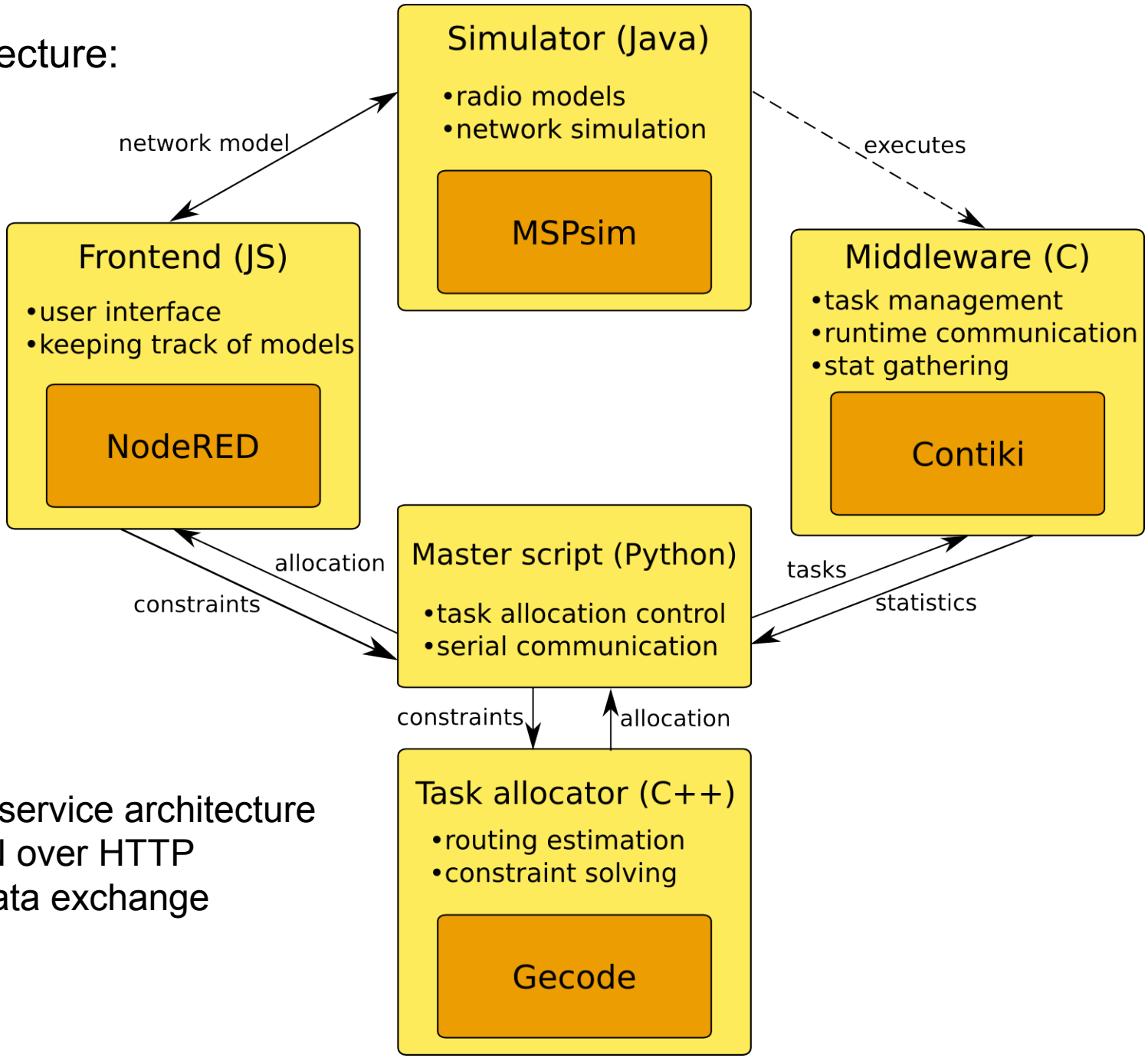
- Duplicated sending tasks:



$$P(\text{fail}) \approx \prod_{i=1}^{n} P_{\{sender_i,link_i\}}(\text{fail}) \ll P_{\{sender,link\}}(\text{fail})$$

Demo...

Architecture:



- Microservice architecture
- JSON over HTTP for data exchange

Task allocation

- **The problem**: map tasks to network nodes, **subject to**:
  - set of placement constraints, such as "*task t requires component c*"
  - set of communication constraints, such as

$$\mathbf{P}(Max(\mathrm{Delay}(\mathrm{task}_1, \mathrm{task}_2)) < 3000\,\mathrm{ms}) > 0.999$$

  - objective function

- Supported **objective functions**:
  - minimize TOTAL energy consumption rate
  - minimize MAXIMAL energy consumption rate
  - maximize minimal REMAINING energy
  - none (accept any constraint-satisfying mapping)

- Future plans:
  - coverage-based constraints and objective functions

Task allocation: algorithm ideas

- Need to solve a constrained optimization problem

- Approach: use Gecode constraint solver library

- Optimization approach: include custom search tree pruning as Gecode propagator

- Consequence: should start with mapping the part of the task graph that's least likely to change

- Idea #1:
  Start with the most central, highly connected and heavily constrained tasks

- Idea #2:
  Map “close” (e.g. directly communicating) tasks to “close” (e.g. directly connected) network nodes

The centrality of a task

- Different centrality measures (degree, closeness, etc.)

- Eigenvalue centrality:
  - A – adjacency matrix
  - x_v – centrality metric of vertex v
  - λ – the greatest eigenvalue of A

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t$$

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

- x_v can be iteratively approximated

- Mapping order of the allocator determined by:
  $merit_v = x_v + a \cdot c_v$

where c_v is the number of constraints on task v,

and a – constant.

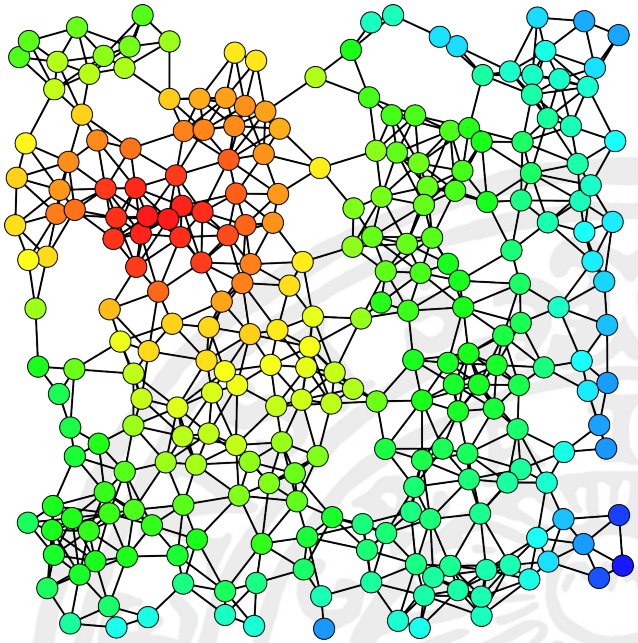
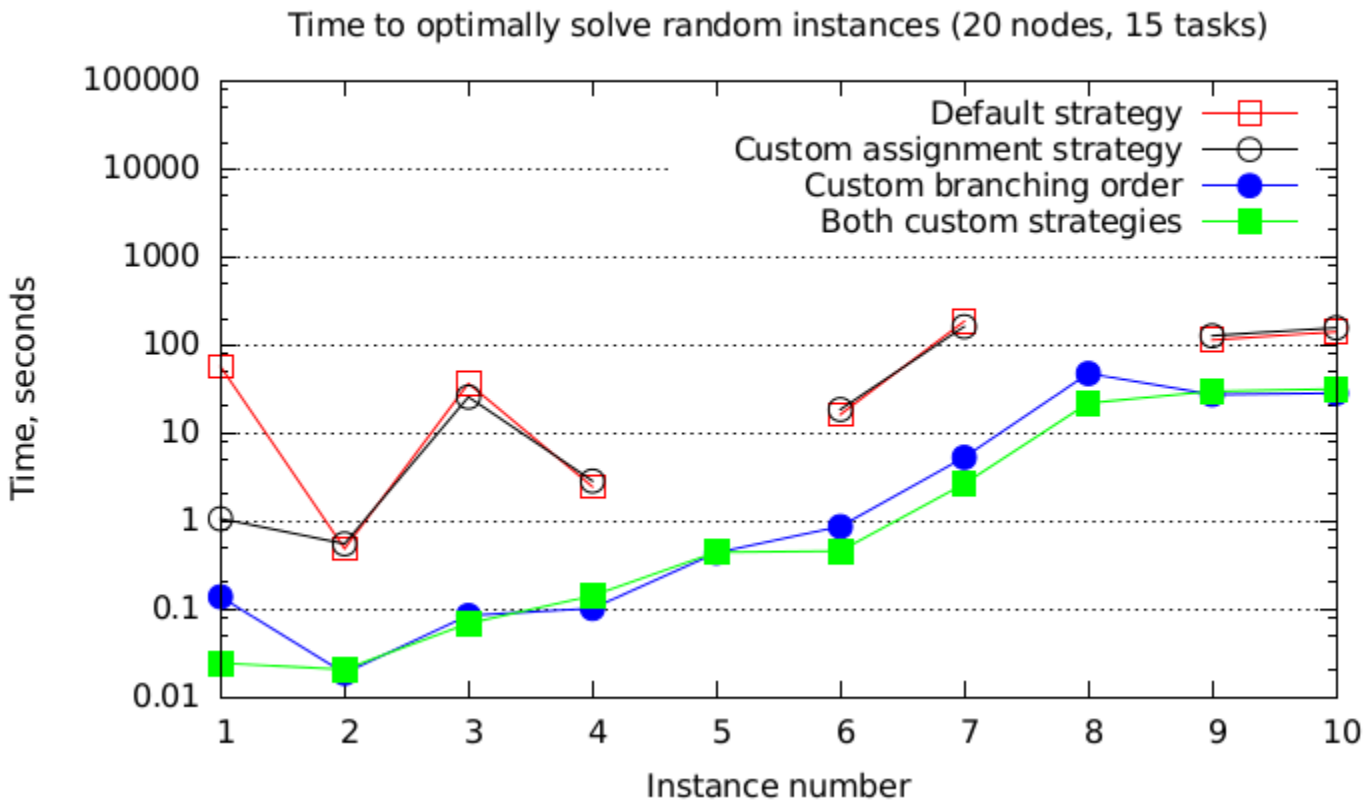


Figure. *Eigenvalue centrality of a graph (red – higher, blue – lower)*

Image source: Wikipedia

Task allocation: results

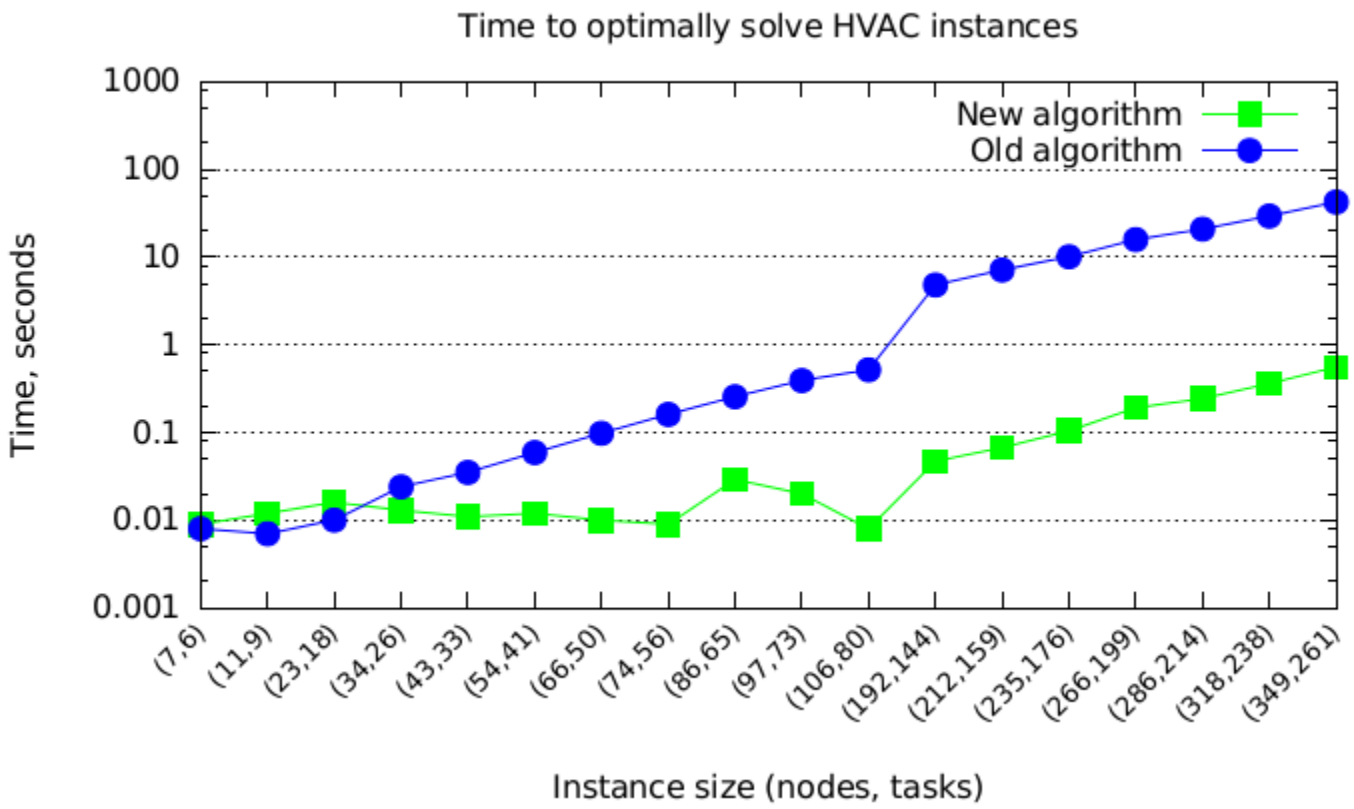
- Randomly generated graphs (same size, same number of connections)
- Experiments:
  - Include centrality metric optimization (as a branching strategy)
  - Include similarity metric optimization (as an assignment strategy)



Task allocation: results

- New algorithm: heuristic estimation based search tree pruning
- Old algorithm: the problem modelled as a Gecode problem

(from *Energy-Efficient Task-Mapping for Data-Driven Sensor Network Macroprogramming Using Constraint Programming*, Farshid Hassani et al., 2011.)



Runtime system

- Communication *within* the sensor network:

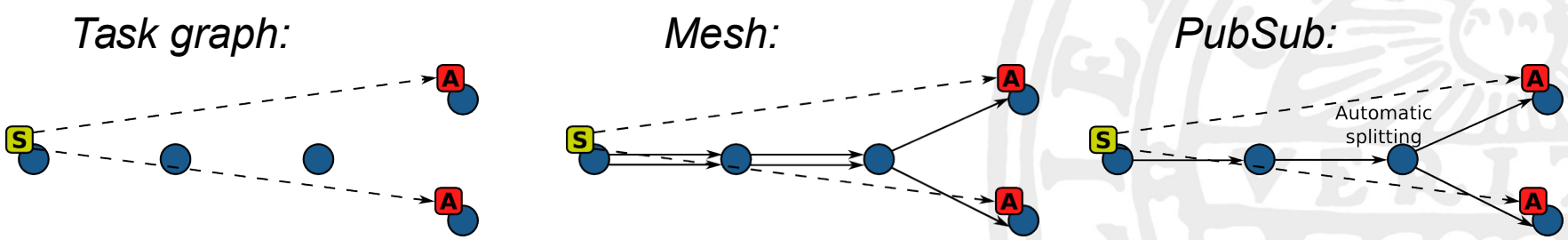
Pattern	Protocol used
Task-to-task communication	*mesh / reliable mesh / publish-subscribe*
Base station to task	-- “ --
Task to base station	*collect*

- Future plans: **Glossy / Low power wireless bus** (for base station ↔ task)
  - uses constructive wireless in intereference to make whole-network broadcasts
  - automatic time synchronization for free
  - a dedicated communication channel (does not interfere with the application)

Task-to-task communication

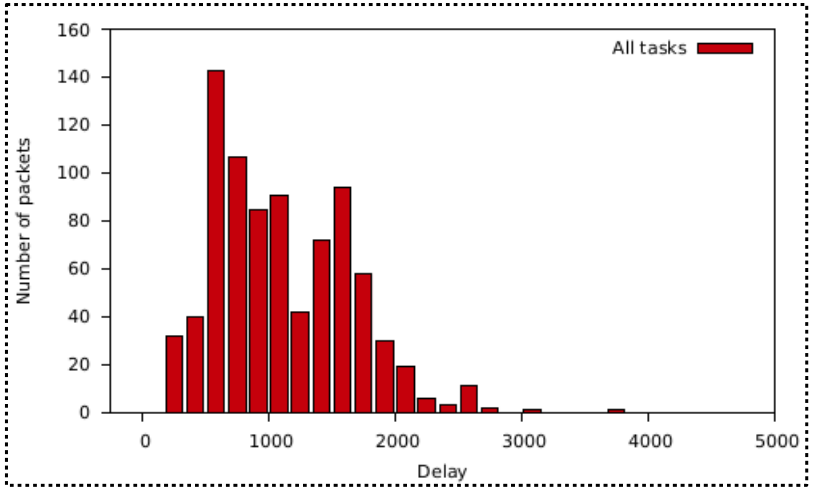
- Problems of the *mesh* protocol:
  - discovers routes to addresses, not data
  - data forwarding is not taskgraph-aware: leads to traffic overhead
  - reactive routing leads to traffic bursts

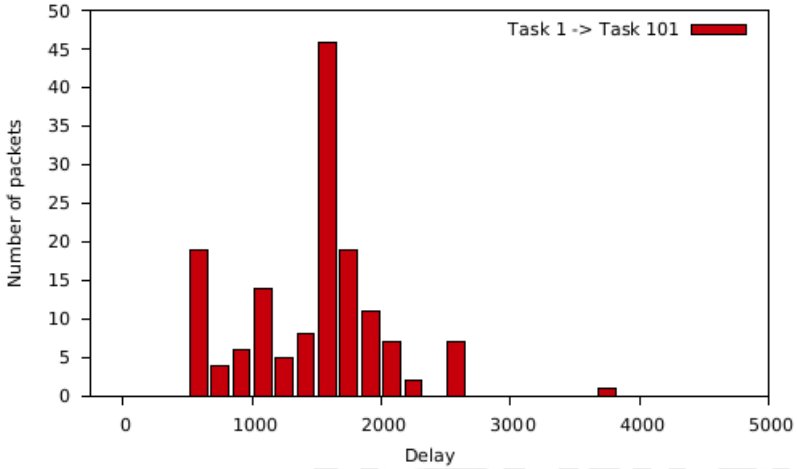
- PubSub:
  - data-centric routing
  - automatically splits traffic to subscribers
  - *(planned)* proactive routing with “slow start” to reduce burstiness

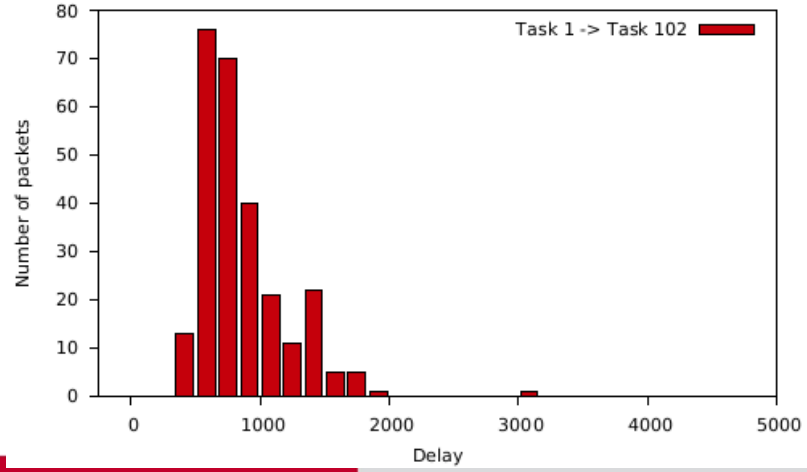


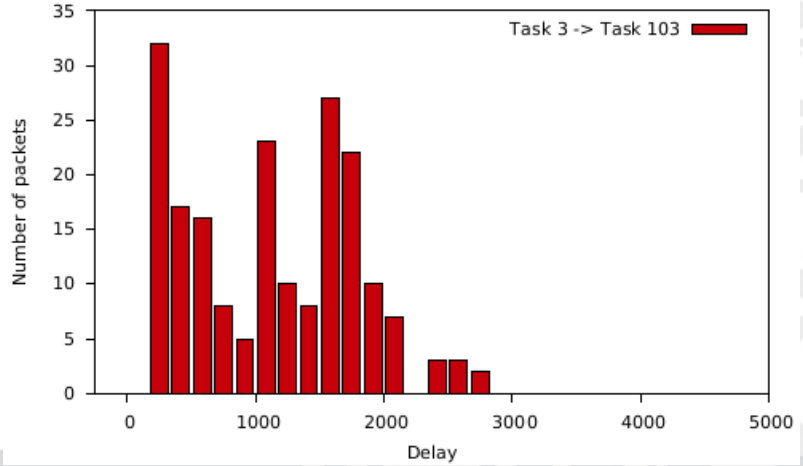
Some ”real” (simulated) delay distributions

20 node network, 6 communicating task pairs, 2 hop paths everywhere









Future directions

- Better run-time feedback integration
  - complete the “allocate → deploy → measure → update models” loop
- Adaptations
  - Some work in this direction in: *A case for node-local runtime parameter adaptation in WSN*, Atis Elsts and Edith Ngai, SNCNW 2014
- Look into higher level languages for definition of tasks
  - SEAL (my PhD work) integration is in progress
- Task allocation optimizations
  - finding the optimal mapping still is time consuming even for average-sized networks

Future directions

- Validation in real networks
  - concrete application ideas?
- Formal modelling + verification of some parts?
- Integration with Internet of Things?
- Security and other missing features?
- Hope for more collaborations!
  - especially for ideas how existing ProFuN work can be integrated in this