Applications
Three sample applications

- Fuzzy inferno
- Nostalgic cow
- Twilight Eden
Fuzzy inferno

Tatoosh wildfire, British Columbia
Fuzzy inferno

- Call for FiFiH drops
Fuzzy inferno

Firefighter Helpers in action
Fuzzy inferno

- I see now. Air ...
  Trucks... Let's go!
Nostalgic cow

Tags & Pegs blueprint.
The devices:
Nostalgic cow

-periodically broadcast
unique identifiers
Nostalgic cow

- record nodes in vicinity, with aggregated timestamps
Nostalgic cow

- form an ad hoc network in a grid of Pegs and mobile Tags
Nostalgic cow

- optionally log sensor data, trigger alarms
Nostalgic cow

- when so configured, route packets to a local sink node
Nostalgic cow

- respond to queries with verbose or summary data
Twilight Eden

Self-monitoring group blueprint.
The devices:
Twilight Eden

-Record input from variety of sensors
Twilight Eden

- Form ad hoc groups with flexible self-monitoring capabilities
Twilight Eden

- Tigger alarms from sensor readings or violations of configured groups

Mr. Brown fell asleep in plain sun
Ms. White and Ms. Black have met to argue again
Twilight Eden

- Provide efficient event notification
Twilight Eden

-Natural extensions, e.g. mobile groups
Twilight Eden

-Always on.
Patterns, features

**Fuzzy inferno:**
Similar to EcoNet, i.e., data collection towards a “sink” (a centralized processing system)

**Nostalgic cow:**
Local in-node data storage (driven by proximity events) + on-demand queries

**Twilight Eden:**
Sink (sinks) + event detection (distributed predicate evaluation), possible actuators
Each of the three examples …

... represents a certain class of applications (which we call a blueprint)

Fuzzy inferno and EcoNet fall into the same basket

What you have seen so far does not illustrate all the features of EcoNet
Two types of nodes

Collectors:
Equipped with sensors, responsible for the actual collection of data

Aggregators:
Providing mesh connectivity within the network, interfacing the collectors to processing systems (OSS)
Collector

sensor ports
Collector interface

- Diverse types of sensors can be attached to different collector nodes.

- Collector nodes can also receive commands from the network, i.e., they are not merely transmitters.

- Those commands may affect data collection cycles; also, some of the “sensors” can be in fact actuators, e.g., to control watering equipment.
Operation
Cluster: the set of collectors covered by one aggregator. Some collectors may fall into the range of multiple aggregators: their cluster membership is determined dynamically and may change.

50-150m max, depending on the environment
The aggregators form an **ad-hoc network** to ensure that all sensor data arrive at the **master**, a dedicated aggregator with OSS interface. Any aggregator can become a master dynamically, if needed.
Multiple masters are possible, with the global data partitioned among them, and/or replicated for increased reliability.
Collectors may also forward data, if required. In this case, the collector extends the range of its aggregator by providing ad-hoc connectivity to out-of-range neighbors.
Collectors know when their data “makes it.” If not, they can store data locally until connectivity is (re)established. For example, you can deploy collectors before the aggregators are available. The collectors will start their job right away.
When connectivity is established, the pending data will be forwarded to the master.
In principle, aggregators …

... are not needed as a separate type of node.

Their presence is dictated by tradeoffs:

- duty cycling, i.e., energy consumption
- storage requirements
- interfaces
Characteristic features of applications

Data sink

- multiple sinks?
- traffic divided or replicated?
- **perfectly** divided or **perfectly** replicated?
- nothing is perfect in WSN
Characteristic features of applications

Storage

- size
- volatility
- reliability
- durability
- power requirements
Characteristic features of applications

Mobility

- are all nodes mobile?
- how fast?
- proximity-based actions
And, of course, the fundamental question is always …

... how much to expect from a node? There are two facets to it:

- given the requirements, how to build the cheapest node meeting them?
- given a particular node architecture, how much can it accomplish?

Various compromises/design tricks are played in order to maximize the “yield” for a given architecture.
Don’t believe people who say …

… that it all can be done with PDA-sized nodes programmed in Java

the keywords are massive, cheap, energy-efficient

massive and cheap imply small

if big becomes cheaper (as technology advances), then small becomes even cheaper, enabling new applications

note that after all these years, trivially small microcontrollers are still doing very well
A case study: BCG HDL
BCG HDL

a “tunnel”
Node structure

Autonomous sample collection upon request from CPP

Samples can be retrieved in near RT over the RF channel
The realm of feasibility

Target sampling rate: \(8 \times 12 \times 500 = 48 \text{ kbps}\)

channels
bits per sample
samples per second

The raw rate of our channel is up to 200 kbs, with 50 kbps being practical.
Advantages of doing it the small way:

One could say: this is for medical applications, so who cares about the cost!

Not strictly “medical”: one can think of cheap personal monitors, e.g., for joggers ...

Who said that medical MUST be expensive?

Most importantly: avoids RF pollution! It makes sense to accomplish the task with the minimum requirements regarding the energy, range, and bandwidth
Another example: smart badges

We are currently working with an industrial partner on such an application (Włodek’s show and tell)

Again, what appears to be a PDA-caliber solution was made to fit a tiny node

This one even includes a graphic LCD with photos and menus
The message is:

We should always try to make it as small and cheap as possible!

Problems:

Can you do this with a reasonable effort (human work counts, too)

Convenience implies overheads (in all areas):

- programming
- networking
- collaboration (distributed processing)
Summary of issues

Networking: the most visible component
- simplicity: must fit the small node
- flexibility: must cater to all traffic patterns
- robustness: must deal with failures

Programming: the developer’s effort
- program size: minimizing the RAM footprint
- convenience: high-level programming tools
- time: rapid development
- resilience: bug-free code, readability, reusability
Summary of issues

Collaboration: the quality of application

- non-contrivance: one node (of a massive WSN) cannot be responsible for too much
  - the node shouldn’t have to store a lot
  - it shouldn’t participate in complicated intrigues where everything depends on it

- this requires a novel approach to distributed programming ...
  - ... including networking, which is just a special case