19th ACM SIGPLAN International Workshop on Erlang

August 23, 2020
Virtual event

Diversity and Inclusion Sponsor

LIGHTKONE

Lightweight computation for networks at the edge

UCL Université catholique de Louvain

Living on the Edge with Erlang

Keynote talk

August 23, 2020

Erlang 2020 Workshop

Peter Van Roy

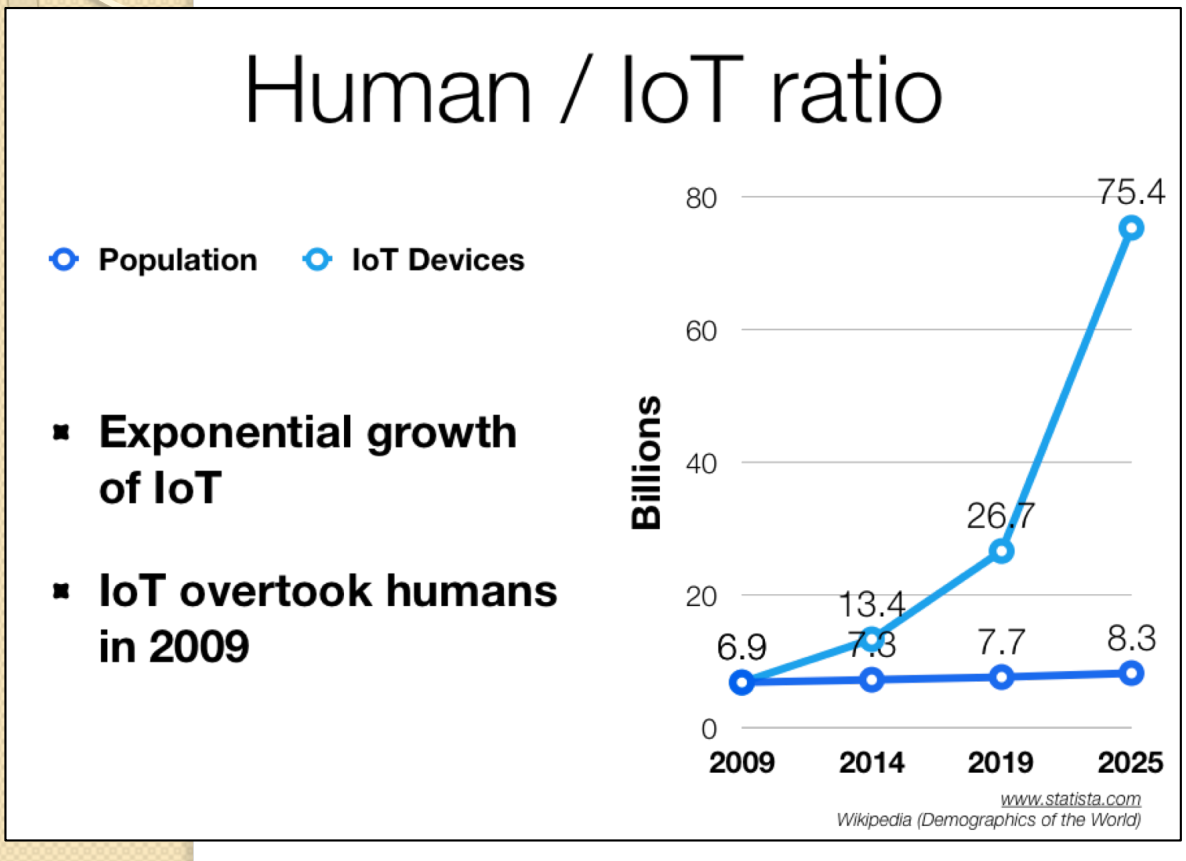
Université catholique de Louvain

Overview

- Exponential growth of IoT
  - Internet of Things: devices at the logical extreme of the Internet (farthest away from the cloud center)
- Applications at the extreme edge
  - Use GRiSP platform and Erlang applications
- Sensor fusion for person tracking
  - Practical results: two moving people in real time
- Big data on the extreme edge
  - Practical results: cheapest possible solution
- Conclusions
  - IoT is just beginning; GRiSP and Erlang are a good platform for anyone who wants to enter this innovative new area

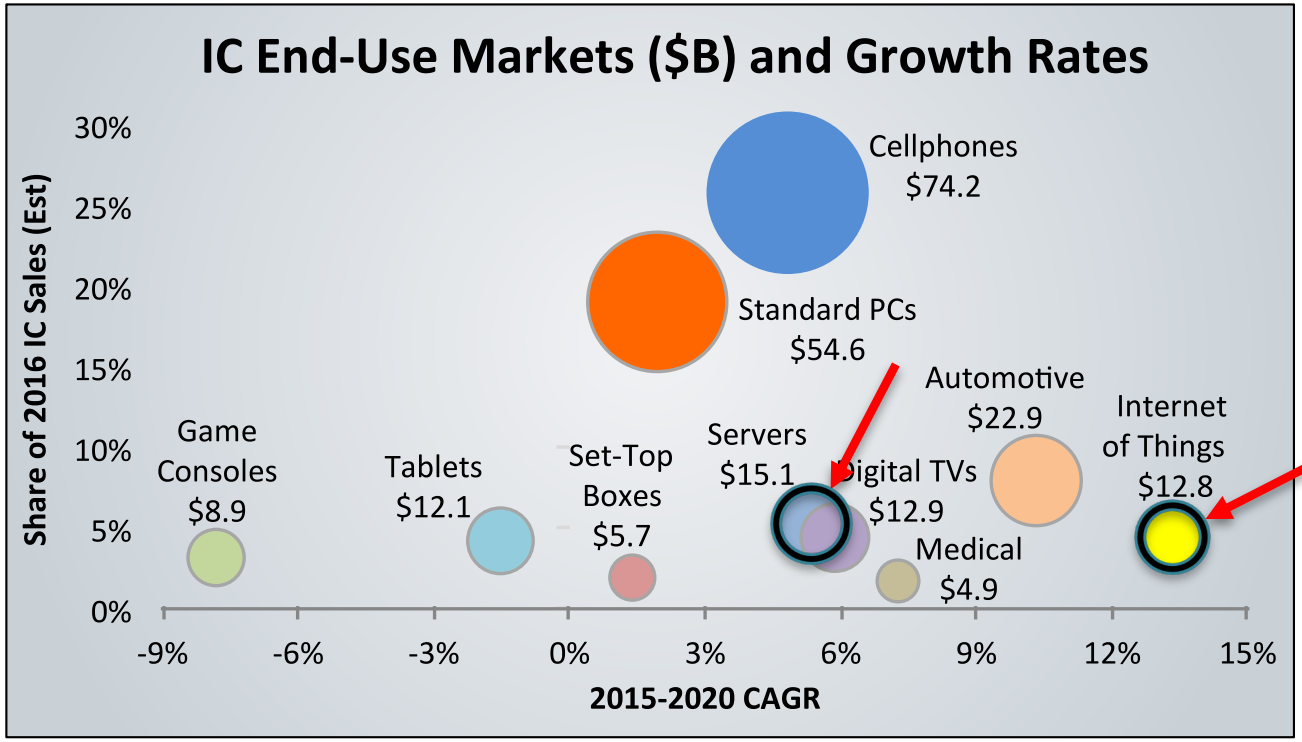
EXPONENTIAL GROWTH OF IOT

IoT: The New Exponential



- IoT (Internet of Things) will have exponential growth for the foreseeable future
  - Moore's Law is fading
  - Mobile phones saturate, but IoT will reach 75B by 2025
- Programmability challenge
  - How do we program this fast-growing, heterogeneous infrastructure?
  - Big companies (GAFA) are all extending their cloud frameworks toward the edge, but is this a final solution?

"The Edge is Becoming the Center"



Holger Pirk, Imperial College, 2020

(CAGR = Compound Annual Growth Rate)

- The IoT market is growing much faster than the cloud: 13%/year versus 5%/year
- Computation must be done closer to the edge instead of in the cloud

APPLICATIONS AT THE EXTREME EDGE

Applications at the extreme edge

- We give two examples of applications that run directly at the extreme edge, i.e., on the sensor boards, with no cloud or computers needed
  - Sensor fusion: accurate real-time person tracking in a room using sonars
  - Big-data computation: MapReduce framework to do big-data-style computations

- We use the GRiSP hardware and software
  - GRiSP provides off-the-shelf support for IoT prototyping using Erlang and Digilent Pmod sensors
  - GRiSP is a complete solution for IoT; no further installations are needed (like, e.g., for Raspberry Pi)

Open-source software

- All software for the two projects presented in this talk is available in open-source repositories
  - All software is written in Erlang/OTP and runs on GRiSP boards

- Lynkia (MapReduce on GRiSP)
  - https://github.com/lynkia

- Hera (Sonar-based person tracking on GRiSP)
  - General sensor fusion framework (independent of GRiSP)
    - https://github.com/guiste10/hera
    - https://github.com/bastinjul/hera_synchronization
  - Sonar-based application for deployment on GRiSP boards:
    - https://github.com/bastinjul/sensor_fusion
  - LiveView application for real-time visualization
    - https://github.com/bastinjul/sensor_fusion_live_view

Foundation of this talk (1)

- Two master's projects done at UCL in 2019-2020, supervised by Peter Van Roy and Igor Kopestenski
  - Both available at https://www.info.ucl.ac.be/~pvr/pldc.html

- Guillaume Neirinckx and Julien Bastin. "Sensor Fusion at the Extreme Edge of an Internet of Things Network". Master's project, Université catholique de Louvain, Aug. 2020.

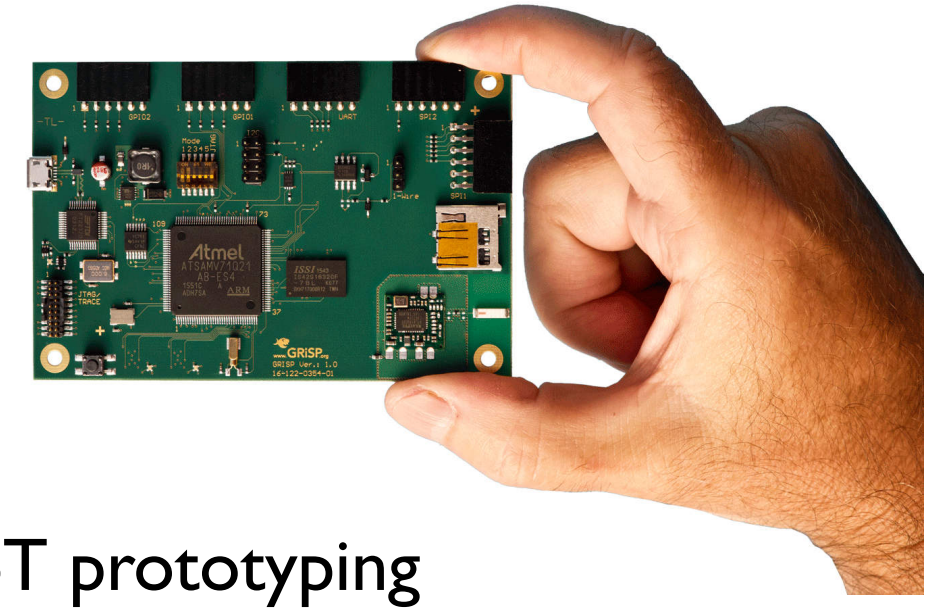
- Julien Banken and Nicolas Xanthos. "Doing Large-Scale Computations on an Internet of Things Network". Master's project, Université catholique de Louvain, Aug. 2020.

Foundation of this talk (2)

- GRiSP 1 platform for building embedded systems with Erlang & Elixir, developed by Stritzinger GmbH.
  See www.grisp.org.

- LightKone H2020 project (2017-2019): Lightweight computation for networks at the edge.
  See www.lightkone.eu.

GRiSP platform

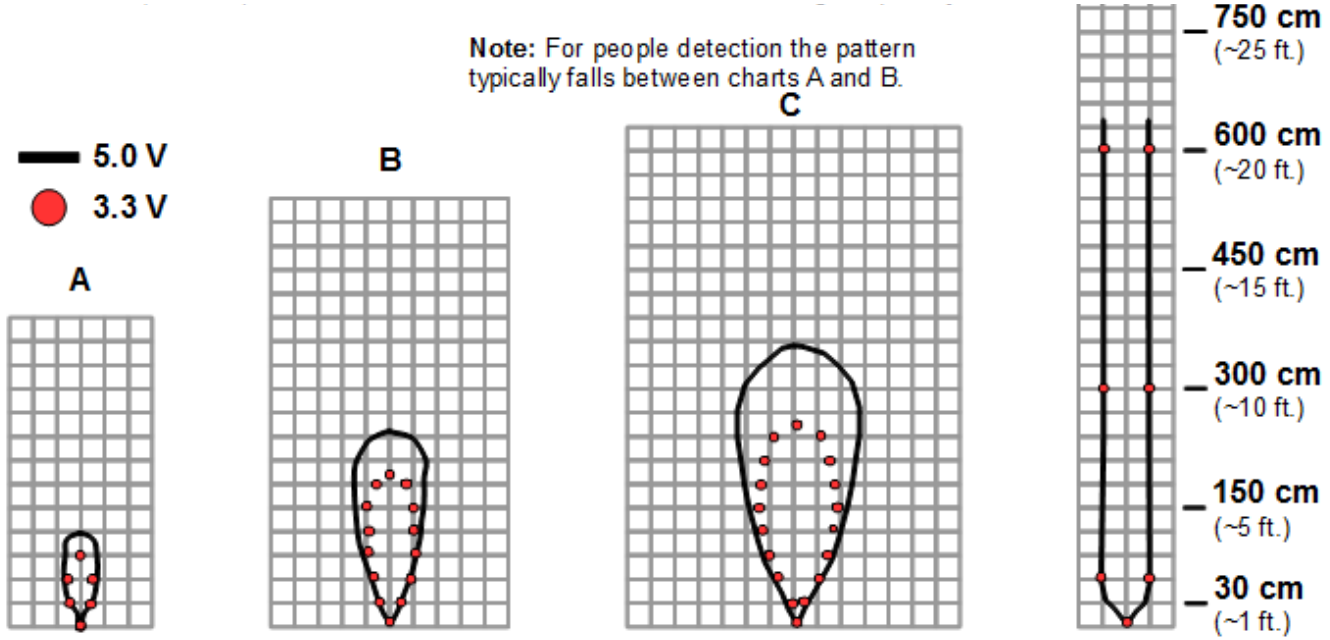


- GRiSP 1 platform for IoT prototyping developed by Stritzinger GmbH
- Hardware: *"Top-of-the-line laptop year 2000"*
  - ARM Cortex M7 at 300 MHz with FPU, 64MB RAM, MicroSD Card "disk", Wi-Fi support
  - Five Digilent Pmod interfaces, one I2C interface, one Dallas 1-Wire interface
- Software
  - RTEMS real-time operating system
  - Full Erlang/OTP supporting Erlang and Elixir

Pmod MaxSonar sensor

20mm x 22mm x 15.5mm

- This card uses the MB101 LV-MaxSonar-EZ1 sonar range finder
- 1inch resolution, 50ms reading time
- People detection claimed up to 2.4m / 8ft
  - We measure it as somewhat less (see later!)
- Beam characteristic diagrams from datasheet
  - Dowels: A: 0.25inch, B: 1inch, C:3.5inch, D:11inch

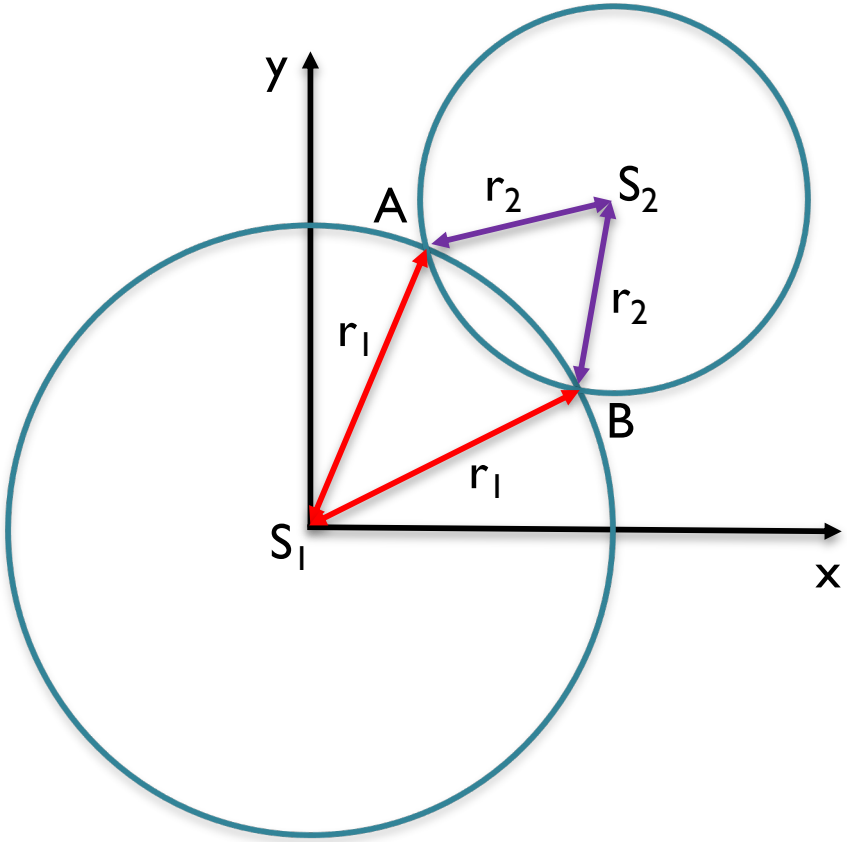


SENSOR FUSION FOR PERSON TRACKING

General approach

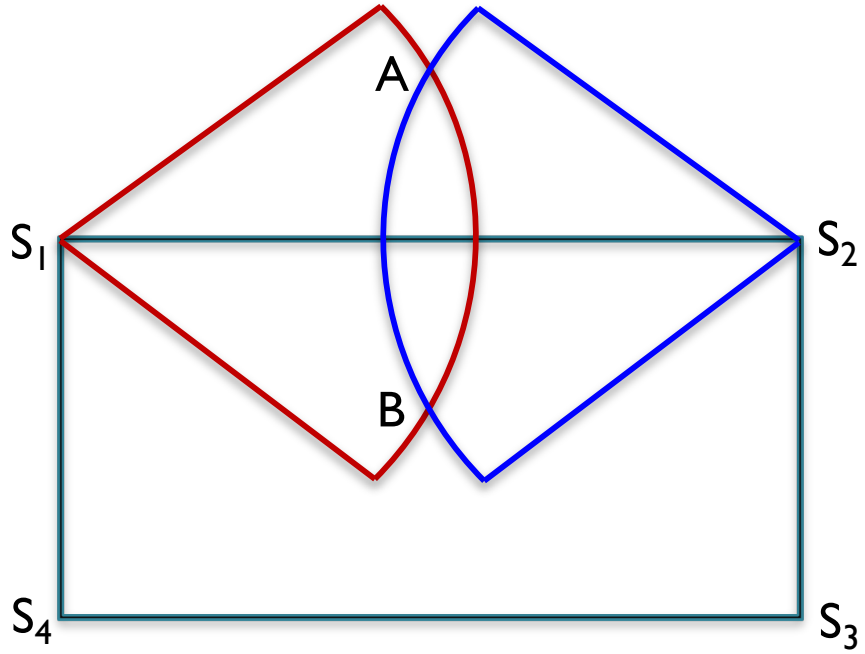
- The system consists of n sonar-equipped GRiSP nodes installed along the walls of a room
- Sonar measurements are done as frequently as possible (limited by the MaxSonar sensor)
- Each node receives all information from all sonars to compute position of static or moving targets
- This general approach is made practical with the following refinements:
  - Trilateration: compute positions of up to 2 people with 4 sonars
  - Noisy sonars: handle noise and incorrect sonar operation
  - Crosstalk: handle interference between sonars in each other's view
  - UDP multicast: needed for real-time operation
  - Live view: support for real-time graphical representation
  - Resilience: implemented using a supervisor tree

Trilateration: one target



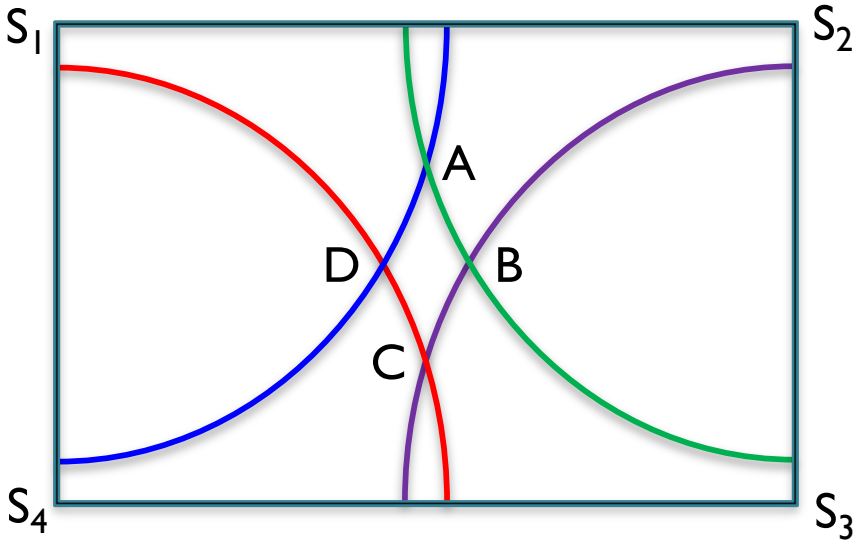
- The target position is computed in 2-dimensional space from two sonars
- For two sonars S_1 and S_2 there are two possible target positions A and B
- To remove ambiguity, three sonars are necessary in theory, although we find that in practice two suffice

Trilateration: two targets



- We place four sonars in a rectangle
- Because of room constraints, two sonars are sufficient to determine the position of one person. We add a position filter to implement these constraints: A is removed by the position filter.
- In most cases (but not all), four sonars suffice to determine the positions of two people

Trilateration: limits and filters



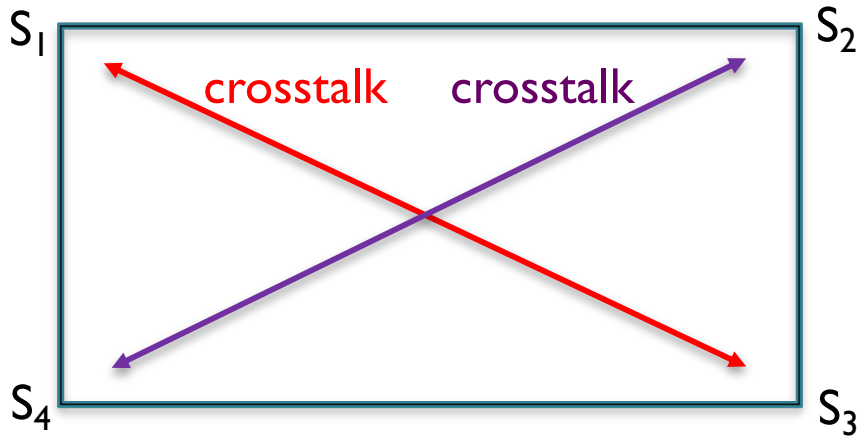
- Four sonars are not always sufficient for two persons
- This example shows two possible positions for two persons in the room
  - A & C is one possibility and B & D is another possibility!
- This happens when the two persons are both near the center of the room
- We solve this problem with a speed filter, which gives a maximum speed of a person. If a position violates this speed limit, it is filtered.
- With the position filter and speed filter, four sonars suffice for two persons

Noisy sonars

- The sonars are not 100% accurate
  - They are noisy (they have jitter) and they occasionally generate completely false measurements

- To handle both inaccurate sonars and computation ambiguities, the system has a general filter module that allows to install multiple filters
  - Generic filter: upper bound on difference in successive measured values (meters, ms)

Crosstalk



- When two sonars are in each other's line of sight, we observe that they interfere with each other and give incorrect measurements
  - S_1 and S_3 interfere, also S_2 and S_4 interfere
- To solve this problem, the sonars need to coordinate
  - We add a synchronization server to activate the sonars in alternation
  - In collaboration with Stritzinger GmbH, we extended the GRiSP MaxSonar driver to allow controlling when measurements are done (the default driver gives only the latest sonar reading)

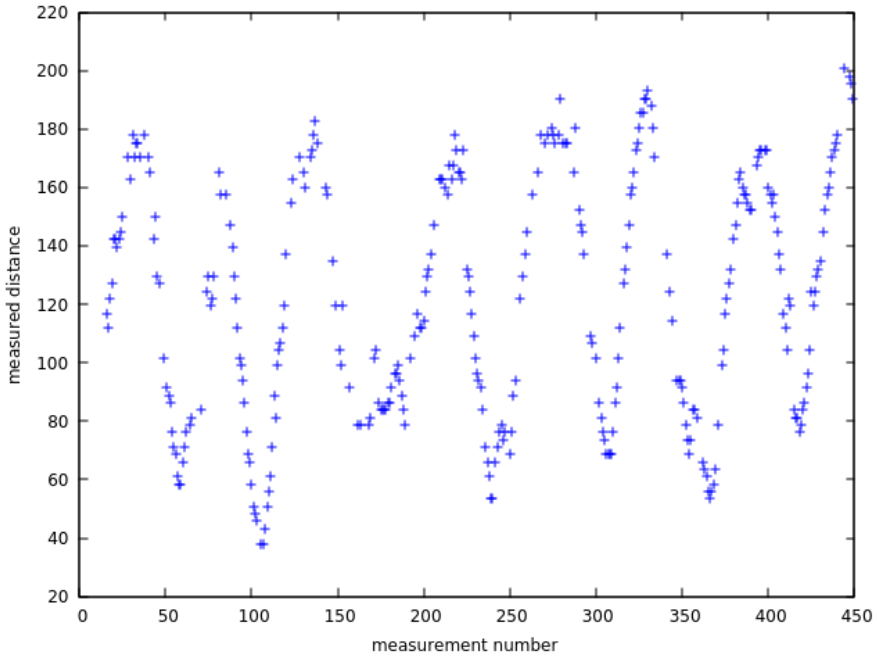
RESULTS FOR PERSON TRACKING

Maximum distance

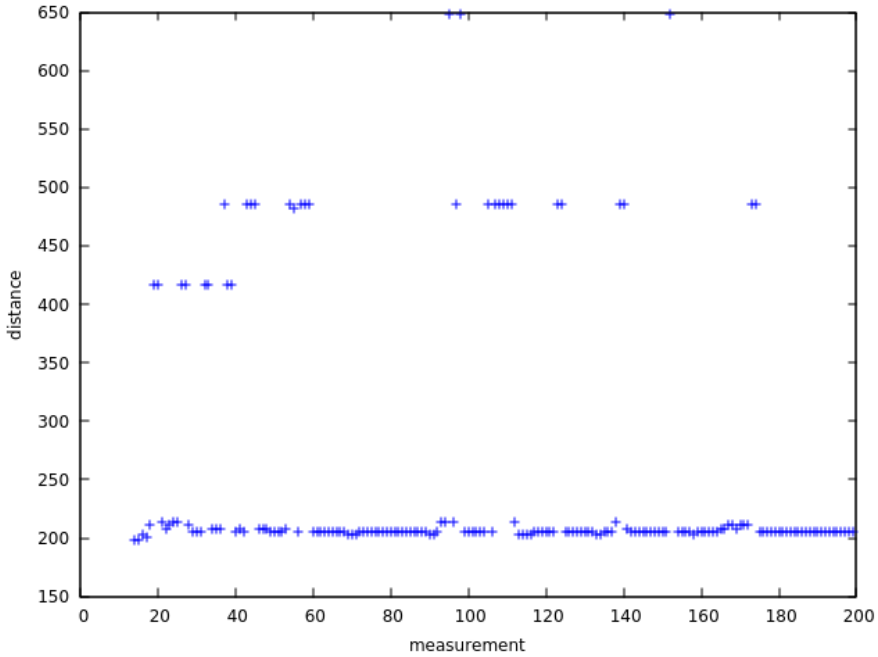
	150 cm	200 cm	250 cm
Front	75%	68%	23%
Side	52%	30%	6%

- We measured one sonar's accuracy for people detection for a stationary person in two orientations (front toward sonar and side toward sonar)
  - We make a series of measurements and compute the fraction of the measurements that are accurate
- For all orientations, it looks like 200 cm is a practical limit
  - The system works in rooms up to perhaps 4m x 4m, not more
- The MaxSonar datasheet gives a limit of 244 cm; this is technically correct but too optimistic for accuracy

Distance to a person (one sonar)



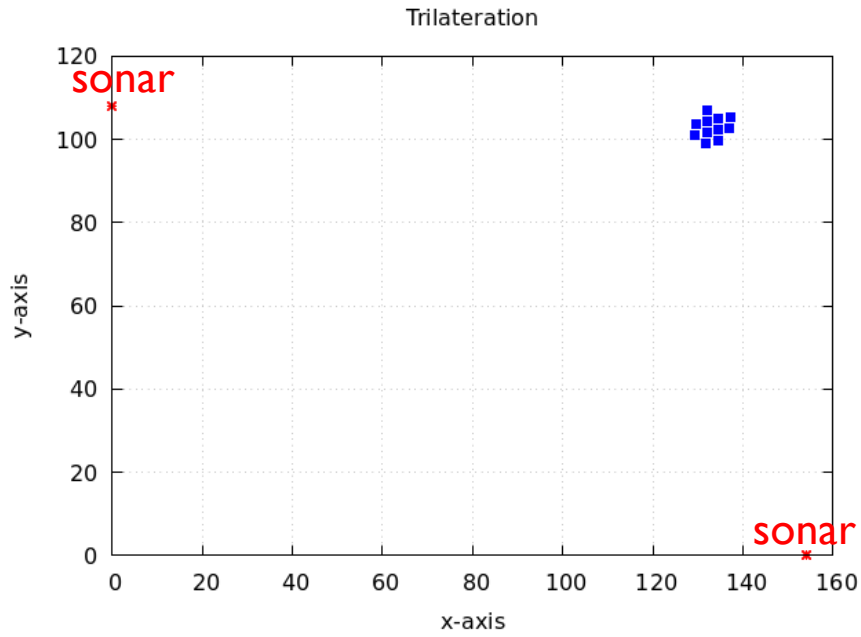
(a) Moving randomly - Obtained with gnuplot



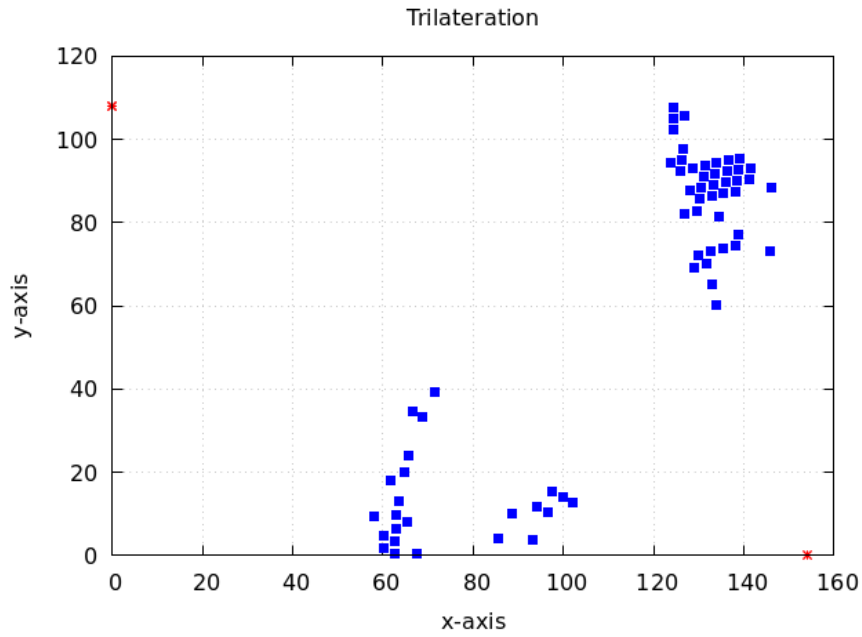
(b) Standing at 200 cm - Obtained with gnuplot

- We show the distance measurements with one sonar for a moving and stationary person
  - 50ms per measurement; 100 measurements is 5 seconds
- Note that the sonar occasionally misses a stationary person

Crosstalk avoidance (2 sonars)



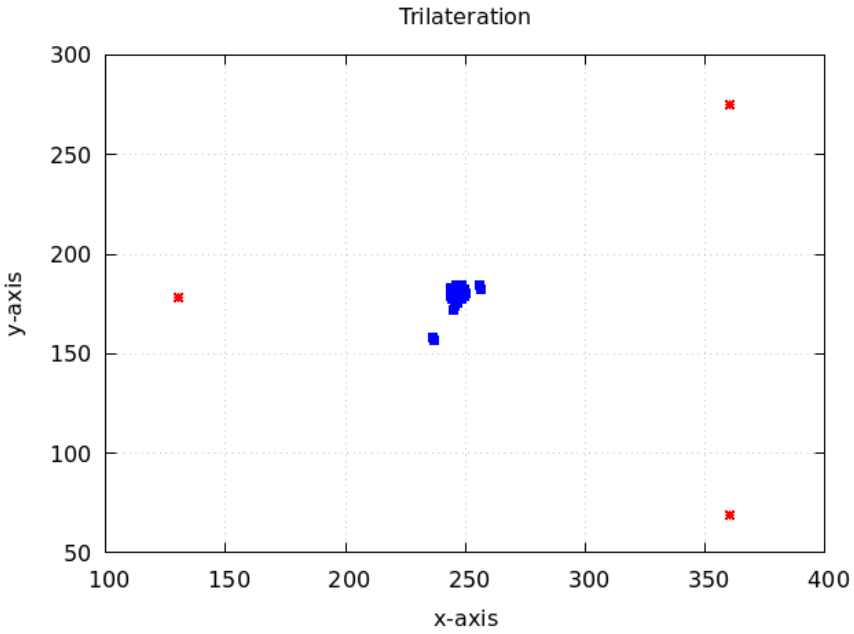
(a) With filter and synchronization

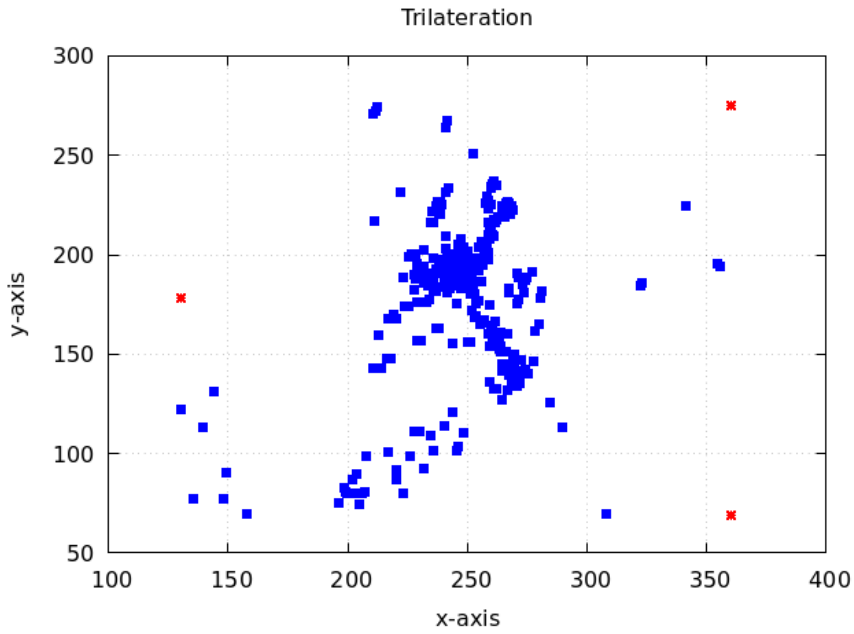


(b) Without filter and synchronization

- This shows the effect of synchronization to remove crosstalk in the case of two perpendicular sonars and one stationary person (with sonars facing each other it is even worse)
- We can see that crosstalk avoidance is essential: without it, no reasonable results can be obtained

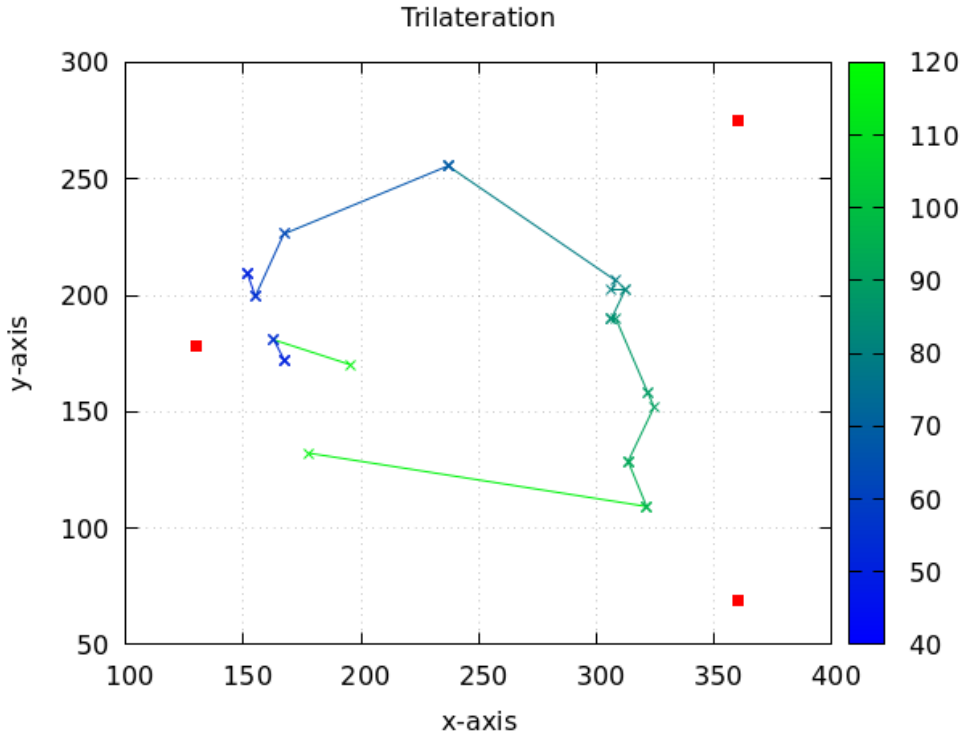
Crosstalk avoidance (3 sonars)





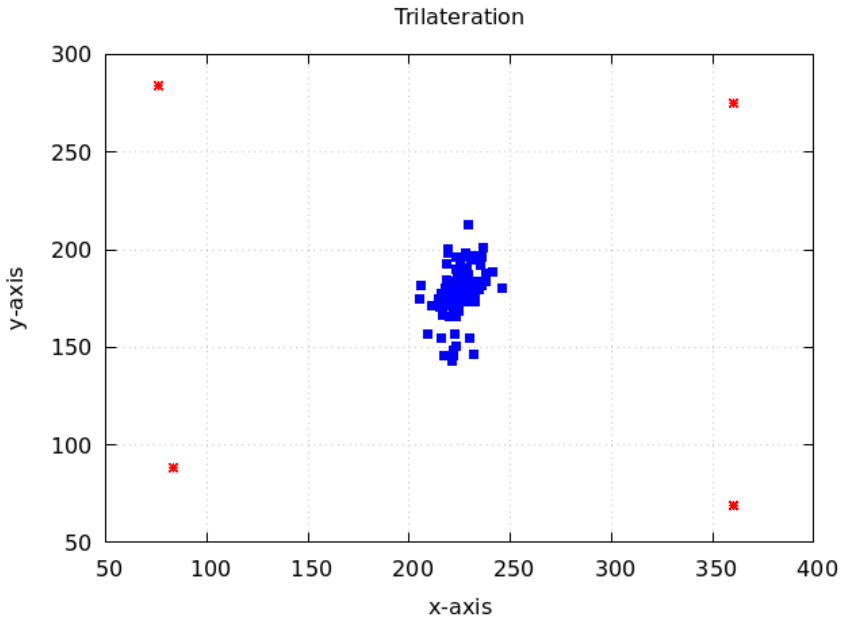
- We show the effect of crosstalk avoidance in a room with 3 sonars in a triangle (red dots show the sonars)
- Both diagrams show a stationary person
- Left diagram is with synchronization and right without synchronization; both diagrams without filter

Moving person (3 sonars)

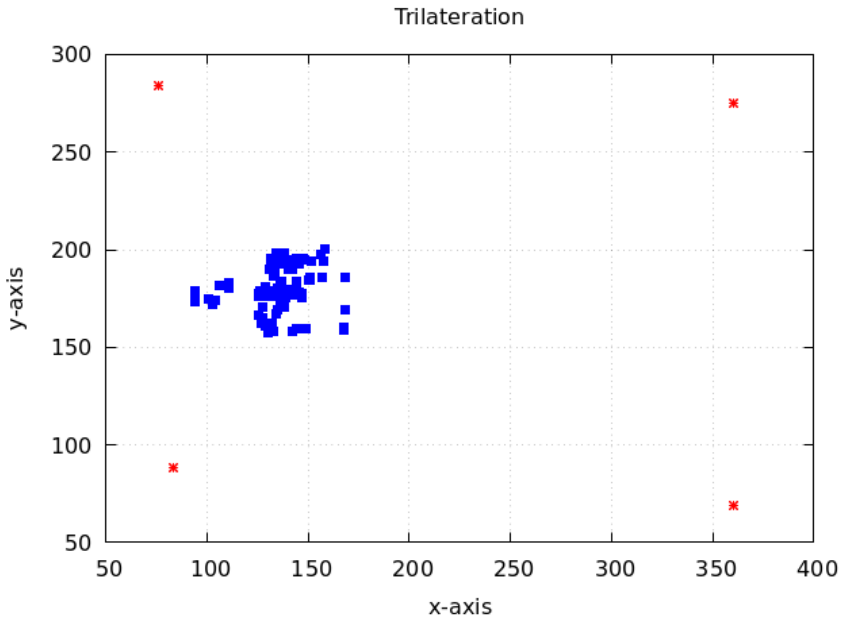


- We show the computed trajectory of a person moving in a circle in a room with three sonars (color shows time, from blue to green, 50ms/iteration, 20 iterations is one second)
- This is computed with both filter and synchronization

Stationary person (4 sonars)



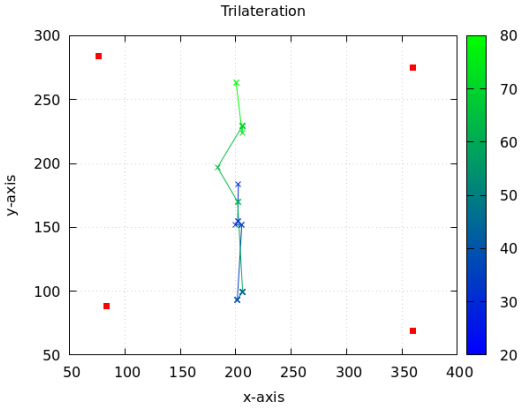
(a) Person on the center of the room



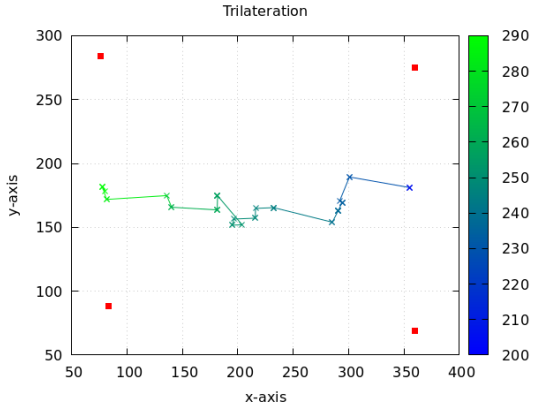
(b) Person on the left side of the room

- This shows a stationary person in two positions, using 4 sonars and synchronization
- The error is higher because of the larger distance: this starts to show the limitations of the sonar

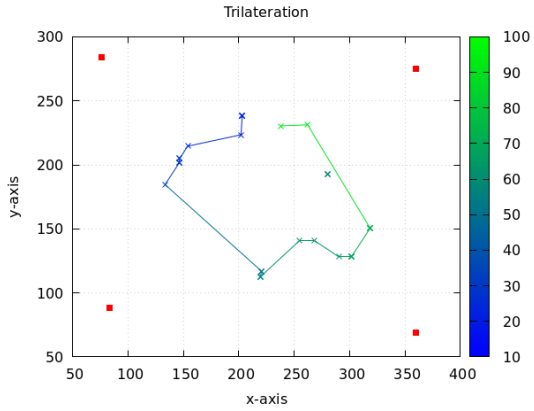
Moving person (4 sonars)



(a) Movement from top to bottom



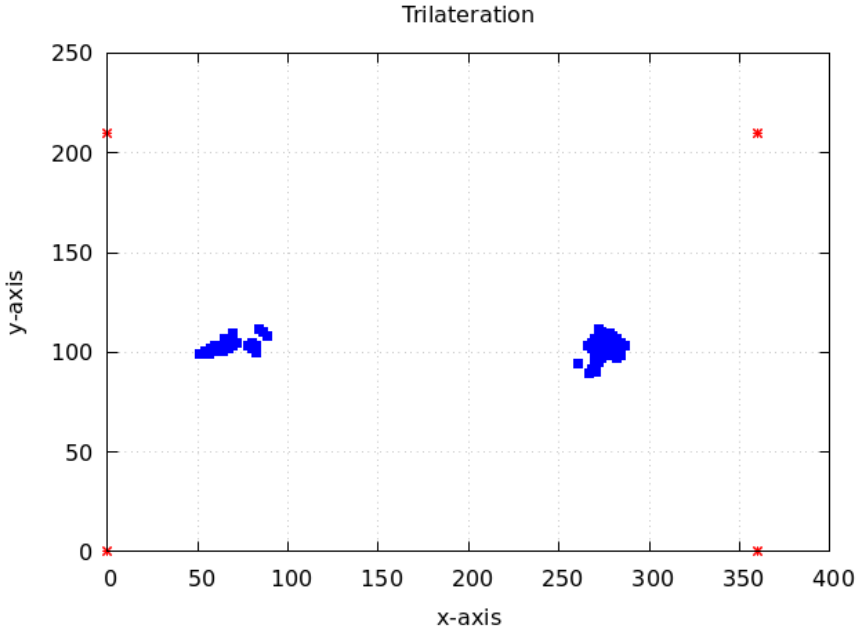
(b) Movement from left to right



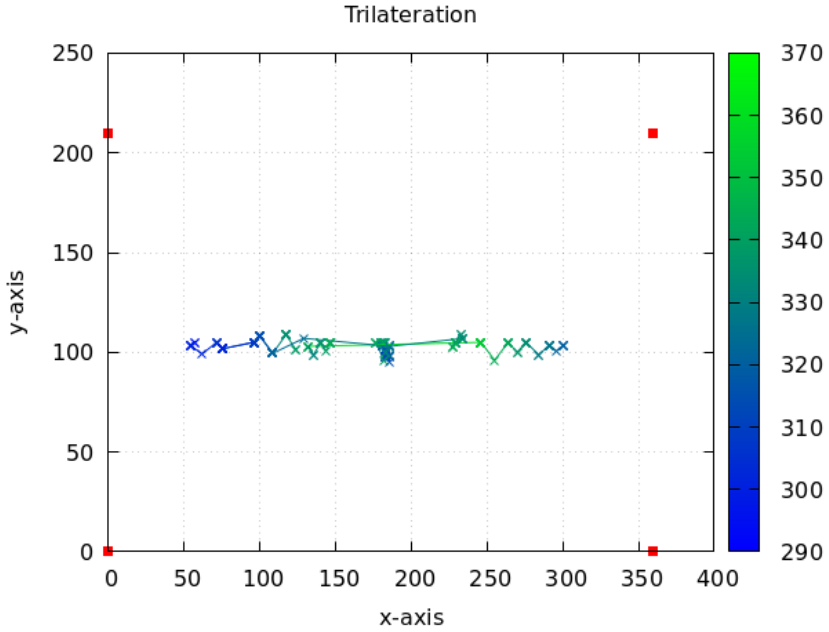
(c) Person describing a circle

- We give three cases of a moving person in a room with four sonars

Two persons (4 sonars)



(a) Two non-moving persons away from each other



(b) Two moving persons synchronously walking to the center

- Our final result is two persons in a room with four sonars, in the stationary and moving cases
- We can see it works pretty well, but it's clear we are close to the limit of the abilities of the sonars

Person-tracking conclusions

- Real-time person tracking works well for one and two persons, stationary or moving, with four sonars in a room up to 3m x 4m
- Four GRiSP boards and Hera software is a practical way to track two people in normal-sized rooms
- The main limitations come from the MaxSonar sensors themselves, but they are good enough for remarkable results
- Crosstalk avoidance is essential
- Live view of the positions is very useful
- For observing more people, more sonars and/or smarter software are needed!

Future work

- Take advantage of improved sonars to increase precision and maximum size of room
- Investigate how to increase number of tracked people with limited number of sonars (using filters, aggregate trajectories, and other information)
- Extend the sonar-based sensor fusion by adding other input
  - Observe people with video cameras and microphones
  - Allow people to carry GRiSP boards, for example, with PmodNav sensors (giving position and acceleration)
- Extend to 3D, for example to do surveillance
- Simplify the initialization by automatically determining sonar positions
- Improve the intelligence by using machine learning
- Future sensor boards will be more powerful: target future improved hardware (GRiSP 2!)

(short video of person tracking)

BIG DATA ON THE EXTREME EDGE

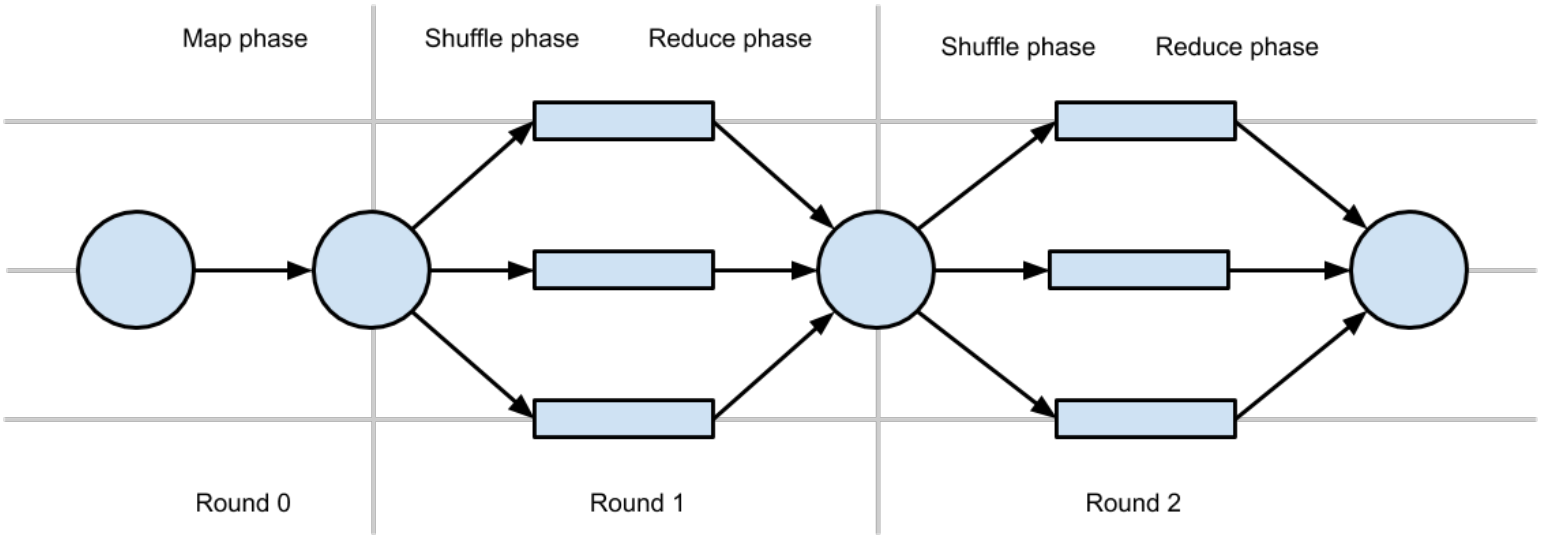
Motivation

- Internet of Things is growing faster than cloud, so computation needs to move out of the cloud and toward the edge
- One possibility is to use local computing nodes near the edge, but this is expensive because of the cost of the node
- The least expensive solution would be to use the sensor nodes themselves to do the computation (this is called the extreme edge)
- In this project we did just that: we made an experiment to see whether big-data-style computations can be done on the extreme edge

Big data on the extreme edge

- We made a proof of concept to do big-data computations on the extreme edge
- Lynkia open-source library
  - Provides a MapReduce computation model with task management, running on a network of GRiSP nodes
  - Built for the hostility of an extreme edge environment: tolerates node crashes, network partitions, and various communication problems
- Evaluation
  - Reliability test with fault injection using different network topologies
  - Load balancing for multiple nodes
  - Benchmarks for computation-intensive tasks

MapReduce capsule summary



- MapReduce is a primitive operation that allows performing massively parallel computation in a distributed environment
  - It was the first important big-data operation in the cloud, invented in the 1990s, later generalized and used as inspiration for many extensions in today's big-data frameworks
- Computations consist of three phases: Map, Shuffle, and Reduce
  - Map and Reduce functions are specified by the user
  - The phases are repeated until a fixpoint is obtained

Lynkia requirements

- Lynkia is designed with respect to the constraints of the extreme edge
- The extreme edge imposes strong requirements:
  - Tolerance of reduced memory size to stay within limited memory of the nodes
    - Extreme edge nodes have limited resources
  - Tolerance of churn, where nodes can leave and join the network at any time
    - Extreme edge nodes can crash (e.g., power loss)
  - Tolerance of unreliable networks, with message loss, delay, reordering, or duplication
    - Extreme edge networks live in hostile environments

Lynkia architecture

- Lynkia library consists of two parts:
  - Task model: Work is divided into "tasks", which can be distributed across nodes
  - MapReduce: A big-data computation that uses the task model
- Lynkia is designed for the extreme edge
  - It is fault tolerant and partition tolerant
  - It tolerates message loss, delay, reordering, and duplication
- Lynkia is built on top of two underlying libraries
  - Partisan: a communication library that uses hybrid gossip to keep connectivity despite node churn (node fails and joins)
  - Lasp: a replicated key/value store that is based on CRDT data structures to guarantee consistency despite message loss, delay, reordering, and duplication

Partisan and Lasp software is designed and written by Christopher Meiklejohn and others in the context of the SyncFree and LightKone European projects

SyncFree

LIGHTKONE

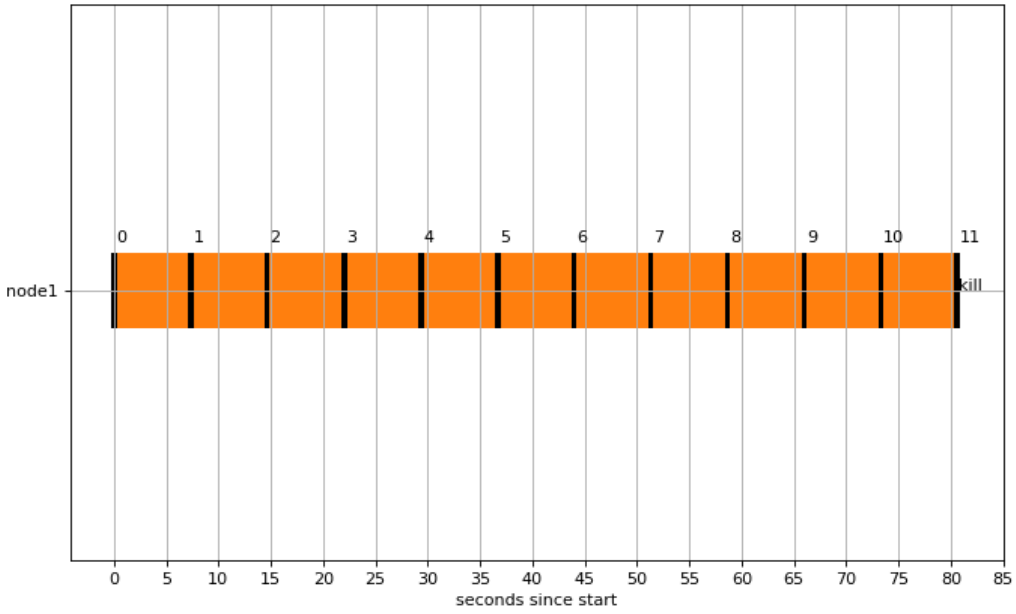
Lightweight computation for networks at the edge

Algorithm operation

- Data is stored at all nodes for redundancy
  - This limits the size of the MapReduce computation but enables reliability
- Nodes have two roles, leader and observer
  - In normal operation there is one leader and others are observers
  - Because of unreliability, there may temporarily be none or more than one leader; in that case the system converges to one leader
- Each computation round has three phases
  - During the map phase, the leader broadcasts checkpoint information to all observers for reliability
  - During the shuffle phase, the leader splits the data into batches and creates tasks for computing on each batch
  - During the reduce phase, the leader waits for all results, and either continues to the next round or terminates
- Task distribution
  - The task model manages task distribution according to each node's load, to optimize performance and balance load

RESULTS FOR MAPREDUCE

Single node test (base case)

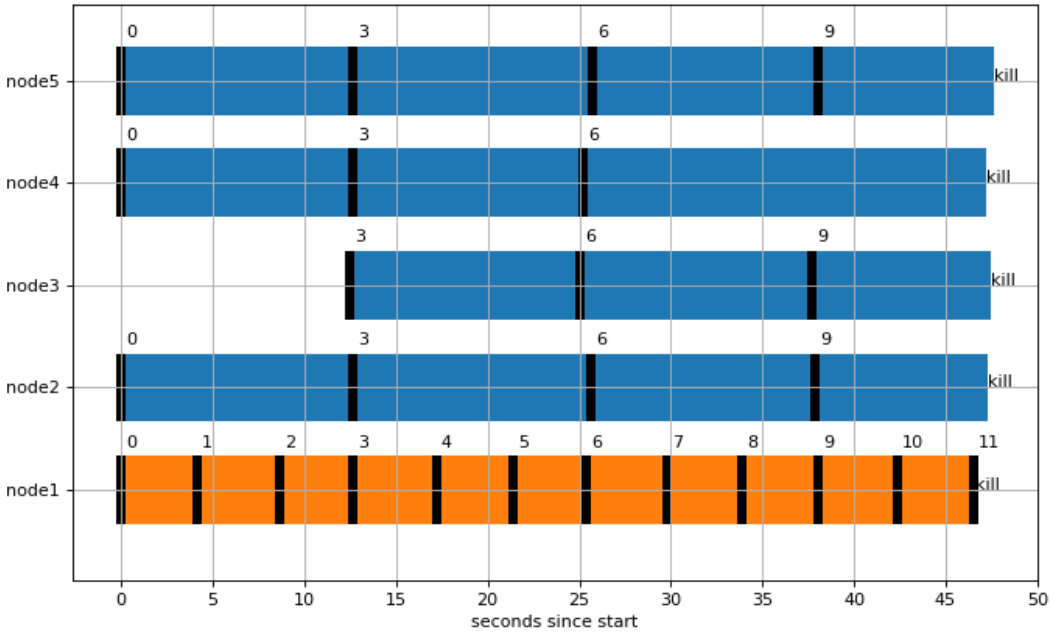


Notation:

- Orange: leader
- Blue: observer
- Line: end of round
- Kill: successful end

- The system is tested in the base case of one GRiSP node with no network
  - Synthetic benchmark that requires 12 rounds
  - The node becomes leader and executes all the rounds

Five node test (no crashes)

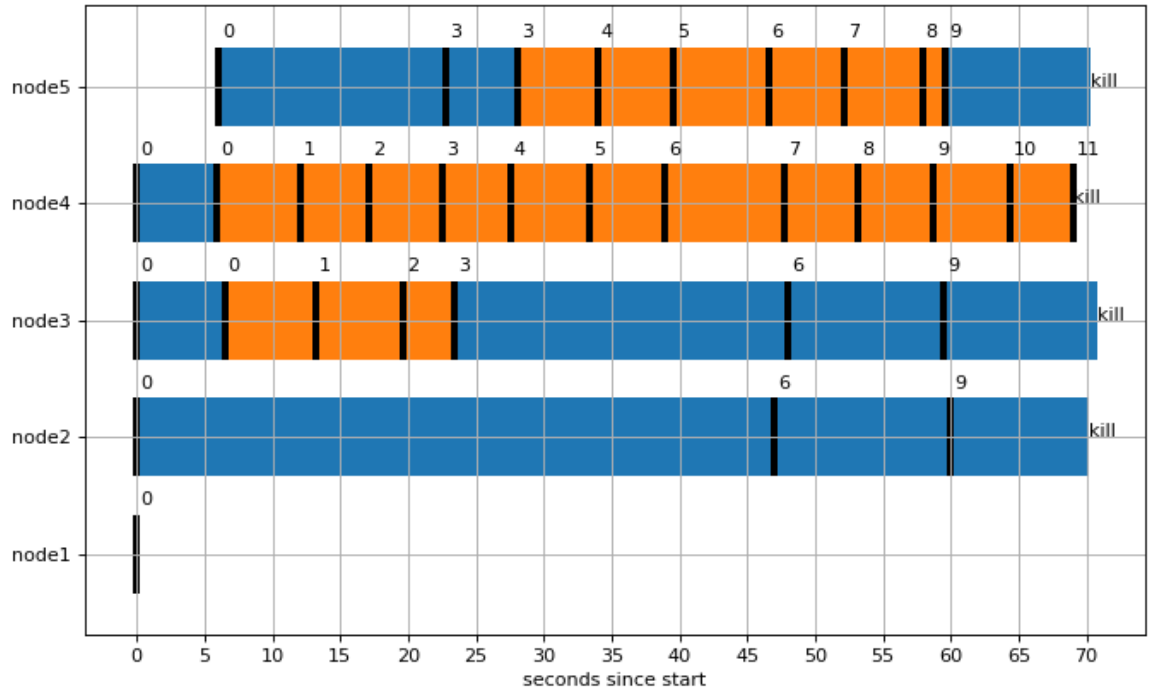


Notation:

- Orange: leader
- Blue: observer
- Line: end of round
- Kill: successful end

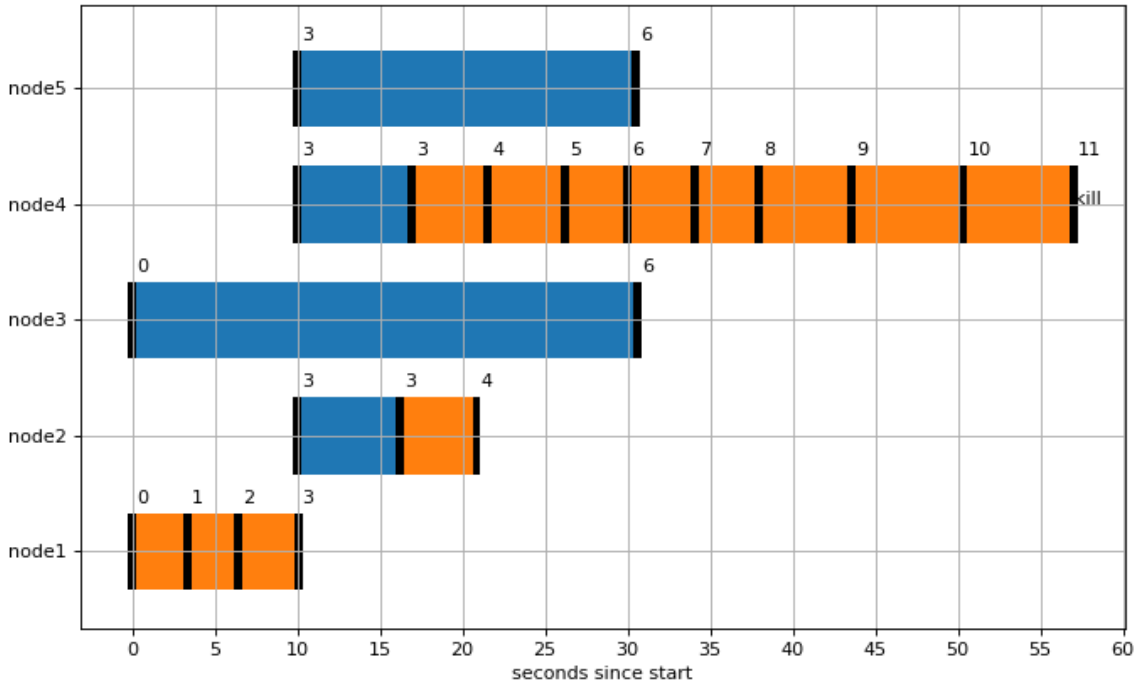
- The system is tested with five GRiSP nodes in a full mesh topology
  - Note that some checkpoint messages were lost or delayed (node 3 in round 0, node 4 in round 9), but redundancy ensures that all nodes reach the final result

Five node test (leader crash)



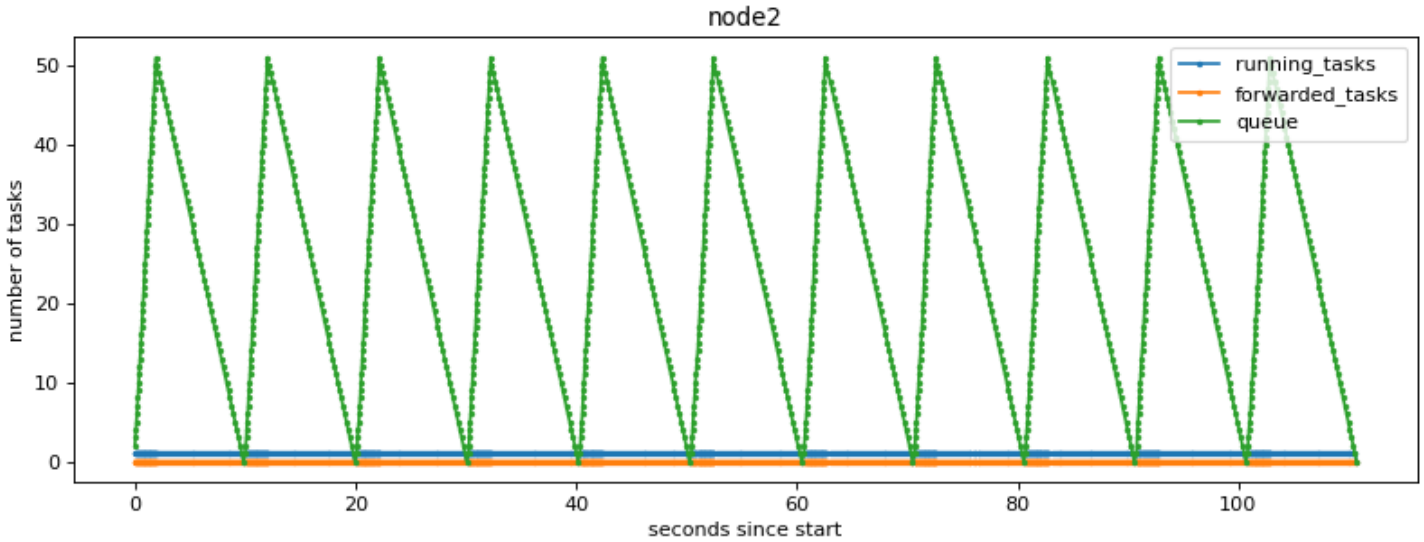
- The leader (node 1) crashes just after broadcasting the initial data
  - Nodes 3 and 4 time out and become leaders, then they compete and node 3 retires because node 4 is more advanced
  - Node 5 gets a time out (because of delayed message) and becomes leader, then it retires because node 4 is more advanced
- At the end, all correct nodes reach the final result

Five node test (4 nodes crash)



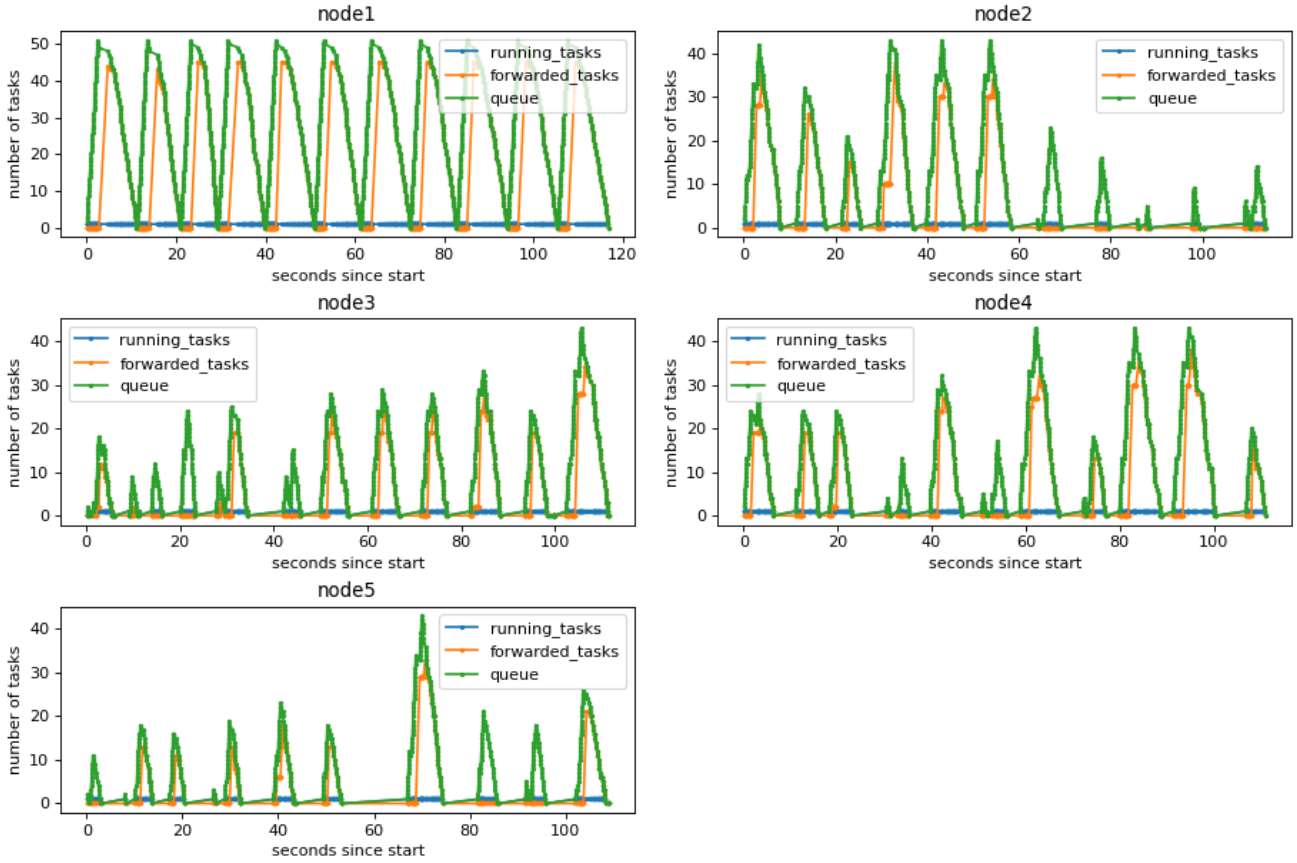
- Purpose of this test is to show that the system is resilient to n-1 crashes
  - At the end, node 4 is the only correct node
- Correct operation with temporary network partitioning was also tested

Task model test (one node)



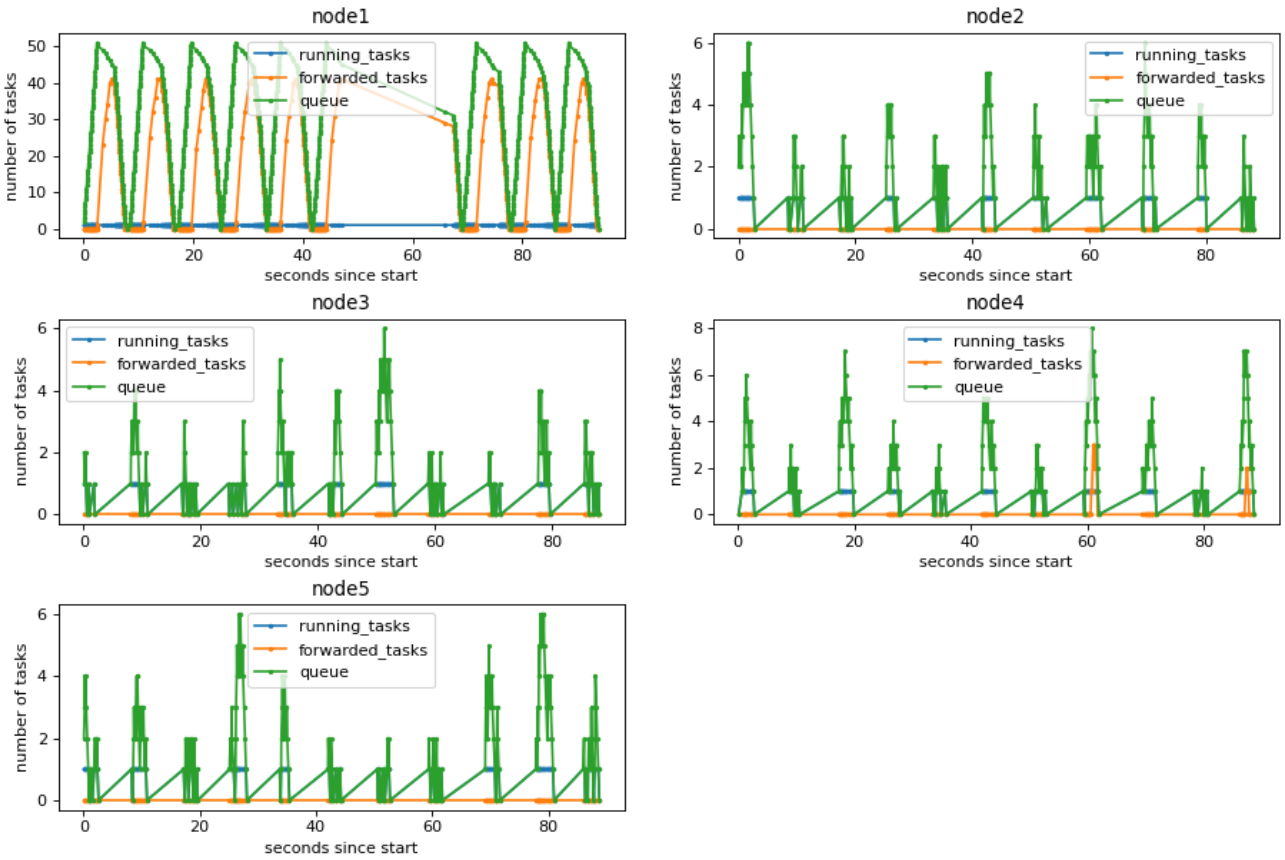
- We analyze the state of the task model: the number of tasks as function of time
  - We use the synthetic benchmark as before, which generates 50 tasks per round
  - Green line gives number of tasks in the run queue
  - Each round is one sawtooth pattern

Load balancing test (pull strategy)



- We test the pull strategy on 5 nodes, where nodes steal tasks from their neighbors
- Note that workload is reasonably well distributed

Load balancing test (push strategy)



- We test the push strategy on 5 nodes, where the leader forwards tasks to its neighbors
- Node 1 has unusual behavior between 40 and 70 seconds, probably due to overloading

Computation test (pull vs. push)

Pull

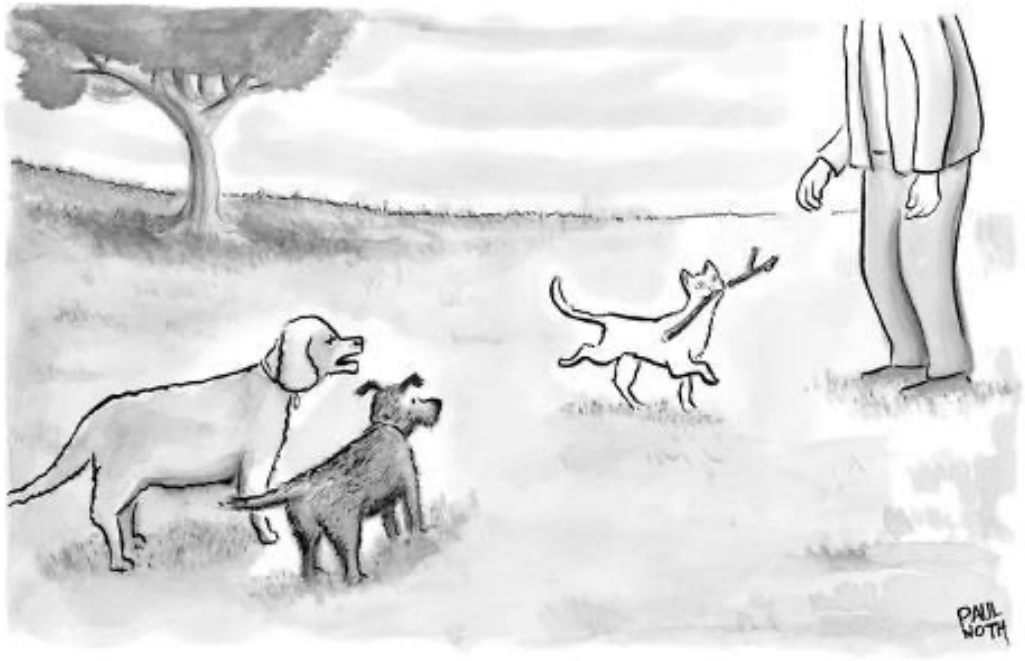
Comp.	Average	Std. Dev.	Min	Max
10	7203	351	6523	8280
20	7477	514	6745	8837
50	7633	433	6708	8223
100	8203	657	7202	9936

Push

Comp.	Average	Std. Dev.	Min	Max
10	12582	307	12099	13190
20	12676	223	12327	13794
50	12778	172	12428	13044
100	12820	289	12156	13387

- We simulate tasks that do large computations; this can give speedups because communication overhead is less important
  - Each node does 100 tasks with computation time given
  - For 10ms to 50ms, forwarding is not effective (tasks should be done locally)
  - For 100ms, pull is clearly more effective than push (better when computation-intensive)

MapReduce conclusions



"It is not done well, but one is surprised to find it done at all."

- We show that it is possible to do MapReduce at the extreme edge
  - On small unreliable sensor boards with an unreliable communication network
- This is the cheapest solution and in many cases it may be sufficient
  - No hardware is needed other than the sensor boards themselves
  - Of course, it cannot compete in performance with a cloud framework! That is not our goal. But the fact that it is both possible and cheap opens the door to many IoT applications that would otherwise be impractical.

GENERAL CONCLUSIONS

General conclusions

- Internet of Things is the Next Big Thing™ !
  - It is growing exponentially and at a much faster rate than the cloud
  - (Together with the other Next Big Thing, Machine Learning ☺)
- Big companies (GAFA) are jumping on the IoT bandwagon, but they all see IoT as extending their cloud infrastructures
  - This leaves a wide-open opportunity for independent developers
- IoT is ripe for exploration by the little guys in their garages
  - GRiSP and Erlang are an ideal platform for this
  - We give two case studies, namely sonar-based person tracking and MapReduce, both running on the extreme edge
- We invite you to download and run the software, and do your own experiments and extensions
  - For example, it's a great basis for interesting class projects
  - At UCL we will continue our experiments with GRiSP for IoT, extending functionality, using more sensors, and building applications