Different Paths to High Availability by Introducing Redundancy in a Distributed SCADA System

Morten Andersen

IT-Vest Thesis Examination – 27-06-2011
Master of IT – Software Development

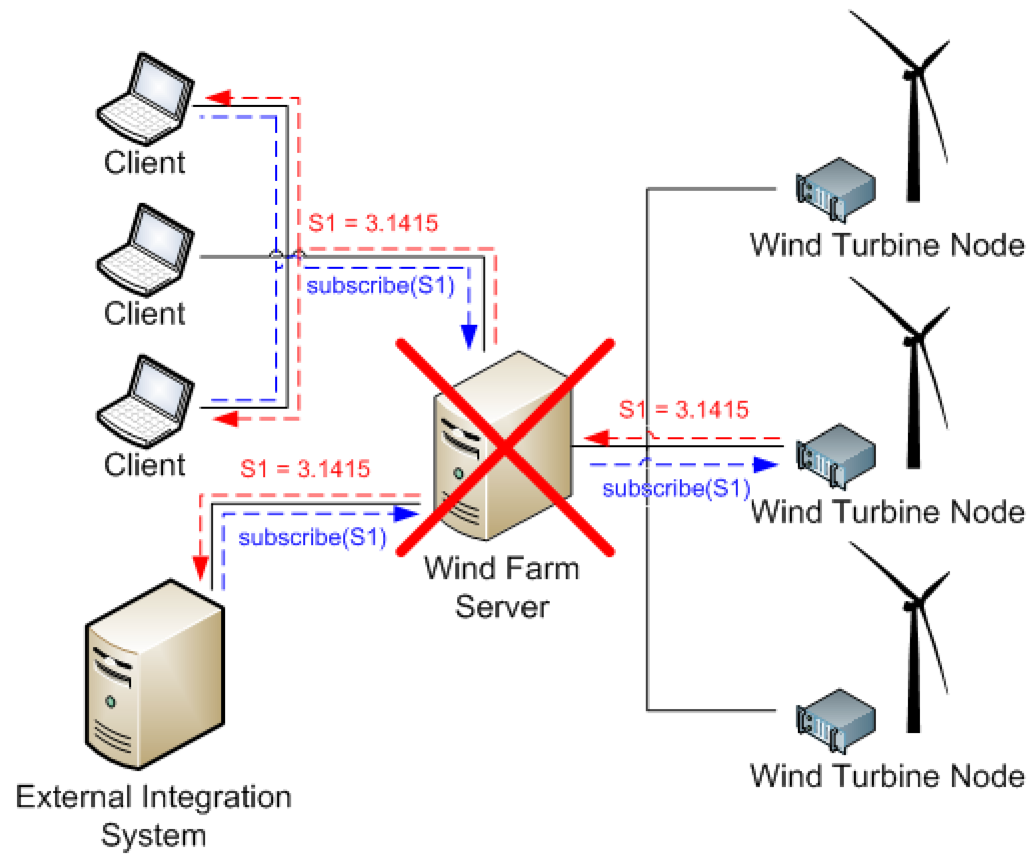
Exam Question

”Med udgangspunkt i din rapport ønskes en præsentation af hovedideerne i din hovedopgave:
motivation, problemstillinger samt hovedkonklusioner.

Herefter ønskes en detaljeret redegørelse for test opstillingerne og evalueringen (§ 7+8).”

Motivation

- Existing distributed SCADA system
- Soft real-time sensor monitoring
- Wind farm server: *multiplexing forwarding observer*
- **Wind farm server: single point of failure**



Motivation – Evolution

Important
standalone
system

10 years →

Important
infrastructure
system

Problem Statement

”To **analyse** how the theoretical high availability tactics can be applied to the distributed wind farm SCADA system as an evolution of the existing system.

Based on this analysis, to **apply** several of these theories as architectural prototypes on a model of the distributed wind farm system.

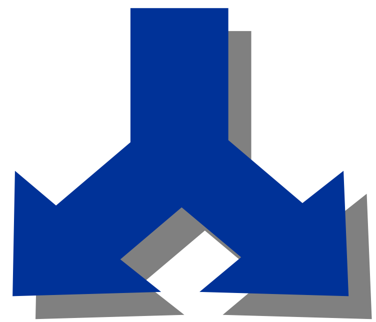
And finally, to **evaluate** the suitability of the applied solutions.”

Architectural QASes

- **QAS1** (availability): 5 seconds deadline for sensor readings at failover time
- **QAS2** (availability): 2 crashed wind farm servers
- **QAS3** (performance): 1 second deadline for sensor readings at normal operation

Availability Tactics

Spare
Wind farm server

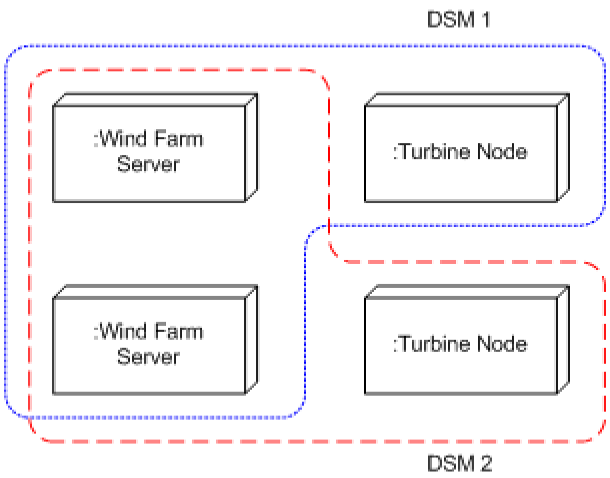


Passive Redundancy

Active Redundancy

Reconstruct-able/Soft State
End-to-End Principle

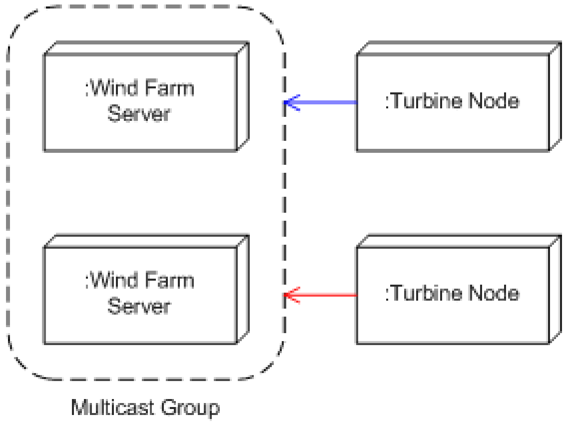
Passive Redundancy Prototype



- DSM based
  - One region per turbine
- Using **Terracotta**
  - DSM = network attached memory
  - Distributed objects
  - Consistency: Hybrid, entry consistency
- Transparency: Distribution of data is hidden

Active Redundancy Prototype

- Multicast based
  - Turbine nodes not part of multicast group
- Using **Hazelcast**
  - Explicit API - library
  - JMS Topic alike interface
- Explicit communication



End-to-End Based Prototype

- Extend original solution
- "Redundancy in network routes"
  - Turbine nodes sees wind farm servers as individual clients – i.e. no group concept
  - "Multicast" (hand crafted) – but not to entire group
- Re-computable / soft state

Evaluating the Prototypes

- Distributed system
- Hard to verify
  - Cost of hardware / setup
  - Non-deterministic

Evaluating the Prototypes

- Quantitative test (qualitative – see app. F)
- Monte Carlo experiment
- Simulated network with random crashes
  - Repeatable sequence (known seed)
  - Within bounds (QAS2 – maximum 2 crashed hosts)
- Amazon EC2 virtual server platform
  - Cost effective: price and time

Conclusions

- **Passive redundancy DSM based prototype**
  - Failing in fulfilling all QASes
  - Significant network overhead
  - Large code changes
- **Active redundancy multicast based prototype**
  - First impression: Solves all QASes
  - Experiments revealed: QAS1 only fulfilled in 20% of tests
  - Code changes feels natural

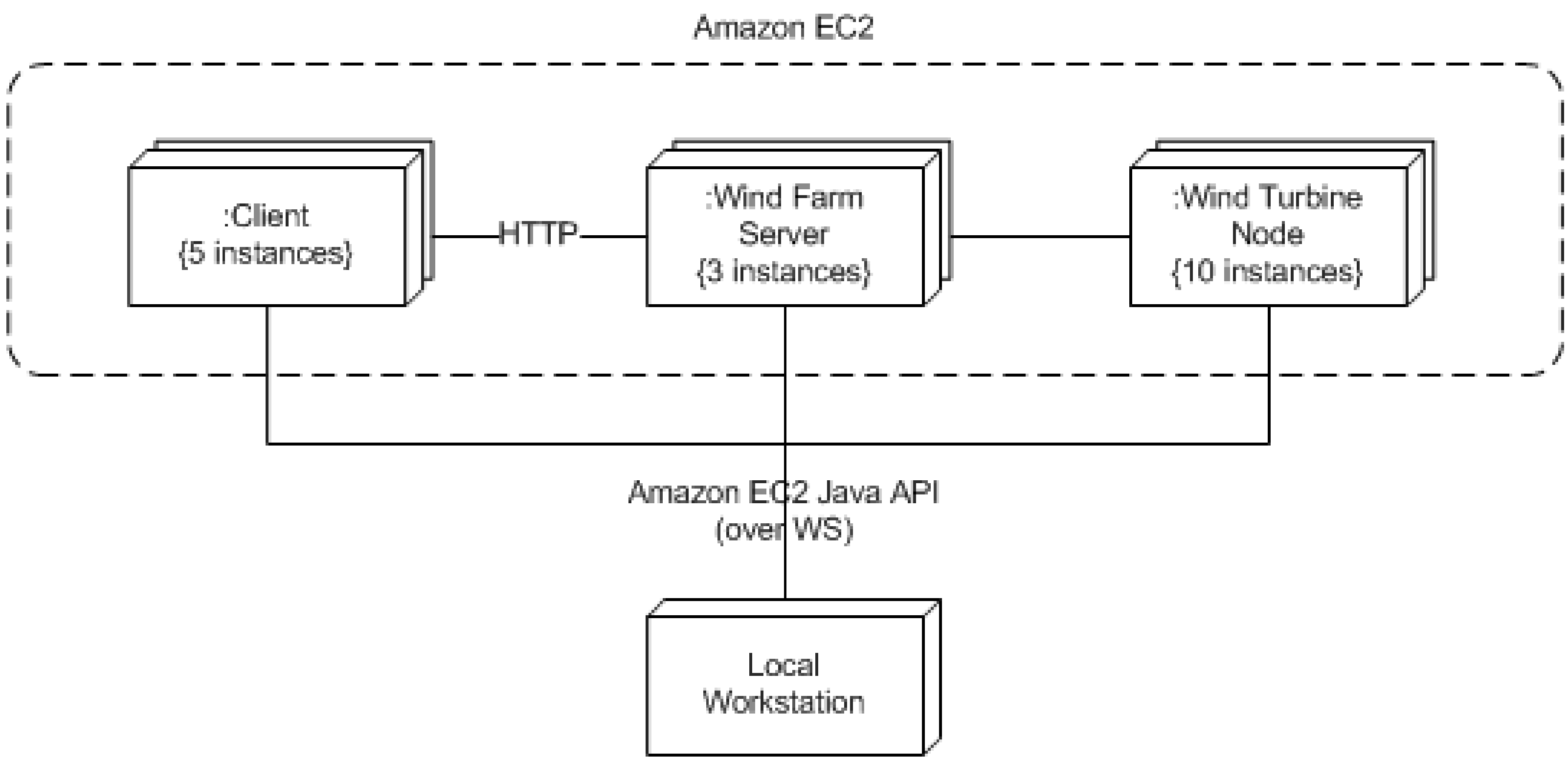
Conclusions

- **End-to-end based prototype**
  - Excels in fulfilling all QASes
  - Complex state – distributed responsibility
  - Hard to maintain

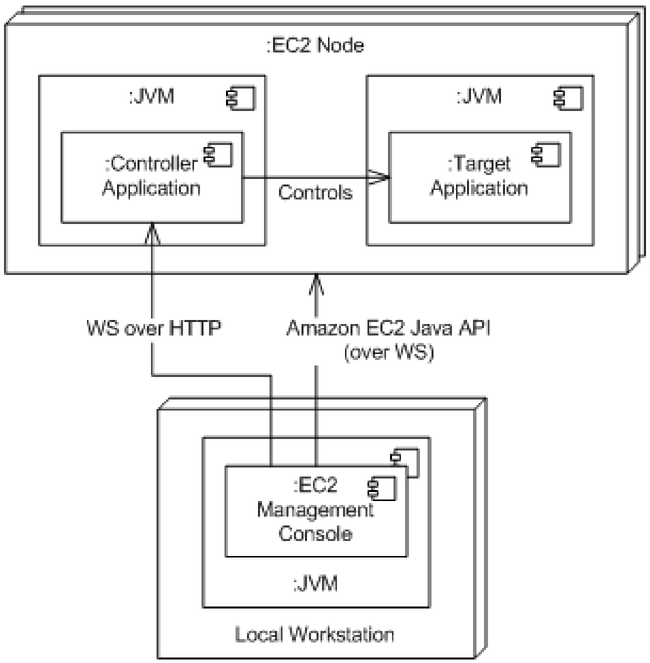
Conclusions

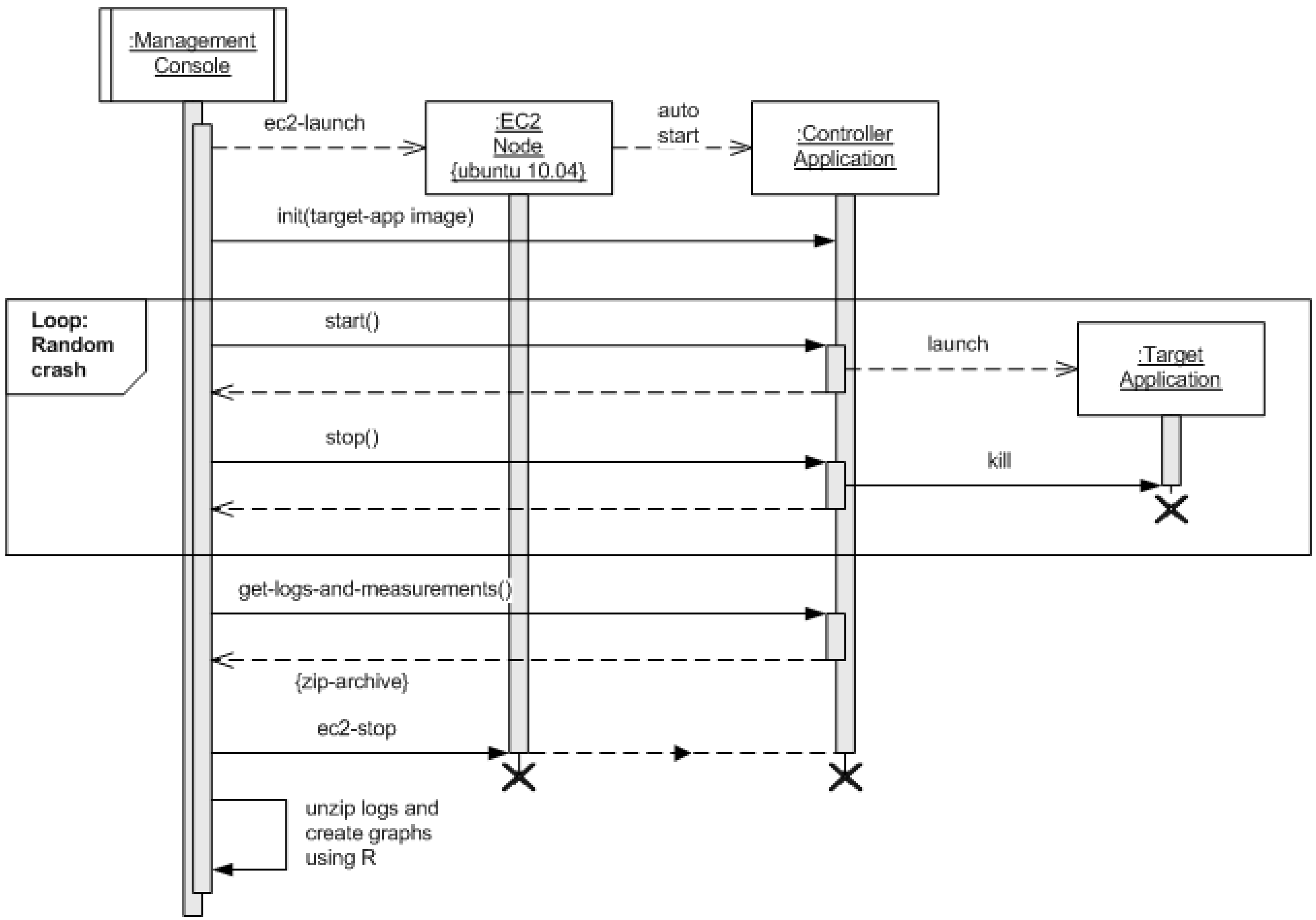
- **Well defined test bench**
  - Solutions comparable
  - Fact based decisions instead of based on feelings
  - Generic test controller components
- **Amazon EC2**
  - Programmable / scriptable
  - Cost effective
  - Hard to debug (fact of multi-node system?)
  - No control over node placement

Test Bench – EC2



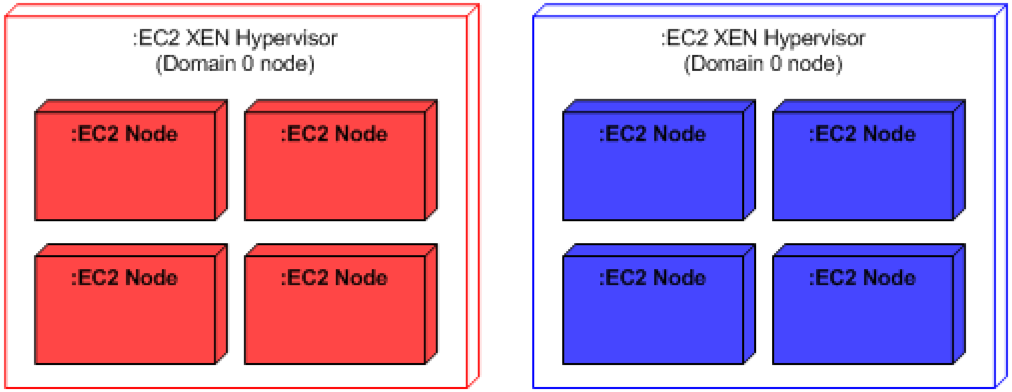
Test Bench – EC2





EC2 – Network Transparency

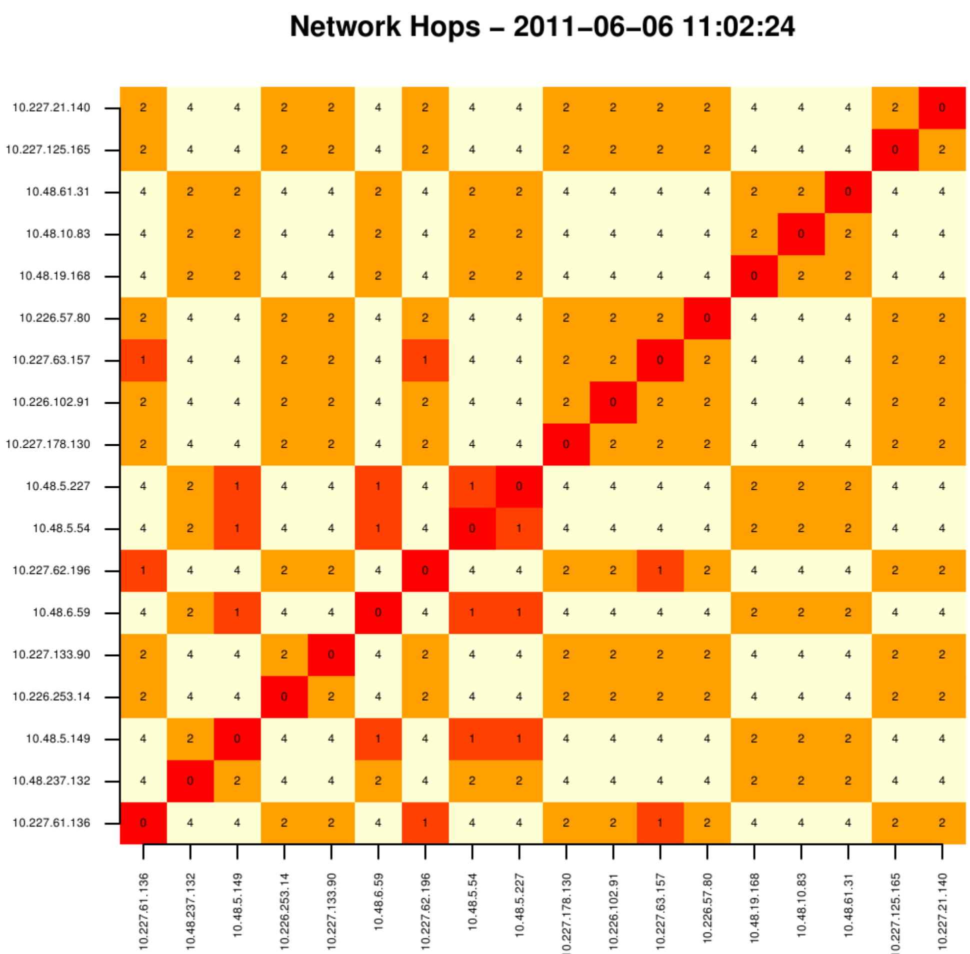
- Network layout not transparent
- Risk of node co-residence
- Averse effects from other nodes



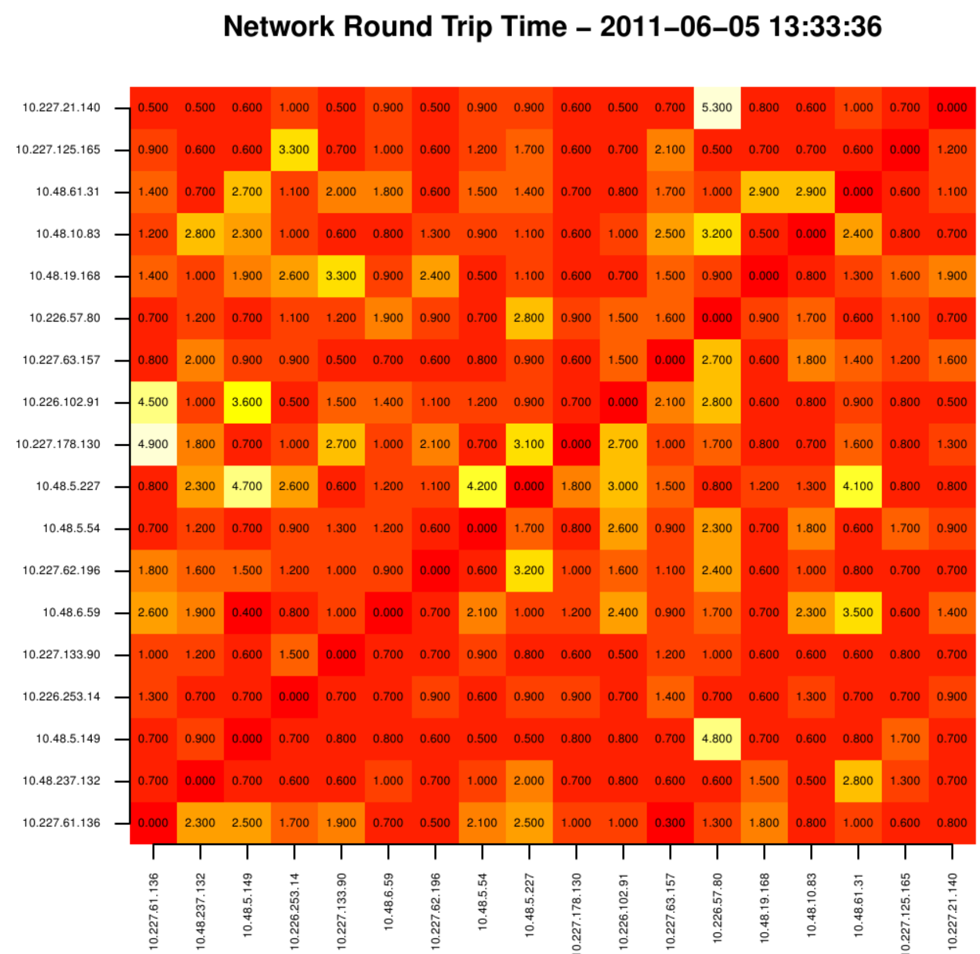
EC2 – Remedies

- Tests run on running instance
- IO – Network measurements
  - Network latency fluctuations
  - No visible correlation: hops <-> latency
  - Fixed layout – no live migration
- CPU measurements
  - Not performed – applications not CPU bound
  - Indirect – e.g. timing a known calculation

EC2 – Network Transparency

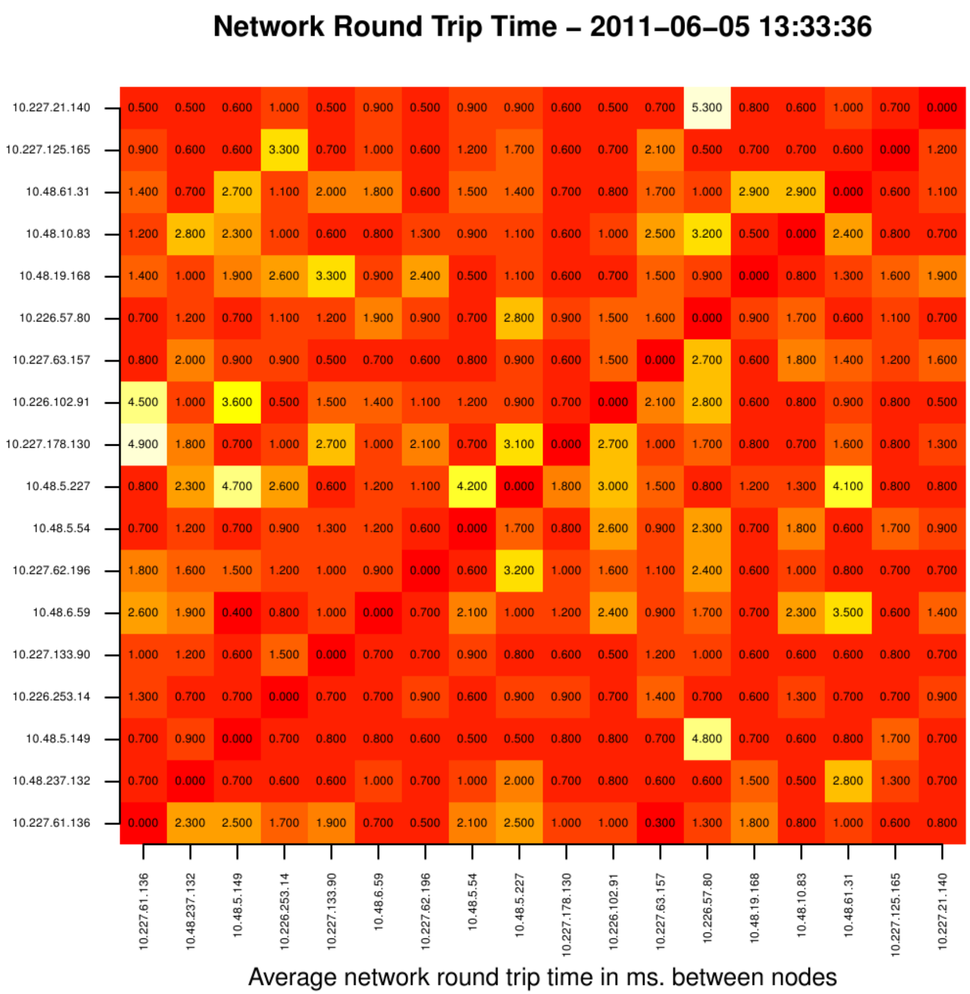


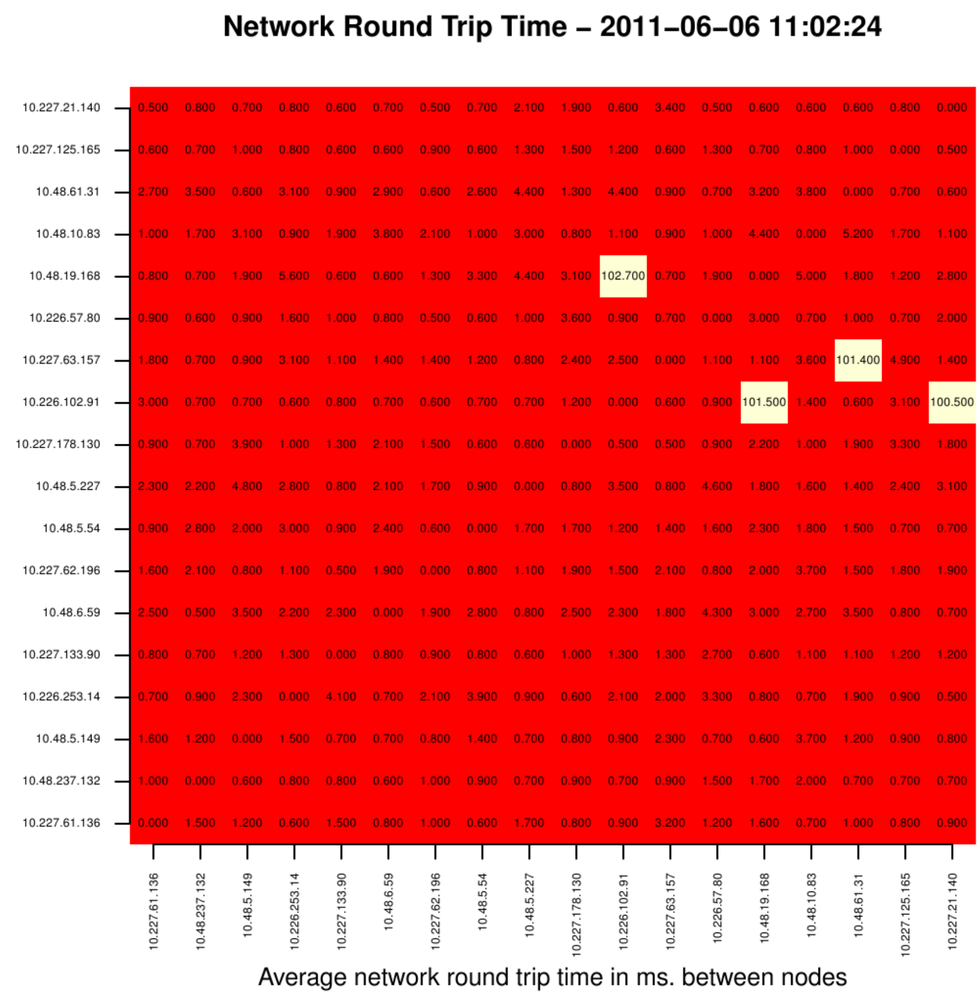
Number of network hops between nodes



Average network round trip time in ms. between nodes

EC2 – Network Transparency



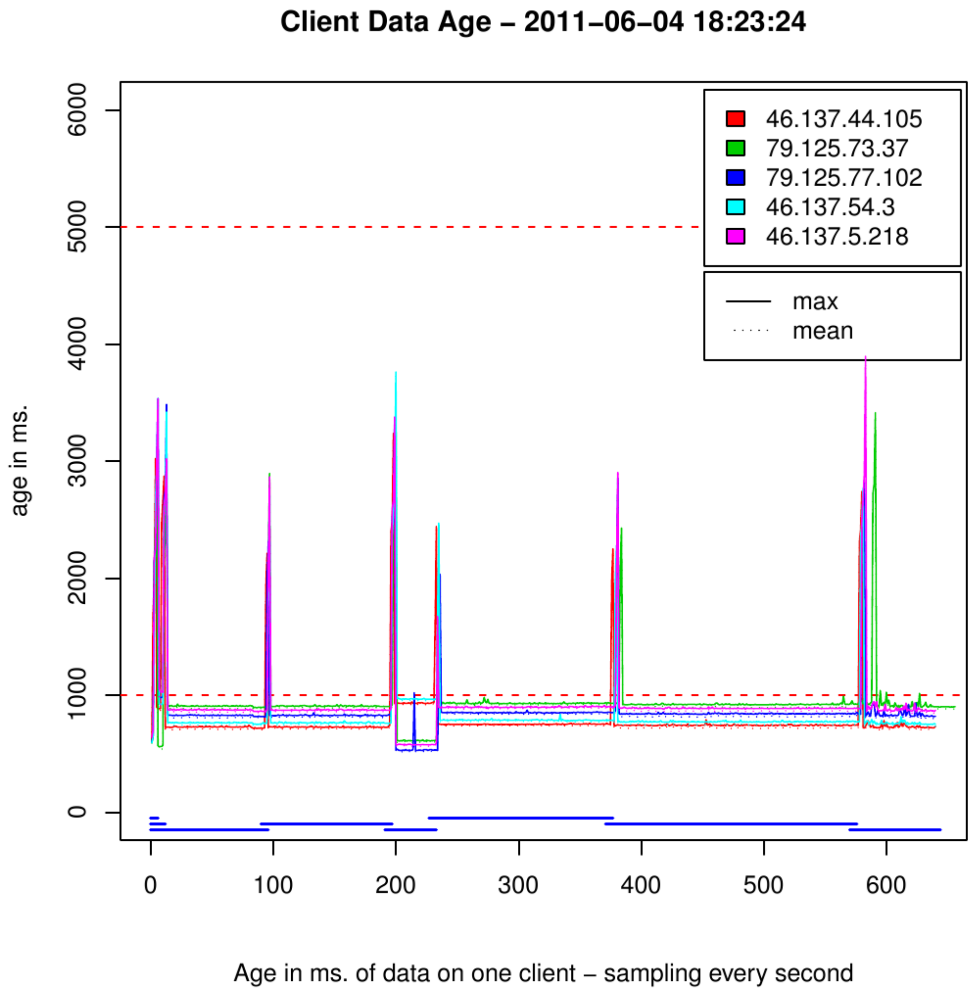


Test

- 100 test runs
  - 20 random length periods [20;40] seconds
  - Random server crash – same state in max 5 periods
  - Per test length ~= 10-12 minutes
  - Total length ~= 24 hours
  - Average of 5-6 server crashes per test

Measurement

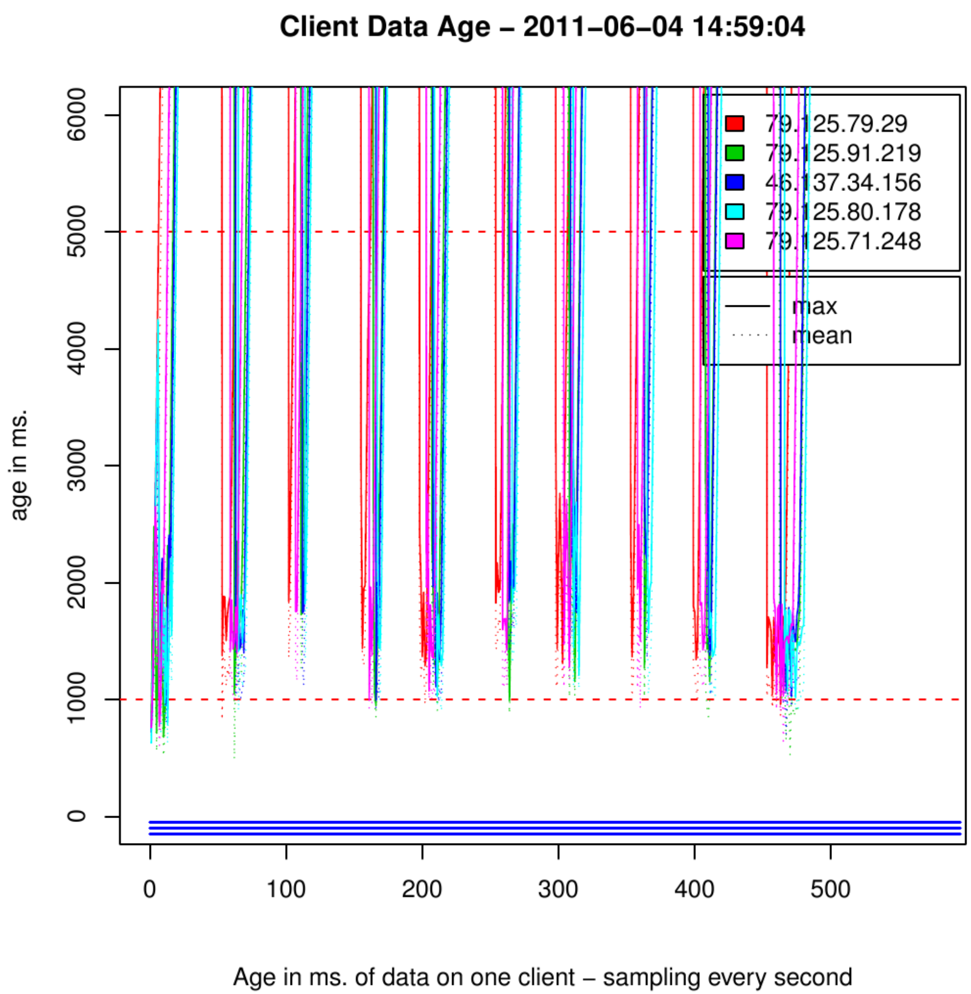
- Timestamp sensor readings on turbine node
- Record age when reading reaches client

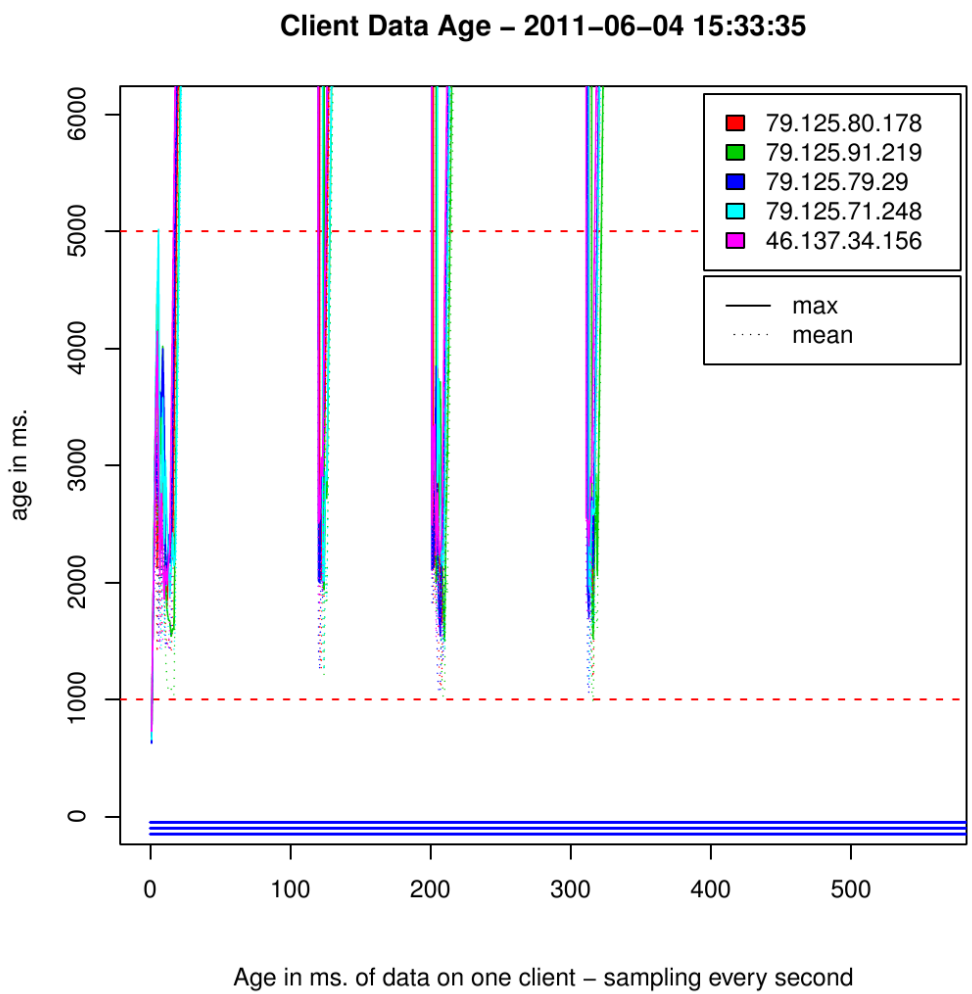


Prototype Evaluation – Terracotta

- 3. party library has big footprint
  - Startup time (+ 20 seconds)
  - Network traffic (factor 10, with larger peaks)

Prototype Evaluation – Terracotta

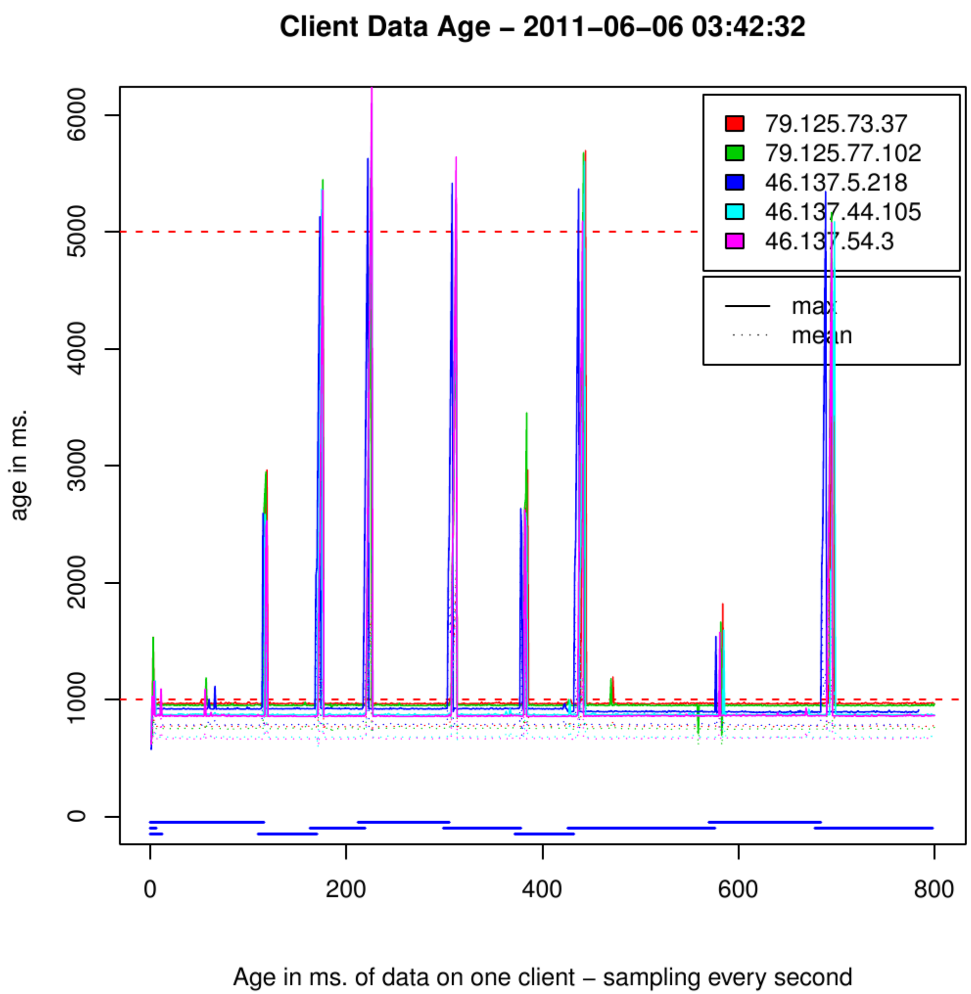


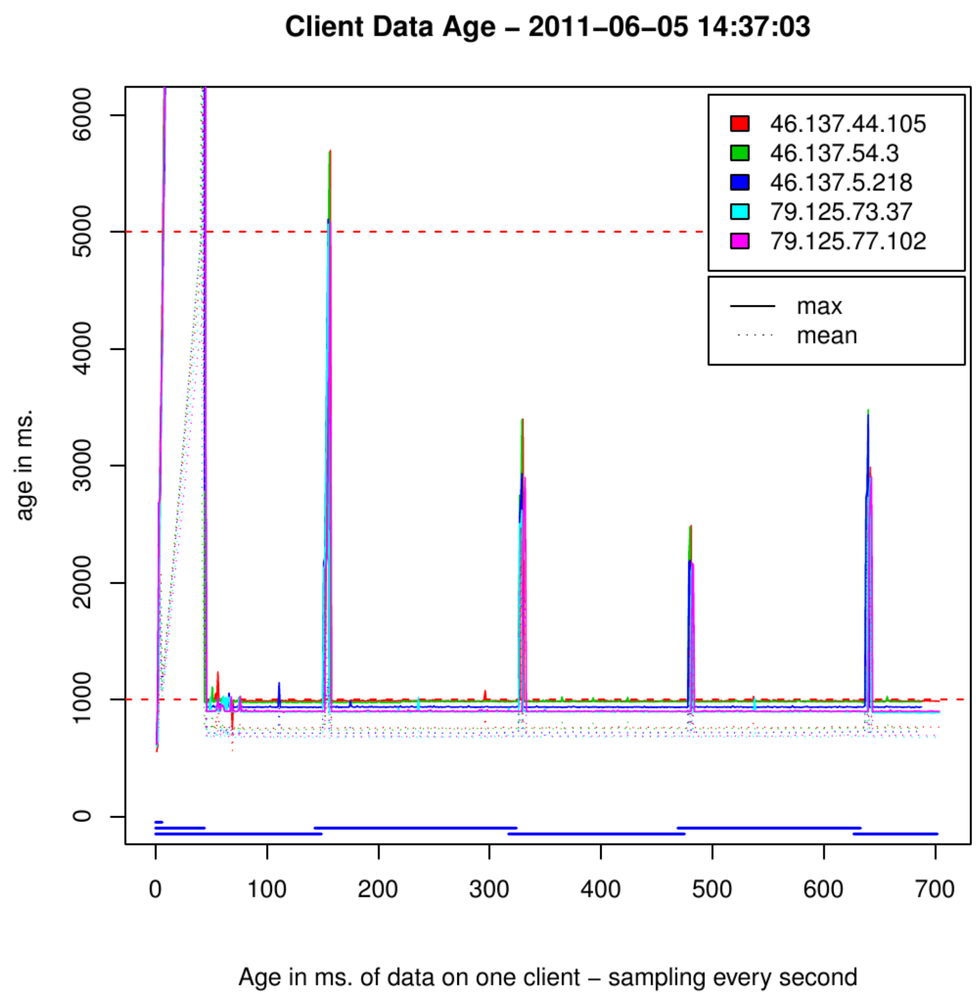


Prototype Evaluation – Hazelcast

- Low network overhead
- **QAS problems in 80% of tests**
  1) Startup problem (bug?)
  2) QAS1 occasionally exceeded

Prototype Evaluation – Hazelcast

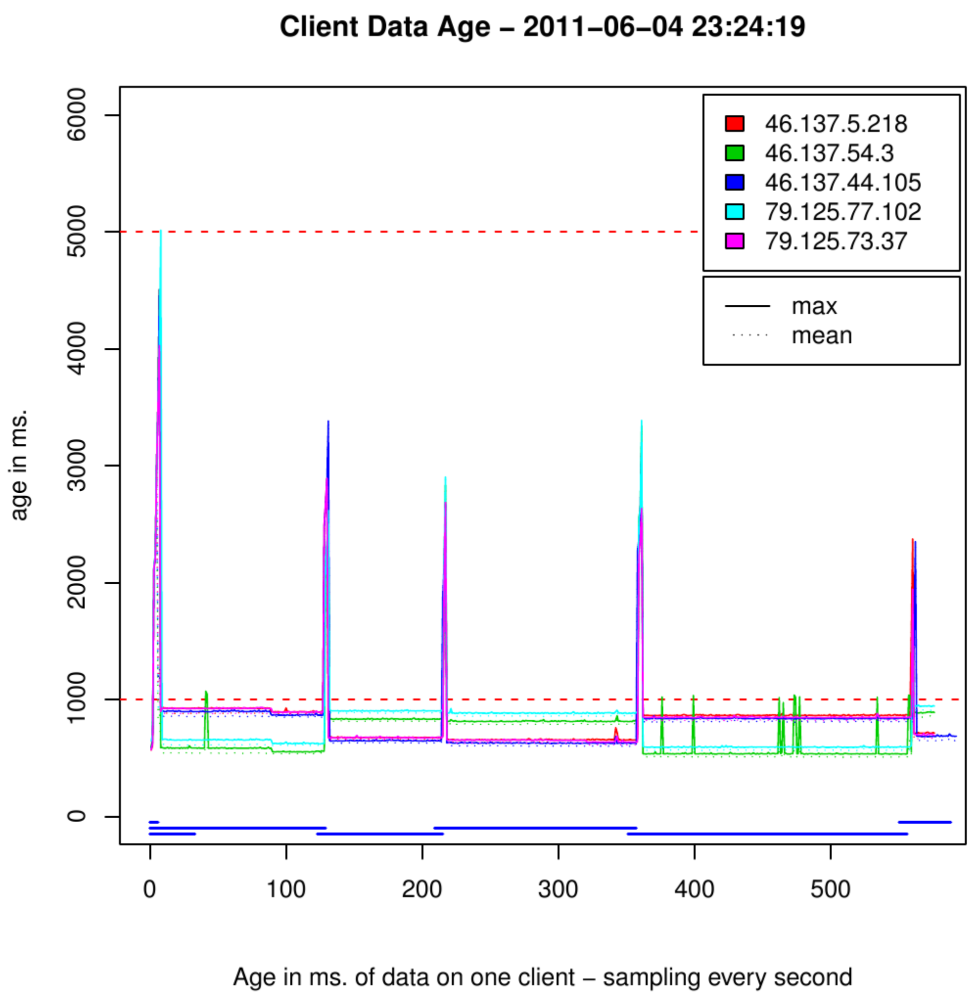




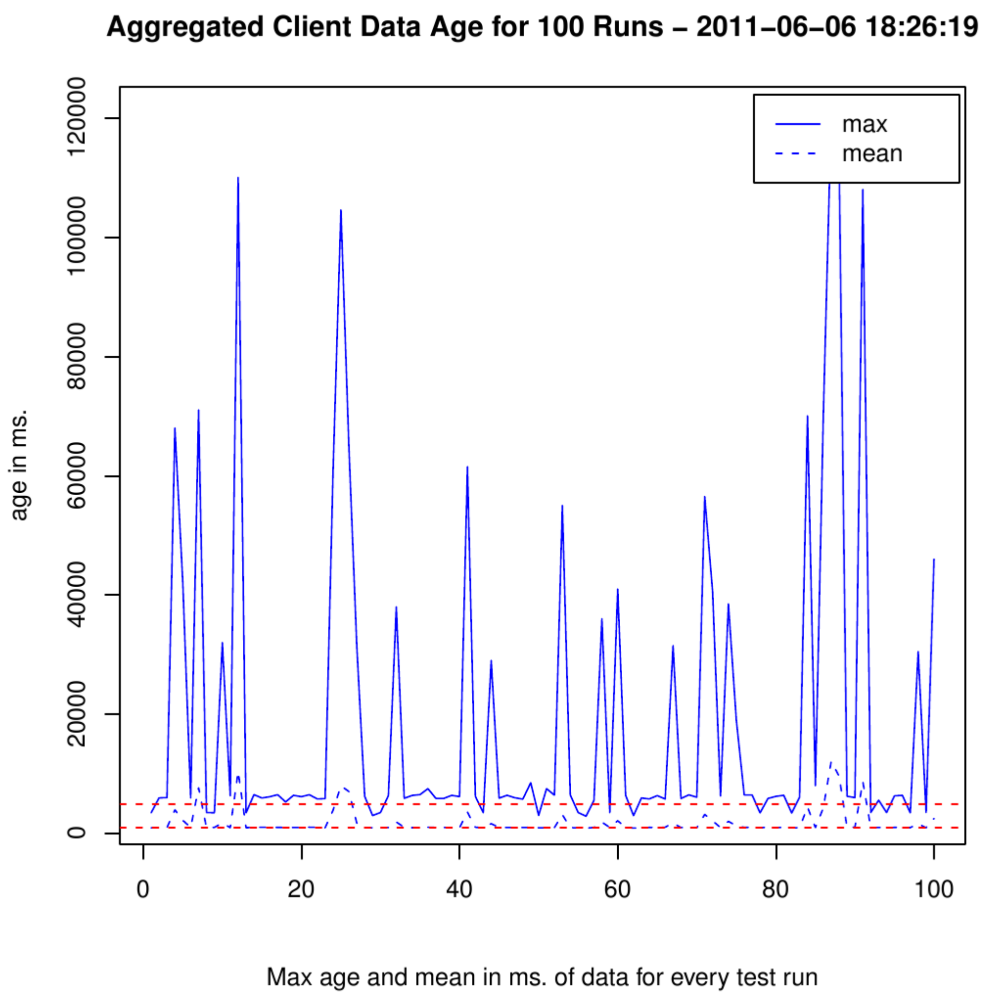
Prototype Evaluation – End-to-End

- **Succeeded on all QASes**
- Segmented network not handled
- NIH syndrome?
- Complex to ensure correctness (fig. 29, p.63)
  - Distributed across nodes
  - Hard to maintain

Prototype Evaluation – End-to-End



Prototype Evaluation – Hazelcast



Network Measurements

Wind Farm Server

Prototype	Kbps/sec in	Kbps/sec out
Passive (Terracotta)	6,000 (peak at 40,000)	10,000 (peak at 75,000)
Active (Hazelcast)	570	1,600
End-to-End	590	1,250

Turbine Node

Prototype	Kbps/sec in	Kbps/sec out
Passive (Terracotta)	150 (peak at 300)	600 (peak at 9,000)
Active (Hazelcast)	41	47
End-to-End	6	48

Critique of Test Method

- No statistical basic
  - Standard deviation
  - Confidence interval
  - Distribution
- Crash of management console during Hazelcast test sequence (test run #17)
- Missing network transparency
- Platform: Ubuntu vs. Windows

00:20:00

"The world is never going to be perfect, either on- or offline; so let's not set impossibly high standards for online"

Esther Dyson