Dissertation for the degree of
Master of Science in Computer Science

Super Sensor Network

Bård Fjukstad

Advisors

Otto Anshus

John Markus Bjørndalen

Faculty of Science

Department of Computer Science

University of Tromsø

May, 2008

Preface

This dissertation is a product of many good discussions with my advisors, Otto Anshus and John Markus Bjørndalen. Also the good residents at the Display Wall lab at the University of Tromsø has contributed significantly to the well being and progress during this work. The creative collection of PhD students that is part of this lab, and what they achieve, is truly amazing.

This dissertation would also never have been possible without the help and support of my family.

Abstract

This dissertation studies composing a super sensor network from the combination of three functional sensor networks; A Sensor data producing network, a sensor data computing network and a sensor controlling network. The target devices are today labeled as large sensor nodes. The communication are based on an IP network using HTTP as the main protocol.

Bonjour is used for service discovery, with some adjustments for technical reasons. This allows for naming and location of available services without centralized servers, and it is implementable in small devices.

A super sensor network for meteorological observations is emulated using a computer cluster. The emulated measurements are accessed from stations available from observation collection systems accessible on the Internet. Images from web cameras are one kind of observation type used. The implemented system uses Python for rapid prototyping and for support for multiple operating systems.

This dissertation demonstrates that the selected technology and architecture may handle some of the demands in a sensor network, and that the architecture gives new opportunities on how to handle updates and sensor network control.

The implemented system also demonstrates that using standard Internet protocols can make access to services in the sensor network easy. A web browser may become the preferred user interface for controlling and accessing all parts of the sensor network, as it has for controlling printers and simple network devices.

Keywords

Sensor network, Overlay network, IP in sensor networks, Service discovery, Bonjour, Zeroconf, Meteorological observations, A-synoptic observations.

Contents

List of Figures

List of Tables

1 Introduction

Wireless Sensor Networks is currently a very popular area that generates much research. The popularity and maturity of the industry has also resulted in actual deployments of large WSNs, i.e., in many hundreds of nodes. Many sensor networks are deployed within industrial monitoring and in environmental monitoring. The future may look bright as a book on the Roadmap of wireless sensor networks state on page 1 [1]:

> *According to a market study performed by ON World Inc. on Wireless Sensor Networks called "Wireless Sensor Networks - Growing Markets, Accelerating Demands" from July 2005, 127 million wireless sensor network nodes are expected to be deployed in 2010 the growth of this market later on is expected to increase in certain application domains.*

Meteorological stations are a subset of the general class of sensor networks. In traditional meteorological networks, the individual nodes are reporting directly to a central collecting system and not as part of a network. Traditionally, most of the computations and all of the control have been done within the central collection systems. In this dissertation meteorological stations will be used as a framework for looking at architectures in sensor networks.

The need for software updates for deployed hardware/systems is often first noticed when some error occurs in the node. A classic tale of remote updates comes from NASA. The story found on the web [3] tells that the Mars Pathfinder (1998) had serious priority inversion problems that caused the system to frequently reset. This problem was fixed when a global parameter was set as part of a software update from ground control on the Earth. The conditions leading up to the fault was not anticipated before deployment. The meteorological package was thought to have a maximum rate, and software priorities and testing was done using this rate. On site, this proved to be insufficient as the conditions proved much more favorable and the package was able to operate at a very high rate.

Marketing studies have also shown that size is not a major limiting factor for deployment of sensor networks as shown in Figure 1 cited in [1]. The most important aspect was reported to be reliability.

In meteorological networks today, is is most common to have a fixed observation frequency. This frequency is usually determined from the different types of observing stations. World Meteorological Organization (WMO), operates a network for data exchange between countries and organizations. In the Word Weather Watch [4] program, coordination of observation times are important. A surface observation from a land station is called a SYNOP, from Greek *Sunoptokos* meaning *Together-seeing*. It is important to make observations at different places at the same time so that a common state of the atmosphere could be analyzed.

In modern days, satellites and radars provide observations at all times, and there exist a need for improving the control of the observing network to mach the

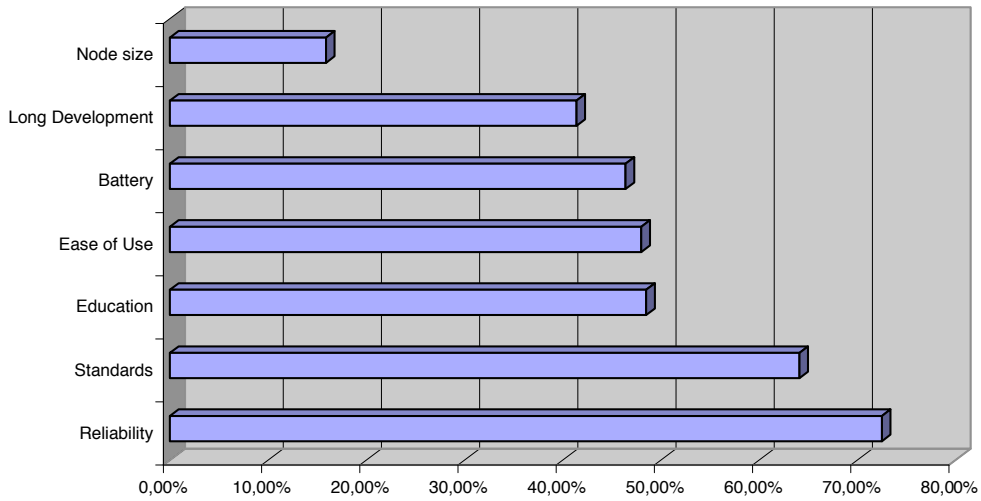

Figure 1: From ON World Inc. Study: Wireless Sensor Network Adoption Inhibitors, 2005 as cited by [1]

new systems. The system needs to be flexible to accommodate new types of information. On the observation-user side, new methods have emerged. ECMWF [5] gives one example of new analysis schemas in use in the atmospheric models. The new schemas are called "4DVAR" , where the 4th dimension reflects time and the other three is the spatial coordinates. An illustration of this method is given in Figure 1. This method can make use of observations outside the fixed times. The question becomes how to manage the network nodes and how to use the networks in an efficient manner dependent on the actual situation.

Including new forms of observations like images or video, increases the need for local processing of data to reduce the need for bandwidth. In [2] a Camera mote for intelligent surveillance is developed. Figure 3 show how bandwidth requirements drop as the level of processing (Intelligence) is increased. Where to place this processing is dependent on many parameters. Local sensor nodes may have severe restrictions in available electrical power and may not support all forms of processing. Limited bandwidth in the communication network may also prohibit moving observations (images or video) between nodes. What level of processing that are needed is not always predictable on deployment of the sensor network, and may need to be adjusted as conditions change.

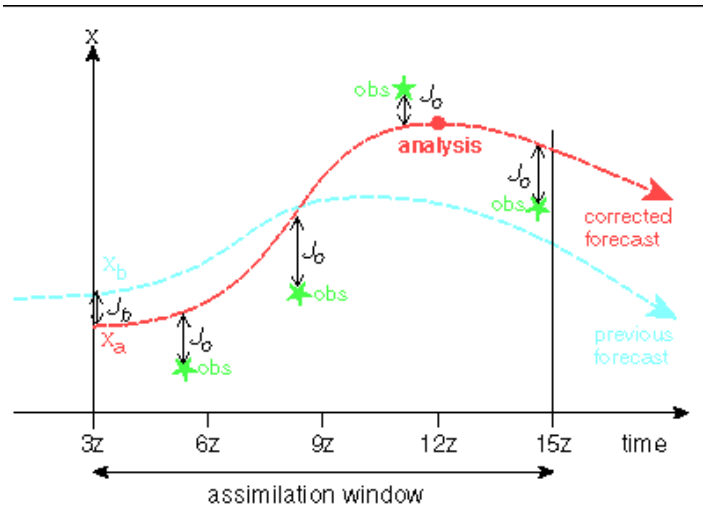

Figure 2: Illustration of 4DVAR. From ECMWF

Assimilitation The process of creating an analysis from a first guess and observations.

Forecast Numerical forecast from the atmospheric model.

obs Observed values at specific times.

z The clock is always in UTC time. Previously GMT, also known as *Zulu, z.*

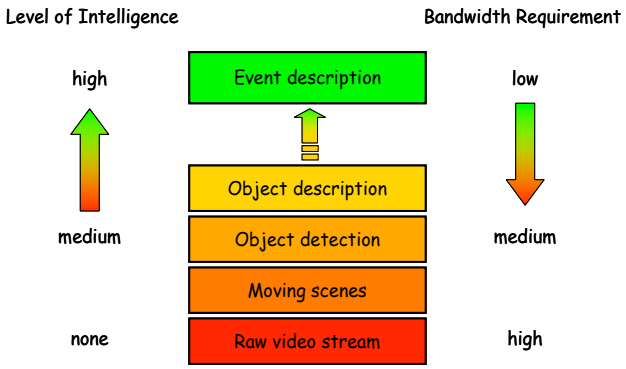

Figure 3: Intelligence-added surveillance. from [2]

2 The basic thesis

This dissertation will examine the overall structure and architecture of a sensor network and the separation of concerns into functional networks. A super sensor network is defined as a cooperating collection of functional networks, as illustrated in figure 4. The following functional networks are identified as part of a Super Sensor Network:

- **Sensor Data Producing Network**. This is the network of nodes that are producing raw observations, either on an individual basis or as a cooperation between several nodes.
- **Sensor Data Computing Network**. This is a network of cooperating processes on nodes that takes the raw observations and process these into elements wanted by other parts of the network or external users.
- **System Control Network**. This network must organize the sensor network such that resources are well managed and that nodes are kept updated at all times.

The basic concept is that the physical placement of each functional component may be adjusted during the lifetime of the sensor network. This also includes the Sensor data producing nodes, as they are fixed in position on deployment, but not in capabilities. All nodes are not part of all functional networks.

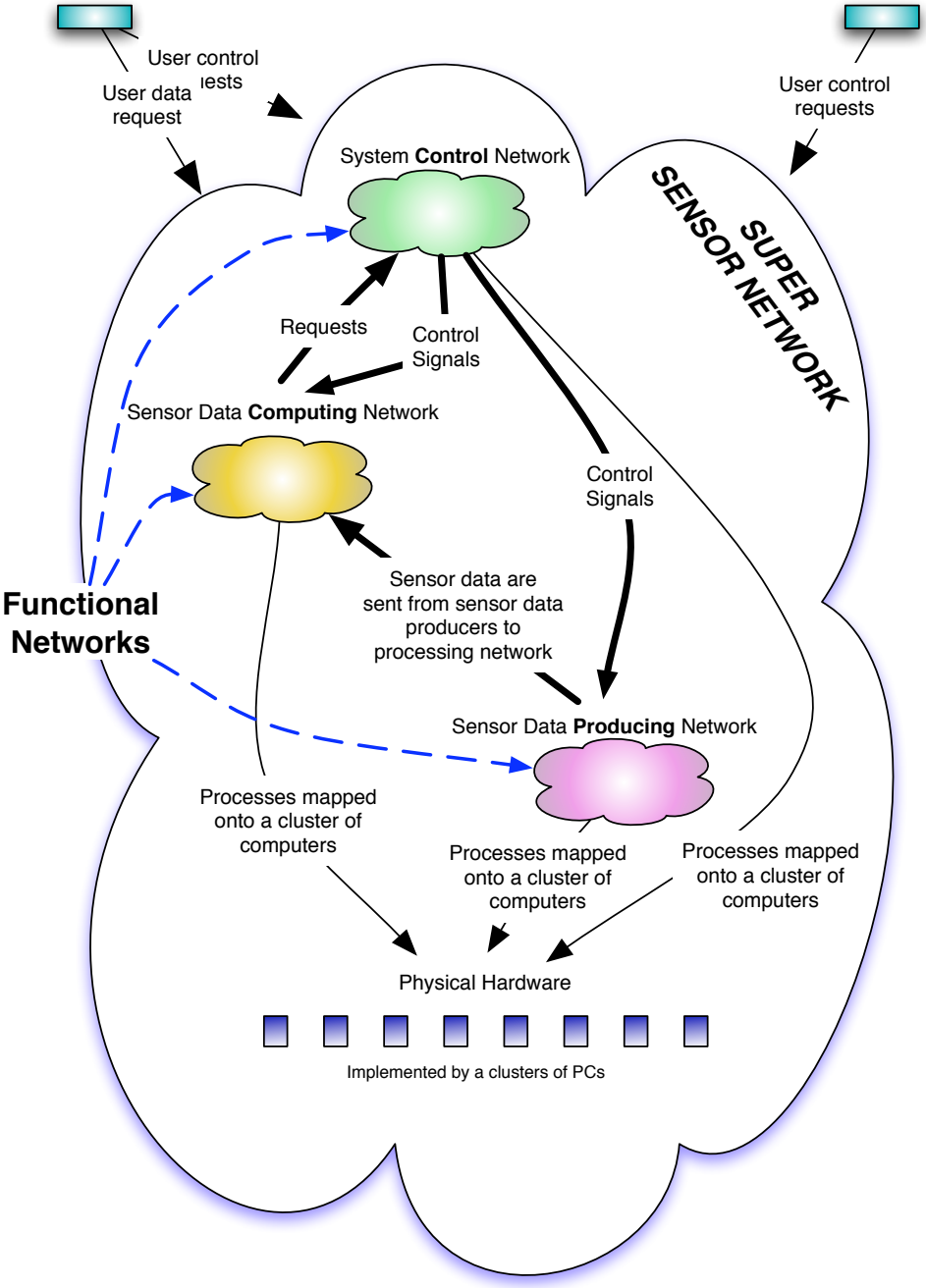

Figure 4: Proposed architecture of the Super Sensor Network and is Functional Sensor Networks

3 Emulation

To study the proposed architecture for a Super Sensor Network, an emulation of a collection of nodes connected by a network is created. This emulation is run on the cluster available in the Display Wall lab at the University of Tromsø. An assumed network of meteorological stations is used as a basis for this emulation and comparison with todays meteorological network will be made. A station on Karl XII Island in the Svalbard archipelago is used for comparison. The station is described in chapter 4.3.

This section describes the various assumptions made for the emulated system to be studied. This will to some degree differ from currently available sensor networks, and is intended as an example of a platform that will be available in a few years time.

3.1 Hardware

The computer cluster used for this emulation is a few years old, and can be regarded as typical of the hardware that will be used in large sensor nodes in a few years. The computer cluster consists of around 30 Dell Precision 370 Workstations with 160 Gb disk, 2 Gb memory and an Intel Pentium 4 processor running at 3,2 GHz. A total of 28 computers are used to drive the projectors and the actual display part, and additional computers for the infrastructure. The computers are connected with gigabyte capacity network.

The various basic nodes in the sensor data producing network are assumed to be capable of limited local data computing. As an example the wind sensor is capable of reporting both instantaneous wind and 10 minutes average wind speed.

Sufficient electric power is assumed to be present to allow the units be able to :

- Sustain a standard observational frequency. Typically each 10 min in meteorological networks.
- Sustain any computing needed at each node. This may vary between nodes.
- Sustain any wanted mode of communication.
- Sustain the node for a reasonable time of operations, typical in the order of 1 year.

A real-world sensor network may have limited electrical power available. The Norwegian polar station has a power consumption of 60 - 100 Ah / year, where 50% is used by the satellite radio transmitter [6]. The battery gives an operational period of 1 to 1 1/2 year.

3.2 Communication

Sensor network nodes are emulated with communication capability for a two-way communication and IP capable.

The communication hardware is assumed to be able to communicate with high reliability. The communication channel is assumed to have sufficient bandwidth for data and control information. The communication may use land-lines or wireless communication. Todays standard WIFI (IEEE 802.11 a/b/g/n) setup may be an example of the communication capabilities we try to emulate. With effective transfer speeds in the order of 1.4 - 31 MB/s [7].

The individual nodes or devices are configured such that they may be reached by other nodes or devices on the same subnet. This is done using standard DHCP and DNS. For technology like Bonjour/Zeroconf (see chapter 9.7.3 for details) this is not strictly necessary, but as the available platform for emulation did not support all of this technology, some prior configuration must be present.

3.3 Security

No security problems or challenges are assumed present in the sensor network. Security of sensor networks is an area with much on-going research and a good solution is assumed for all parts of the super sensor network.

3.4 Failure models for the emulated system

The three functional sensor networks may fail in several ways. This dissertation does not describe all possible failure modes, only the effect on the functional networks.

1. A failure in a node in the sensor controlling network or in the sensor data computing network, is assumed to be fatal for the node. All associated processes on the node are stopped. The failure is detected by other nodes in the respective networks, and a new node may have to be started. This implies that there is at least some redundancy in these networks.

2. A failure in one of the sensor data producing nodes is assumed to be so serious that the node is only tried automatically started a limited number of times. No new node is restarted and the node will simply drop out of the sensor data producing network. This emulates the real life situation where a remote node no longer is functional. The data collected by this node cannot be replaced or collected by other nodes. No redundancy is present.

3. Communication hardware is assumed to have very low failure rate and no redundancy is emulated. We also assume very low packet loss in the network and only implement a limited number of retransmissions when using UDP. Standard mechanisms will handle most errors when using TCP.

4. Failure in retrieving data to a user either from a sensor data computing node or a sensor data producing node is always assumed to be non-fatal and a simple error message is returned.

5. We do not emulate sensors that produce incorrect data.

3.5 State reporting

In a real-world situation local error conditions in a node will be transmitted as part of the regular communication and status reporting. In this implementation errors are not reported.

3.6 Update policy

The functionality of sensor nodes may be adjusted using different models:

- Functionality decided by parameters set locally at deployment in the field.
- Reconfiguration or updated software after deployment.

The first approach implies that the programmer has anticipated some problems and states the node may be in, and the functionality is changed with altering some parameters in the node configuration. The software is normally available for extensive testing on the actual hardware prior to deployment. Errors found after deployment is not easily correctable and cannot be tested on the actual hardware, only on replicas on accessible locations.

When updating the configuration or parts of the software on the node, more communication resources is used and a two-way communication channel must be present. For some systems like Maté [8] the actual number of bytes transfered may be very small even for a substantial update. In this model unanticipated states and failures may be handled using a new software update. It is also possible to implement new functionality, like new video codecs, that became available after deployment.

3.7 Scale

In this dissertation the assumption is a number of stations in the hundreds. This would cover parts of a county (Norwegian "fylke"), if deployed similar to the network operated by the Norwegian Meteorological Institute [9].

3.8 Naming

For addressing any node, it has to have a name/address which has to be unique in the network. The nodes should also be addressable as a group using different types of multicast. In this dissertation such a naming is assumed present.

3.9 Synchronization

One of the aspects of synchronization is to maintain global coordination in sensor networks. This may be the problem of ensuring common data handling during software updates, or maintaining a routing algorithm. Meteorological observations are always tagged with a time stamp. This makes important that the clock on the individual nodes is accurate to a selected level. In current sensor network implementations, this problem has been solved with a resolution of sub-milliseconds [10].

The clock used in the Norwegian Meteorological institute polar weather station has a drift of around +10 min pr. year [6]. The large clock drift is offset by setting the clock 15 min slow on deployment. The needed accuracy of the time for meteorological stations, is within several minutes. This is a function of location and the density of stations and the phenomena that are expected to be resolved by the observations. Typical requirements of these meteorological stations are clocks accurate within ±1 min/month.

In this dissertation we use the clocks on the computers in the cluster. These clocks are accurate to a higher degree than required for meteorological networks.

3.10 Data storage

In traditional wireless sensor networks the nodes themselves have limited storage capacity. In the type of station that is emulated in this dissertation, the normal storage has the capacity to store all sensor measurements for 3 years with an observing frequency of one measurement for all parameters every hour [6].

This amount of storage available for the emulated stations is therefore considered so large that no restrictions are placed on the storage capacity of the local nodes.

3.11 The emulated station compared with other sensor networks

Karl and Willig [11] lists a number of different general characteristics of Wireless Sensor Networks. The type of system emulated in this dissertation differ from other sensor networks in several aspects.

- Quality of Service. The observations from remote stations are considered important but single observations may be missing without large difficulties. Observations may also be delayed without problems. The transmission schedule also delays observations.

- Fault tolerance. In the network operated by the Norwegian Meteorological Institute the individual node is communicating directly to a central data store. Fault in one node is therefore not critical for other nodes.

- Lifetime. In some sensor networks the lifetime of a single node may be very short. Nodes may be distributed to observe events with short durations. In our system the lifetime is very long with continuous operations for up to 1 1/2 years, depending only on the power supply.

- Scalability. One problem in meteorological networks is the scalability of the communication system in use. Remote stations often use ARGOS [12]. The communication in the ARGOS system is one-way and the available bandwidth is limited. In other systems a very large number of nodes may be operating within a small area and the communication and data flow have to handle a large accumulated volume of traffic.

- Node densities. In todays meteorological system the sensor nodes may be several hundred kilometers apart. This makes communication overlap less likely, but as the nodes may be communicating with the same satellite at the same time, radio interference is probable. In other systems high node density may lead to the need for protocols to avoid data duplicity since several nodes may have almost identical observation values.

- The energy requirements. Meteorological observing stations has battery capacity for operations of up to 1 1/2 year. The stations use most of their energy in communication (approx. 50%) and in environmental measurement, where the need to heat the pressure sensor is the largest drain of electrical power. The station is sending data using its radio every 200 seconds. The limited battery power requires the station to be inspected yearly. Due to bad weather the station on Karl XII Island was not reachable the summer of 2007, and was expected to close down in the spring of 2008. Other types of sensor nodes may have even more serious restrictions in communication and operation.

- Network densities. The Norwegian Meteorological Institute has a limited number of nodes in the Svalbard area. Other sensor networks may have a large number of nodes used in a small area. This affects both communication and the ability to resolve the observed phenomena with sufficient resolution.

- Mobility. The meteorological station compared with here is a stationary station. There is also some stations on floating buoys drifting in the ocean or on ice floes. These stations are not locally connected, and they are also using the ARGOS system for communications. So even if these buoys are mobile, the needs and systems closely resembles the above described systems. Other sensor networks may involve truly mobile devices. One example of this is InVANET [13], Intelligent Vehicular ad-hoc Network where research is focusing on vehicles and mobile telephones.

Most sensor networks rely on some form of a communication network, and severals issues has to be considered:

- Energy efficiency. Since most of the energy used in a wireless sensor node is for communication, this has to be done in a very efficient manner. This applies to sensor nodes running on limited electrical resources and not systems like urban surveillance cameras, where electrical power may be supplied from the local electrical grid.

- Auto-configuration. Most sensor networks do not know a priori where their location is and what neighboring nodes that are present. The need to self-discovery is therefore a driving force for implementing self-configuration systems. In the nodes compared with, the actual location was not known before the station was in place and since the station could not rely on neighboring nodes and did not have inboard GPS, the location had to be

manually set on location. Other sensor networks may have local GPS and also other forms of location-finding technics.

- Data- or address centricity. The station compared with is address centric as data is transfered between the station and one data sink. Other sensor networks may use a highly redundant deployment of many nodes where a failure of an individual node may not cripple the entire network. This also creates the need for high optimized routing protocols.

One way of handling failures is to mask the failure using redundancy. In very high density sensor networks the failed node may simply be ignored and removed from the network without loss of observing resolution. This is not an option for most meteorological networks, as the cost of each node prevents the deployment of more than the absolute minimum number of stations.

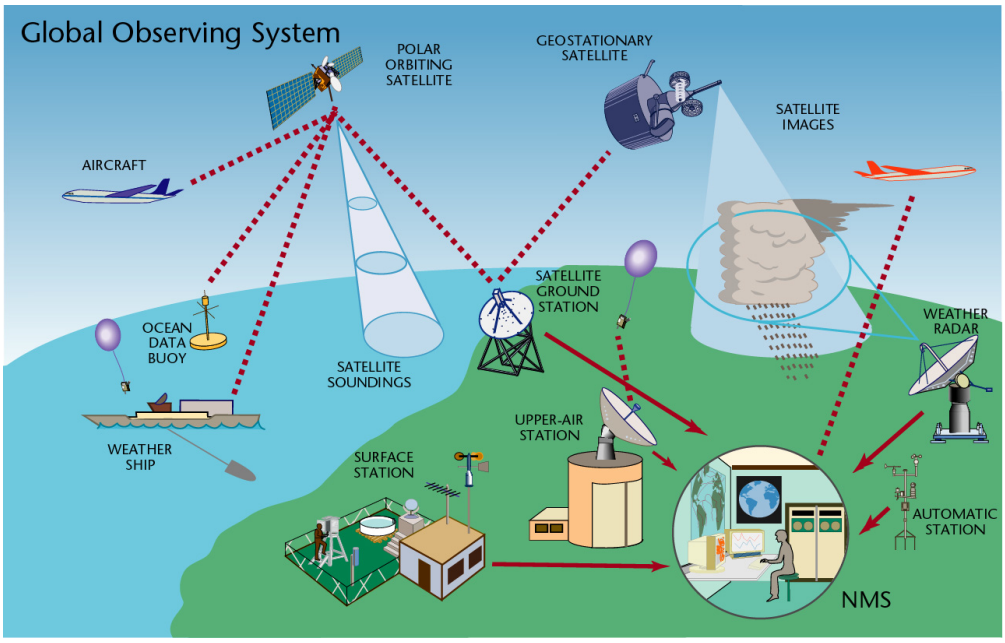

Figure 5: Illustration of the meteorological observing network. From WMO

4 Real world sensor nodes

Figure 5 describes the general system for collecting and measuring meteorological data. World Meteorological Organization (WMO) coordinates the global effort and defines standard for data exchange. On a typical day the distribution of surface observations at one specific hour may be as illustrated in Figure 6.

4.1 Scale

In meteorological networks the number of stations are large. For individual countries the total number may be much smaller. Some typical numbers are illustrated in the Table 4.1.

Table 1: Approximate number of observing stations in different networks

Scope (source)	**Number of station**	**Station type**
Global (Surface network)	28400	Meteorological
Norway (met.no)	700	Meteorological
Sweden (Road authorities)	700	Road weather stations

The total number of national meteorological stations operated by the Norwegian Meteorological Institute has slowly decreased. This is partly due to difficulties with having people available to take observations as frequently as

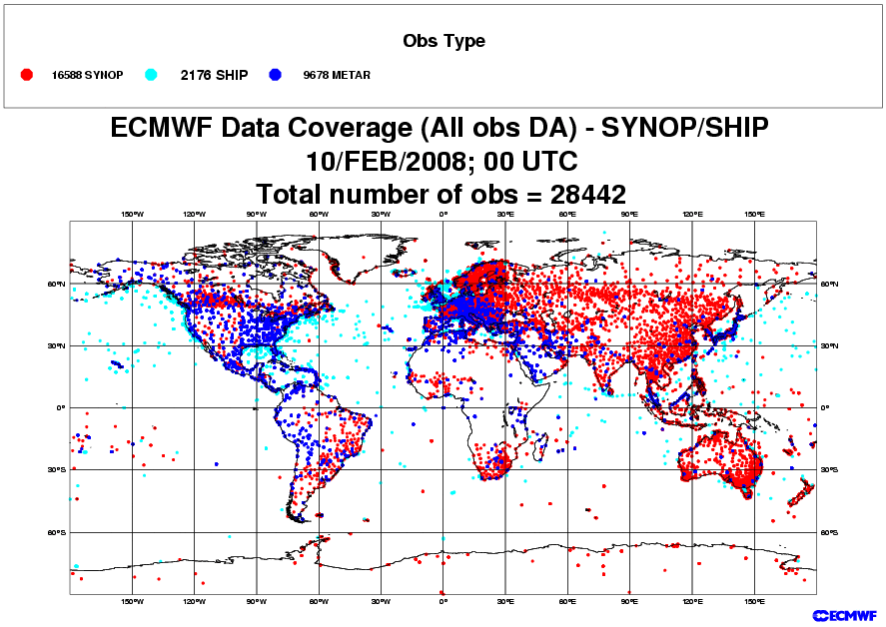

Figure 6: Typical global station coverage at one specific date and hour, from ECMWF

needed. As an example it can be noted that all lighthouses on the coast in Norway, are now unmanned. This reduction in stations is mitigated by increasing the number of automatic meteorological stations. The transition to automatic stations is dependent on available funding and will take several years. The number of meteorological stations deployed by other authorities like the road and railroad authorities, is rising and cooperation has been established. This also makes the integration of different technologies a challenge. The number of such stations will probably equal the meteorological stations within a few years. As illustrated by table 4.1, the road authorities in Sweden are already operating many stations today.

The global number of observation stations is not expected to rise by a huge factor. The bulk of the new observations come from weather radars and satellites. Both of these data types are traditionally collected by a limited number of agencies and then distributed to the rest of the users.

4.2 Data storage

The data storage in meteorological stations are typically data-loggers containing EPROM's with a capacity of 10K – 640K words. A word is typically 8 bits [14] [15]. These devices may operate in a very large range of environmental conditions, like temperatures from -60 °C to +60 °C.

4.3 Karl XII Island

To create an emulated sensor in a sensor network, some of the typical restrictions on real sensor should be used as a basic assumption. The following describes some elements from the meteorological stations operated by the Norwegian Meteorological Institute on remote locations in the Spitsbergen archipelago. All elements from a description of this Polar automatic weather station in this dissertation, comes from an internal Technical report [6] from the Norwegian Meteorological Institute.

One of the most remote stations is on Karl XII Island. This is a very small island in the north eastern parts of the archipelago, bordering the polar basin (see figure 7). The station is so remote that visits once a year cannot be guaranteed. Access is by helicopter, usually as part of a yearly tour by the Governor of Svalbard (Sysselmannen). In the later years the sea ice during early autumn has been greatly reduced giving improved access to these remote parts by ships. The station was deployed in 2003 and was last serviced in the autumn of 2006. Due to weather related problems, the station could not be serviced in the summer of 2007, and ceased to report in April of 2008, most probably due to lack of electrical power.

In the following some of the basic restrictions and elements of this type of station are described. An image of the hardware with size, placement and organization is given in Figure 10.

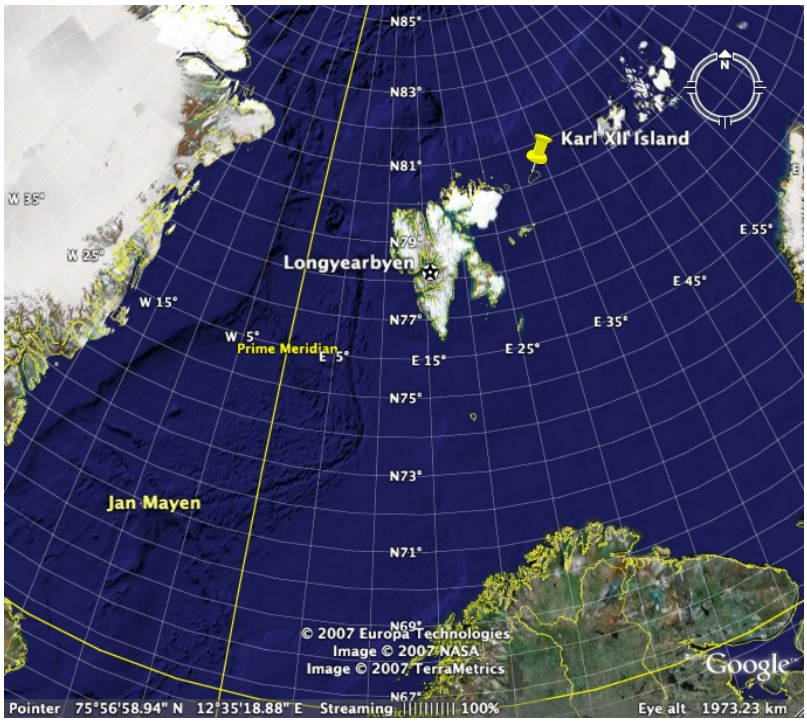

Figure 7: Map of Karl XII Island

4.4 Detail of the station

4.4.1 Observing frequency

The electronic equipment on a regular meteorological station is divided into several different physical parts. One part handles the collection and reading of sensor values. To reduce the power consumption, it is customary to reduce the measuring frequency. On Svalbard the stations normally measure at 10 minutes intervals. The wind sensors handle continuous data collection in the sensor, but the collection produces 10 minute discrete measurements. The pressure sensor is one of the most power consuming sensors, as the sensor has to be heated before measurement. It is therefore used as sparsely as possible.

The physical sensor units are connected to a highly customized unit that reads, stores and processes the data elements before transmission. The transmission is also done at discrete intervals, typically every 200 seconds.

4.4.2 Storage

The scanning unit needs to store values over time and needs permanent storage capacity. The stations described here have a storage capacity to store hourly measurements from all sensors from a 3 year period.

Figure 8: Details from Karl XII Island

4.4.3 Communications

The observations from the meteorological stations are broadcasted via radio every 200 seconds, using a standard unit for the global ARGOS satellite system

[12]. The radio system is a purely one-way system and the stations do not have any means for receiving.

The transmission is highly optimized for low transmission costs in terms of power usage. This also includes optimizing the data volume. One transmission contains 25 word of 10 bits, in a 256 bit package. Of these 25 words, 24 is actually used for sensor data in every transmission including a checksum word. 10 bit words are used as this is suitable for the resolution delivered by the sensors.

4.4.4 Data computing system

The stations are of the type "Aandreaa 1997 " [6] and use a custom real-time operating system and custom closed source programs. The internal details are not known.

4.4.5 Physical constraints

The geographical location of this station implies some physical constraints. The station has to be able to withstand strong winds, very low temperatures and also visits from curious polar bears. An example may be the image in figure 9 taken from a visiting tourist ship in 2007. The station is therefore enclosed in a very strong, safe-like steel box (See also Figure 8 and 10). All cables are housed in steal pipes and not reachable by the polar bears. The bears have previously shown a liking to the insulation material used.

Figure 9: The station visited by a polar bear. ©martin@gnejs.se

4.5 Failure models

The real wold stations like the station described above are difficult to access for repairs. They must therefore have a failure model that provides the obser-

vations also in gradually deteriorating conditions. Some parts of the physical sensor package can have failures that does not affect the other sensors. The communication and computing parts of the node have to be able to handle these types of failures. The system is also able to handle loss of radio communication as observations are stored locally for later retrieval, when the station is maintained. The observation data collected during maintenance visits may be used for climate studies later, and therefore has a significant value.

4.6 Road weather stations

The Norwegian Road authorities has invested much in new road weather stations. Some of these are delivered by ScanMatic AS [16] and an example can be seen in Figure 11. Some of these stations have cameras and may produce images like in Figure 12. At present around 40 road weather stations are in use in Northern Norway, around 20 of these have cameras.

The Road weather stations are typically instrumented to monitor :

- Air temperature
- Road-surface temperature
- Precipitation
- Relative humidity
- Wind speed and direction
- Some stations has a camera

The observations are sent to the nearest Road Traffic central (VTS) and are used for road maintenance planning and traffic monitoring. For remote stations GSM/GPRS is used for transmitting observations, but the stations may also use many other communications channels. Dependent on location, both landline communication and power from the electrical grid may be available.

Some of the stations provided by ScanMatic uses the QNX Real time operating system (RTOS) [17].

Figure 10: Hardware layout of the Norwegian Polar Automatic station

DSU, Data storage unit

DSU contr. Data storage unit controller.

Trykksensor, Pressure sensor.

SU enhet. Sensor unit (wind)

Logger 3010. Data logger and main computing unit.

mV converter and Gust adapter. Part of the wind measuring sensors.

Figure 11: Example of Road weather station. ScanMatic

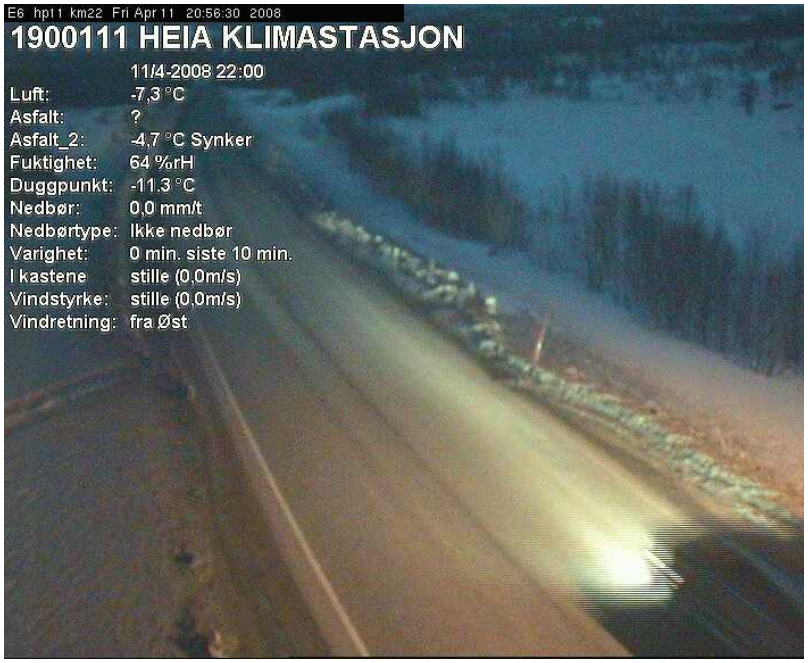

Figure 12: Example of image produced by one Road Weather station

5 Architecture

The goal of a system architecture is to do a decomposition and partition the system into manageable "chunks" of components. Ideally there should be a loose coupling between components, and clean interfaces should be identified.

The architecture of the system in this dissertation must support the deployment, running, updating and data delivery in a sensor network. The architecture must support a complete "Sensor network" consisting of **controlling**, **computing** and **producing** logical networks overlayed the physical network of nodes (see also Figure 4). The architecture must allow several different networks to operate on any node without conflicts and with a efficient use of resources.

The external interface of the "Sensor network" is basically two interfaces: The Control and the Data computing interface. Physically these interfaces would be implemented on Gateways into the "Sensor network".

5.1 The networks

5.1.1 Data producing network

The producing network handles the main goal of most sensor networks. This network will produce the observations used by the other parts. This network consists of a network of cooperating nodes that :

- Collect sensor observations at a given frequency.
- Transmits the wanted part of the sensor observations to a network gateway.
- Participate in the communication of other nodes observations to the gateway.
- Report capabilities available at the node.
- Can be updated with new software or configuration.
- Be queried by other parts of the sensor network.

As an "observation" any data type or set of data types are possible. Both reading from individual sensors to video streams is in principle possible. In this dissertation these data types will be restricted to non-streaming types.

5.1.2 Data computing network

The data computing network consists of nodes that participate in handing of observations coming from the data producing networks. This can be as storage nodes or as nodes aggregating observations from several nodes. The data computing nodes can therefore also be part of the data producing networks if new data types are produced.

The computing network consists of cooperating nodes that :

- Receive observations from the producing network.

- Processes observations.
- Receives and handles queries from other parts of the sensor network
- Transmits the wanted part of the processed observations to a network gateway and/or functions as a network gateway.

The computing network may also be queried by external users directly.

Data computing in a sensor network may be several different types. Some of the possible tasks are listed below.

- Data storage.
- Sensor data aggregation.
- Feature extraction from multiple sensor nodes.

In this dissertation only one task for the data computing network was required: **Data storage**.

The sensor data computing network has to report at least one node as a data sink for the sensor data producing network. This data sink may be several cooperating nodes, and where the data is delivered should not influence its use. The data sink must store and make available all reported observations.

5.1.3 Sensor control network

The sensor control network must assure that all types of nodes are started, running and stopped on request. New nodes may be added as they become available. Nodes must be updated with new software or new configurations.

The controlling network may be distributed on several cooperating general nodes and may also have special nodes functioning as gateways to the sensor network. These gateways has to be visible and available externally to the sensor network.

The sensor network control has basically two tasks. It has to handle externally initiated actions, like software updates or changes in observation frequency. The network also has to handle internal housekeeping. One of the major tasks is to provide a service discovery functionality for the other parts of the sensor network. The control has to maintain an updated list of which nodes that provide which services.

The control network also has to decide on the wanted topology of its own network. The control network has to assume that the data computing network maintain it own topology and is capable of reporting at least one data sink to the data producing network. The sensor control network must maintain the status of such common resources for all parts of the networks.

5.2 Emulation control

In this dissertation a system was emulated using a computer cluster. There is therefore a need for a component that can deploy, start and stop the emulation. This may be a single program on a single computer outside of the cluster.

5.3 Client - Server

As previously stated any node may be both a data producing and data computing node. The individual nodes may also be part of the controlling network. This possible distribution of responsibilities on each node indicates that each node may both function as a client in other nodes/networks and as a server. A request may also cascade in a multi-tier fashion. In some respects the nodes also function as routers for data from other nodes on their way to the designated data sink. The whole sensor network may therefore be defined as composed of several Client-Server relationships and the Client-Server paradigm is the best description of the architecture.

5.4 Communication and Protocols

Since the nodes must communicate both with other nodes and with external users, the HTTP protocol was chosen as the main transport between nodes. Since the nodes also has to implement RPC like functionality, the Representational State Transfer (REST) [18] and [19] architecture for resource location and identification, were chosen as a basis for the architecture. The use of TCP/IP makes it easy to have nodes that are reachable from the internet if wanted. Also the use of HTTP makes this access much more in line with the current trends in technology where more of the functionality is placed in the web browser.

5.4.1 Software update

In this dissertation only a very simple software distribution algorithm is expected. The goal is to have a reliable distribution from node to node, where all nodes are reached in finite time. The software update is expected to consist of one single packet. This packet may be an archive file and may therefore contain multiple files.

5.4.2 Routing

There exists several good algorithms for routing traffic in wireless sensor networks. Both three-based as one option in TinyOS [20], and fully meshed as in the systems produced by Dust Networks [10]. In this dissertation routing is considered an issue, and this dissertation therefore assumes that such algorithms exist and may be applied. Only a simple controlled routing is expected. The goal is to have a mechanism where observations may be routed through other nodes to a data sink or directly to the data sink. Also a simple routing of broadcasts and software updates is expected.

Today we can buy a complete wireless network for sensor networks, where the user has to provide a microprocessor to put next to the network node that communicates by sending and receiving packets over a serial port. (One example is Dust Networks systems [21]). The network component will handle all communication and routing issues and it does provide very high reliability of the end-to-end network connectivity. Any node can communicate with any

other node. In practice the user has to think about power consumption and the efficiency of the application protocols.

5.5 Partitioning of the sensor network

For data collection and software updates the sensor network may be partitioned into smaller parts,and therefore more manageable parts. A typical plan for partitioning the sensor networks rely in the minimization of a cost function. Often the cost is closely related to the energy requirements for data transmission and updates. Some kind of clustering is often used, either based on location as the needed transmission powers is reduced by distance, or some other metric.

In this dissertation the need for partitioning and routing efficiency will not be addressed. An efficient protocol is assumed to exist and would have been utilized in a real world deployment. Several studies have been conducted in this field. See as an example Akkaya and Younis [22] for a survey of routing protocols.

5.5.1 Time

There exist at least one good example of a solution for keeping a common clock in a wireless sensor network. This is implemented in Dust Networks Time Synchronized Mesh Protocol, TSMP [10]. In this dissertation the presence of a common clock with sufficient accuracy is expected. An accuracy in the order the resolution of the time-stamp of the filesystem is sufficient in this emulation.

5.6 Service discovery

Both for internal and external use the existence of a service discovery function was assumed. In the external world this may be a simple web site giving the correct URL to access the gateways for the sensor network. One can also envision systems where a new user at a site is automatically aware of the available sensor networks at the same site.

For internal use in the sensor network there has to be a function where a new node is registered and made available to other components. This function has to be an integral part of the network environment for the nodes.

There exists several solutions for such functionality and some of these are described and discussed below.

In this dissertation Bonjour [23] (also known as Zeroconf [24] or Avahi [25]) was selected for Service Discovery. One of the reasons is that it is possible to implement in very small devices. In the SitePlayer [26], a working Bonjour service has been implemented in around 800 bytes of code. See [27] for a motivating talk by Steven Cheshire on Bonjour. Bonjour does not depend on any servers available in the network. The local nodes communicate and cooperate to establish network names and make available services known. The choice of using the HTTP protocol as the main transport, also is a good match as Bonjour is supported in several web browsers, and servers announcing HTTP support will show up automatically. Bonjour is also implemented in various languages, and on several operating systems.

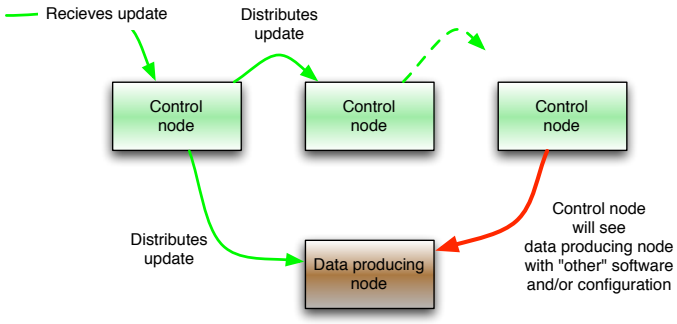

Figure 13: Potential problem in node control

In existing sensor networks service discovery is typical an integral part of the networking middleware. Kuorilehto, Hännilkäinen and Hämäläinen [28] have surveyed several WSN systems, and have noted that service discovery is often an functionality added in the middleware. No definite industry standard have emerged yet.

5.6.1 Coordination of control

One aspect of a controlling network is to coordinate between the cooperating nodes. One example of a problem may be illustrated in Figure 13

If multiple sensor control nodes see the same data producing or data computing node, there is a risk that during software or configuration updates, one or more control nodes have a different view on what is the current "correct" configuration of a node. It is therefore possible to have a situation where an updated node is "downgraded" to older configuration by nodes that do not have the most updated software or configuration.

In this dissertation the architecture was defined so that any controlling node, controls all data computing and data producing nodes.

5.7 The components

On each node different parts may be running. As illustrated in Figure 14 we are building on a platform that provides most basic OS services, at least networking, local storage and optional access to sensor hardware or readings.

The local architecture for a node in the Sensor Data producing network may be illustrated in Figure 15

5.8 The external interface

Access to sensor networks need to be simple both for computer systems accessing the observations but also for external human users. Both of theses concerns can be satisfied using HTTP servers as gateways to the networks. Since we only envision two main interfaces for external access, this can be implemented on one

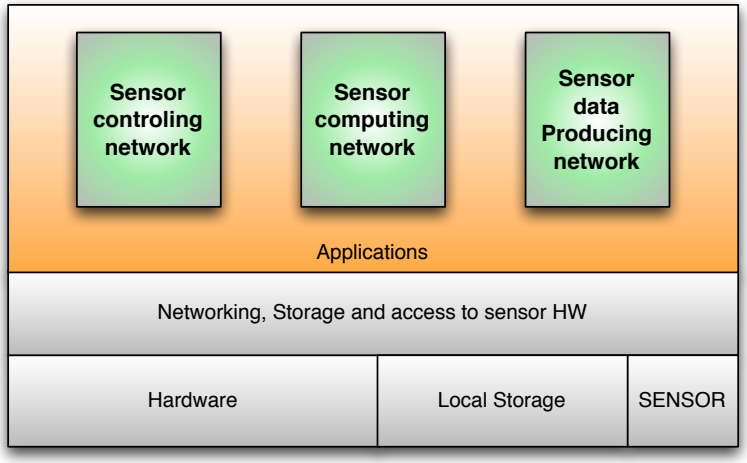

Figure 14: Architecture of a node. The functional networks are applications running in user-space on top of operating system services and hardware.

physical node, but it has to be part of both the controlling and computing network. External access directly to individual sensor nodes may be possible, but the gateways should provide the access to these nodes. This may be illustrated in Figure 16

5.9 Scale

One aspect of sensor networks is the ability to scale when going from a small number of nodes to thousands. The traditional meteorological networks mostly depends on a central control and data storage model, where all data is collected on one single location. Observations go straight from the observing node to the collecting central node. This model is difficult to scale when the number of nodes increases.

In this dissertation functional networks was used that may scale well as the number of nodes increases. The main reason for this is the possibility to dynamically add nodes running the sensor control and sensor computation networks.

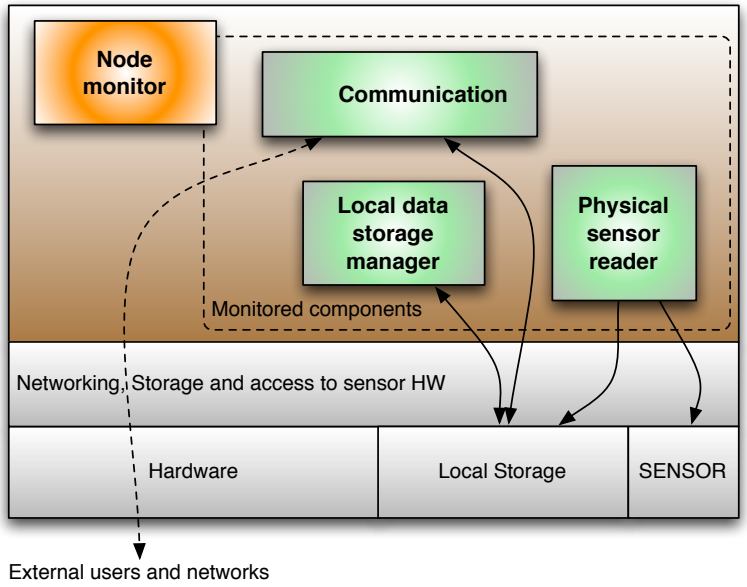

Figure 15: Sensor data producing node components

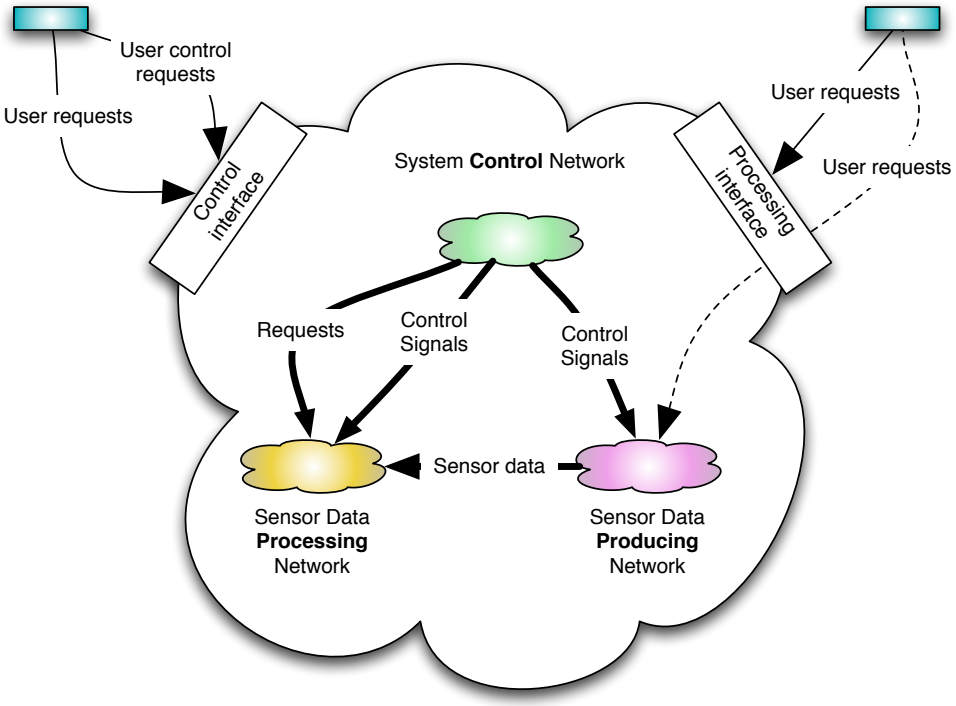

Figure 16: External interfaces and internal messages

6 Design

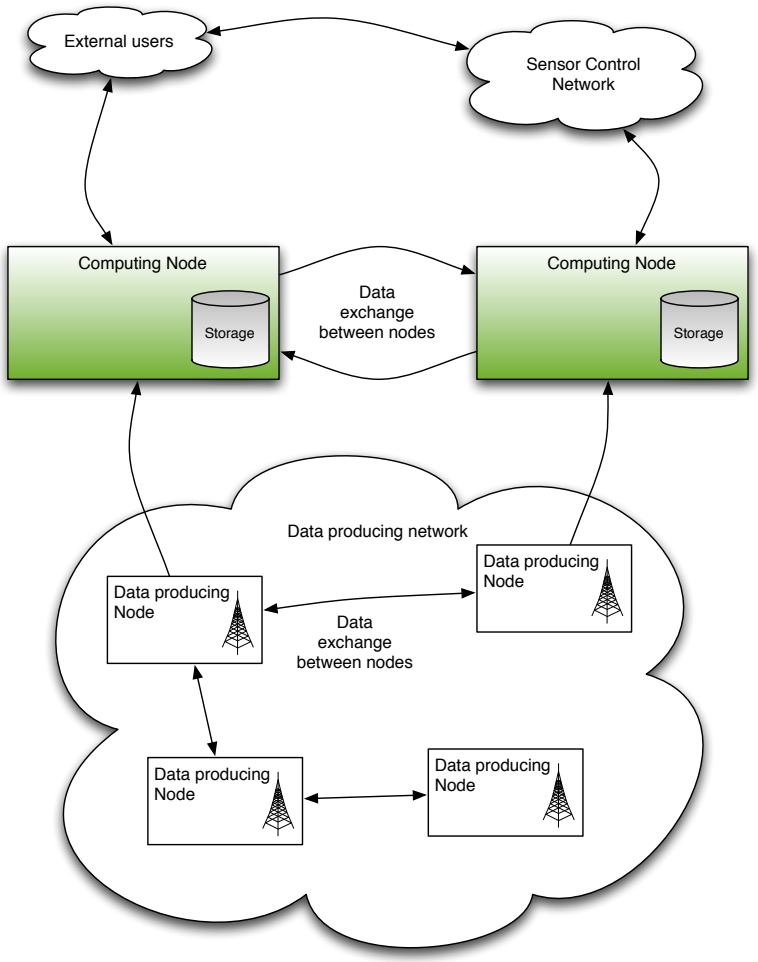

Figure 17: The data computing network

6.1 Communications

The architecture calls for a total network consisting of three networks of cooperating nodes where some are sensor nodes and a few are gateways for external access to the sensor network. This gives a natural division in the needed communications protocols.

The external interface will provide a subset of the internal interface, to ensure efficient code and practice reuse.

6.1.1 Protocols

The two main interfaces are against the controlling and data computing networks. The interface of choice is HTTP servers using the REST style architecture. This design define the structure of the interface, and the data types to be exchanged.

6.2 The data producing network

The data producing network consists of several cooperating sensor nodes. Each node is assumed to be observing some phenomena at some distance from other nodes.

The emulated sensor node is emulated by having each node retrieve some data from an external URL. Typically this is image data from web cameras. The primary source of such images in this dissertation is the web cameras operated by the Norwegian Road authorities around various parts of Northern Norway. An example is shown in Figure 12. The images may contain measurements from other sensors at the site, like temperature and precipitation. These images represent real sensors in practical use, and are a good example of actual data from such stations. The system has non the less to assume any form of data. The stations data are publicly available at `http://www.smallsoft.com/klima/Klimadat.htm`.

6.2.1 Node start up

The emulated sensor nodes need to read the local configuration and initiate the local computing with communication, physical sensor handling and data computing. See also Figure 15 with a schematic view of the components in each node. The node has four basic components.

- Node monitor. To ensure restart after updates or crashes.
- Communication. With other nodes in the network and a data sink.
- Local data store manager. The emulated node is assumed to have some data storage available. This has to be managed for shared use between components.
- Sensor readings. Physical or as in our case, emulated sensor data must be collected at a wanted frequency and the data stored locally on the node.

The emulated sensor node has to register with the local network if such a network is in place. As part of the registration information on local capabilities on the node must also be sent. The sensor needs to establish the next node in the path to the data delivery node (data sink). This is assumed to be done in the registration phase. The registration uses the service discovery function, which should return the complete network address of a data storage node, or a list of such addresses.

On start up, the sensor node needs to establish its own environment. In the emulated sensor nodes this is done using a local configuration file. This file

contains a descriptions of what type of sensor it is to emulate. A small sample of such an configuration file is given in Figure 19.

The format of the configuration file is in the form of:

```
[SECTION]
name=value
```

The file has multiple sections and multiple name/value pairs in each section. This format is used in many systems. One example is in Windows ini-files. There exists several libraries to read such files for multiple computer languages.

The content of the configuration file has to include at least the following elements:

mime Any known types of data may be observed and the type is given using the MIME type defined in numerous internet standards (RFCs : 2045, 2046, 2047, 4288, 4289 and 2077)

name A short string defining each node. In this dissertation a name describing the location where the image is taken. A human-readable name is wanted.

url The URL used to fetch the emulated observation. Any valid URL can be used.

frequency The wanted observing frequency at this node. In seconds.

6.2.2 Node Monitoring

Since the node software may fail due to errors or bugs, a simple process monitoring utility is used. The utility must differentiate between failures and a planed shutdown.

The process monitoring will also have to restart the node program. In the case of repeated software crashes, the node has to start from a known good software base, and not from the last running state. Normally, simply restarting will use updated configuration or software.

In the emulated system, the process monitoring has to run as a process of it selves, so that it does not stop if the other software crashes.

6.2.3 Sensor Data protocol

The data producing sensor nodes needs several communication parts. The sensor node needs to respond to external request for data/observations. This is done using a simple HTTP server. The use of the REST [18] style imposes a clear structure on the way URLs are built up, but requires an established protocol. See also Appendix C for examples.

The following describes the protocol for retrieving data from a sensor data producing node, and also the protocol for another sensor data producing node to deliver data for forwarding to a data storage.

Descriptor	Description
GET	Request for sensor data, configuration information or a default page
POST	Inserts new software or configuration and also data for forwarding to the master node.

The default page is a simple web page that the sensor node will use to deliver a human readable description of the node and available protocol. The same protocol is used for human users and data retrieval from other computer systems.

Descriptor	Description
GET	HTTP://<address:port> <no option>or INDEX.HTML CONFIG <Section><Parameter><Value> <sensor type> keys textkeys data <key >
	RESTART KILL
POST	HTTP://<address:port>/ /DATA /SOFTWARE

INDEX.HTML or no options gives a default page from each sensor that describes the current configuration and the available options.

CONFIG allows for setting configuration parameters by specifying the Section, Parameter and wanted Value.

<SENSOR TYPE> Paths like "IMAGE" gives the default data value from the sensor if the sensor is of "IMAGE" type. This way the server supports any data types.

keys Returns a HTML page with a list of all data elements available at the node. The list is for human use and the keys should be clickable for easy retrieval of one data element.

textkeys Returns a plain text list of all data elements available at the node. The list is to make it easy for other computer systems to retrieve and decode.

data If this path is followed by a valid key, the corresponding data element from the local data storage will be returned.

RESTART and KILL Used for controlling the node. See chapter 6.4

POST /DATA post sensor data from another node for forwarding to the master node.

POST /SOFTWARE post a new data element containing the update to the node. Both configuration and software may be updated using this function. The smallest element to be updated is a file. The data element posted may also include routing information for the update. Updates containing multiple files may be sent as one single gziped tar archive.

6.2.4 Data delivery

New items observed is delivered towards the defined data storage node. This may be an intermediate node in the network. The data is delivered using the same simple HTTP POST protocol as described for data delivery between sensor data producing nodes.

6.2.5 Data forwarding

Any data sent to the sensor node using HTTP POST is forwarded to the nodeś master node. The sensor data is posted to the "DATA" path. In effect this gives a very simple routing algorithm by simply transferring the incoming data, using one step further to the data storage. In this dissertation the routing is not an issue.

6.2.6 Data types

The actual data types used for exchanging sensor data between nodes is not defined in this design and is being referred to the implementation.

The definition of the data types used in the complete system would more naturally be defined in the system design phase if heterogenous implementations were expected, as multiple implementations would be possible afterwards. This decision is delayed to make the implementation simpler, but with the clear realization that this may introduce unwanted dependencies.

6.2.7 Software updates

The software or configuration files on the node may be updated. This is simply done by posting a single file to the node using HTTP POST and identified using the "SOFTWARE" path. This file may be an archive containing several files. This introduces the possibility of the sensor node to be in several states as described in Figure 18. This figure also describes some of the duties of the node monitor that has tasks on behalf of the sensor data node and on behalf of the controlling network. Included in the figure are also states that can be reached because of commands from the controlling network.

The update is assumed to arrive in one complete posting. This is consistent with the stateless architecture style of REST.

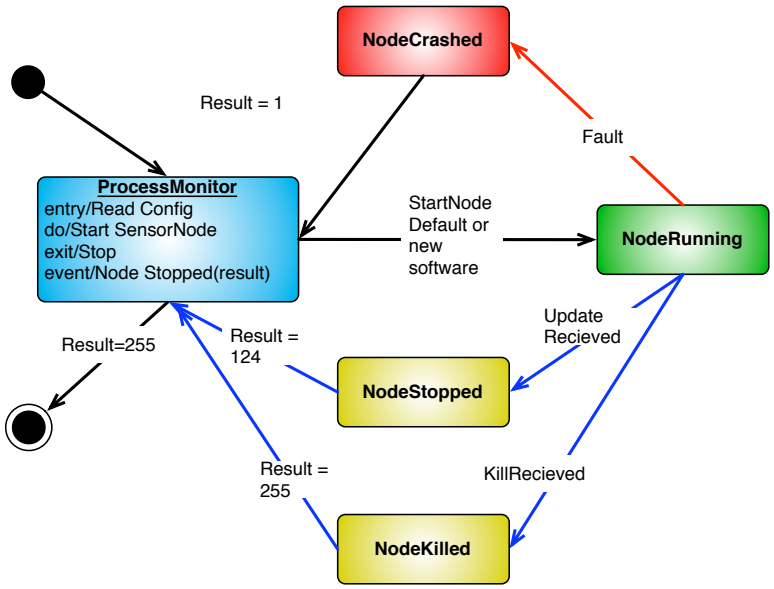

Figure 18: Sensor node state chart

6.2.8 Software update forwarding

Whenever a software update is received it may be forwarded to any known sub nodes using the same HTTP POST protocol. The further routing is controlled with the content of the posted data element and is therefore left to the controlling network to decide. The network may provide a broadcast option, but this is not used in this dissertation.

Since a software update will spread in the networks in single hops there will be a time where old and new software are active in the network at the same time. This implies that any software update have to ensure backward compatibility at all times. A simple scheme of versioning where any data or configuration also contains a version, must be implemented.

A small variation on this method is spreading the software update as a "Worm". The classical "Worm" is described by Shoch and Hupp [29]. A worm is capable of altering its state and strategy dependent on information collected from its segments. It may also migrate to new nodes and as an example, remove it selves from nodes with too high work load. This has the implication that decisions can be postponed and made when local information is available.

6.3 The sensor data computing network

As defined in the architecture the computing network in this dissertation must provide at least one data sink for the sensor data producing network.

6.3.1 Sensor Data Storage

The sensor network data storage makes the following assumptions :

- The data storage system will store all data elements posted to it.
- The system will use a simple database for this storage.
- The system will use a single key for locating the stored element. The key will be based on the sensor name and the current time.

The most frequent users of the service is :

- The sensor data producing network
- Other parts of the sensor data computing network.
- Individual users and processes.

The most frequent use cases are :

- The sensor producing network will send its observation for permanent storage. Also agregated values from the sensor data computing network may be stored.
- Both the sensor data computing network and individual users will read individual sensor observations.

6.3.2 Cooperating nodes

All nodes in the sensor data computing network have to cooperate to solve the given task. In our case this implies that a protocol for exchanging data between nodes must also be established.

Before such a protocol is designed, the overall goal of the system has to be known. In many cases a distributed system is used to ensure sufficient computing power, reliability and availability. In such cases a monitoring system has to be implemented to restart stopped nodes, and to notify cooperating nodes of the status.

In this dissertation the goal of the data computing network and the data storage task is to provide at least one data sink to the sensor data producing network. The solution is to have the nodes in the data computing network exchange all incoming data. Figure 17 illustrates the flow of observational and processed data.

6.3.3 Communication

A subset of the sensor node protocol is used for sending data to storage or retrieving data. The protocol is shown in chapter 6.2.3. Only the following items are needed at a data storage node:

- POST /DATA
- GET /data <key>
- GET /keys

- GET /textkeys
- GET /haskey

The protocol allows for different implementations of data handling. The only limitation is that the system must be able to handle the data elements that are chosen. See also chapter 6.2.6. This also has the effect that querying a sensor data producing node and a sensor computing node can be done using the same protocol.

To facilitate data exchange between nodes a simple expansion to the network protocol is used. The nodes do not need to send data to other nodes that the remote node already have received. So a test to explore if a node has a data item is needed in the form of the "GET /haskey" operator. The necessary protocol for the nodes in the data computing network, supporting only data storage is given below.

Descriptor	**Description**
GET	keys textkeys data <key > haskey <key >
	RESTART KILL
POST	HTTP://<address:port>/ /DATA /SOFTWARE

The "GET /haskey" operator must return "YES" in plain text if the node already has the data element with the given key and "NO" if not.

The data computing network relies on the service discovery to provide an overview of the available nodes in the network at all times, and to provide an overview of the capabilities of the nodes.

6.4 The control network

The sensor control network assumes the following requirements:

- The controller network must deploy and start the all software on the computer nodes.
- The controller network must be able to stop the whole sensor network.
- The controller network distributes software updates.
- The controller network monitors the state of the networks.

The ability to stop and start the whole sensor network is important in an emulated situation. During testing and development several errors can lead to runaway nodes and efficient means to stop these are needed.

6.4.1 Starting the emulated sensor network

Starting the emulated sensor network in the available computer cluster should be done using a simple distribution of all needed files to all nodes. The files are assumed to contain software and node configuration may be similar on many nodes. The controlling network has to do a remote start of all nodes.

The distribution of files containing software and configuration involves several tasks

- Reading a short configuration file containing the following information :
  - The files in the software and configuration package.
  - The machine names in the computer cluster to run software on.
  - The name of the individual sensor node configuration file.
  - The external URLs to use as emulated sensor data. Each sensor node monitors one external source.
- For each computer node to distribute to :
  - Create local directory for temporary storage of files.
  - Copy wanted files to this local directory
  - Adapt the sensor configuration file to the wanted node.
  - Using “scp” copy the files to the remote node
  - Start the node software using the Process Monitoring application.”ssh” is used for this.

This will make it possible to run the emulation on a computer cluster with a shared filesystem and with local filesystems on each computer.

The designed process above does use both "scp" and "ssh" or file transfer and process starting as this is well supported on the available computer cluster.

6.4.2 Registering newly started nodes

As each node is started it will use the service discovery service, chosen to be Bonjour, to register its name and capabilities. It will also query the service for an address of a data storage to use as a sink for its observational data.

The service discovery system has to contain all needed information for further handling.

6.4.3 Control functions that needs protocol support

- Restarting a node.
  - Each node may be sent a "STOP" command using HTTP GET /STOP. This will stop the node software, and the node will be restarted when the monitoring sensor program find that the node has stopped. The software has to stop with an error code that the monitoring process can detect. The value of "1" is selected for this purpose.

- Stopping a node permanently
  - Each node is sent a “KILL” command using HTTP GET /KILL. The node software must stop with an "255" error code, so that the process monitoring will not restart the sensor. The process monitor will also stop at this point.
- Updating software
  - Using a simple HTTP POST the software may be transfered to the node. All content needed for a complete update must be sent in one package, so that a node simply does a restart to activate new configuration or software. The package may be an archive and will be unpacked. This allows update of several files at once.

The software update is expected to be generated externally to the whole sensor network and in our emulation this is done using the controller in chapter 6.4.1.

6.4.4 Status monitoring

Each sensor data producer will deliver its data to the controller. This may registered as a simple heartbeat. Actions to restart a node may be taken if some time limit is exceeded. Alternatively this may be done using the service discovery system that also has to keep track of active nodes in the network.

7 Implementation

7.1 Programming environment

In this dissertation the following hardware and software developing platform was chosen

- Linux and OS X operating systems.
- Python language with standard modules.
- TCP/IP local network between nodes.
- A cluster of computers running Linux.
- Linux and OS X workstations.

Python has many modules for most communication protocols, so there is no need to implement such protocols from scratch, and newer protocols could be based on existing standards. Python is also available on all relevant operating systems with almost all of the same modules.

7.2 Threads vs Processes

Several prototypes of the sensor producing nodes were built. One of the identified problems, was the choice between using a single large application on each node with several threads for the different tasks, or using multiple processes and splitting the tasks between them.

Using multiple threads inside a single process was found to lead to complex coordination problems, and the use of a large number of queues for delivering data to, and exchange between threads. The synchronization between different parts of the application was difficult. It was also difficult to keep a good overview during coding, due to the size.

In the meteorological weather station we are emulating, the different functions are divided between several hardware units. See Figure 10 for an image of the physical layout and partitioning. The emulated system would also more closely resemble the actual systems, if software are partitioned more or less in several processes with few dependencies between them.

The sensor producing nodes were therefore partitioned into several cooperating processes with the following parts.

- Sensor reader process. Single threaded.
  - Reads a local configuration file containing the URL and frequency of observations. An example is given in Figure 19.
  - Reads the values from the sensor. In our case, download an item from the URL. The item may be an image.
  - Stores the values in a local file on the node.
- Sensor data server process. Multi threaded.

  - Registers the node with the service discovery function and get the address of the data sink.
  - Reads the stored resource and send to the data sink at the wanted frequency.
  - Provides an HTTP server for other nodes to POST data elements to and to query for stored data.
  - Provides an HTTP server for other nodes to POST software updates to.

- Sensor monitor process. Single threaded.
  - Reads a configuration file and start all wanted processes.
  - Monitors all started processes.
  - Restarts stopped processes or stop all processes dependent of the status received from the stopped services. If all processes are stopped, it will also stop it selves.
  - The process monitor can not be updated and must therefore be very simple and efficient.

```
[Sensor]
mime=image/jpeg
name=Andenes-Ramnan
url=http://alomar.rocketrange.no/images/IAPweb.jpg
frequency=60.0
```

Figure 19: Example of sensor node configuration file

Threads will still be used in the Sensor data server process as it has to do at least two different tasks in parallel. It has to send observations (items) to the data sink, and also maintain a local HTTP server. With the limited number of task, it is easy to keep a good overview of handling and synchronization of multiple threads. A simple queue is used to send items between the threads. The only use is when another node does a POST with data to this node. The data element is put in a queue so that the thread that does the sending of data, can pick it up and send it towards the data sink.

7.3 Data types

With the use of Python as the programming language, the data types to be exchanged between nodes and for local storage, were selected for programming ease. The simplest was to use a python dictionary. The use of a dictionary allows us to add new fields or keys as the use is expanded.

For simple storage and transfer between nodes, the dictionary containing the observation or other data is converted into a string representation using Pythons **Pickle** module. The main benefit is that the structure of the dictionary does

not need to be known on the receivers side, as pickle will return the dictionary when used upon the received string. This also applies for stored items on a local file or database.

A pickled dictionary is not easily readable by other languages or systems. The most useful approach would be to embed the python interpreter in a C program. This also has the effect that all HTTP servers serving external requests for data, must return the requested data item in its original form. For exchange between cooperating nodes or processes within the whole sensor networks, a pickled dictionary was used.

A pickled item transfered between two nodes is not to be considered a secure transfer. It is possible and quite easy to insert or change items in the data stream. This is underlined in the Python documentation that states:

Warning: The pickle module is not intended to be secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

7.4 Local data storage

For local data storage a simple database is needed. The design specifies that any data item that is to be stored is identified with only a simple key. The key is a concatenation of the node name and the time stamp for the observation in a simple text form without blanks.

Python has several options for persistent storage. The choice in this dissertation was to use the **anydbm** module. After opening a database file, the content can be handled in the same way as any python dictionary with access to the *has_key* and *keys* methods. The key to the database is any simple string, and any string can be stored in the database. Therefore the combination of the anydbm module and python pickled objects fits the task well.

7.4.1 Problems and bugs

During the work with this dissertation, Apple released the 10.5 version of OS X and Python was upgraded from version 2.4 to version 2.5. For most parts this did not have any impact on the code already produced. The cluster were running version 2.4 of Python.

On OS X version 10.5 an error in the **anydbm** module was found. When opening an existing database with many stored elements, the database would not report the correct number of keys available. The items not reported could nevertheless be accessed. This bug was reported to Apple with problem id "5606751" and it is still an open report.

To work around this problem a simple module was developed that mimicked the functionality of the "anydbm" module. This module was called **mydbm**. This module simply stores any data element as a file in a specially named directory using the given key as the filename. This module therefore have the same limitations as the filesystems the program is running on, but this was not considered a problem for this emulation. An added benefit was that the stored

items could be accessed outside of the program and therefore make debugging easier.

7.5 Naming

In this dissertation the sensor nodes are assumed to be named by the network used. This gives the nodes on the computer cluster and desktops a name of the type "tile-0-0.local". The general form of the name i "machine"."local" where machine is the name given to the node during installation of the operating system. The computer cluster in use is Display Wall [30] cluster at the Institute of Computer Science. The 28 computers controlling the 28 projectors are named after the model "`tile-<column> - <row> `", like : "tile-0-0", "tile-0-1" and "tile-7-3". The ".local" domain is the default domain given using the Bonjour/Avahi systems. (See also chapter 9.7.3). The desktop and laptops used have names in the same way.

To get the name of the local nodes the following python code was used:

```
node_name = os.uname()[1]
```

This uses the standard OS module and the "uname" function. This function is only available on recent UNIX-like operating systems. Under OSX 10.5 element "[1]" of this tuple is "Epsilon.local", the name of the Macbook laptop.

7.6 HTTP server

For all different nodes the "**BaseHTTPServer**" python module is used for the HTTP server. This module provides a very simple programming interface, with the web server running in its own thread, and provides an easy interface to implement the REST style URL handling. To start and run the HTTP server, only a simple code is used :

```
httpd = BaseHTTPServer.HTTPServer(("", PORT), Handler )
httpd.serve_forever()
```

"Handler" is one instance of the **BaseHTTPServer.BaseHTTPRequestHandler** class.

The handler class only has to implement the functions "do_GET" and "do_POST" to handle these HTTP events. The REST style is followed as a resource is named in the URL and the program is responsible for returning the correct representation of this resource. In these HTTP servers, accessing and resource through an URL is also similar to using an RPC call to the node.

7.7 Time

The meteorological demands for clock accuracy, removes the need for higher level of synchronization than provided by the standard clock on the computers in the cluster. The computers all use a common timeserver and the standard "ntp" protocol for keeping a common time.

7.8 Sensor Lookup

The Bonjour/Zeroconf/Avahi technology was due to some technical restrictions, not available on the computer cluster in use for this emulation. So to observe the benefits of easy access and self configuration, an extra module was needed. The goal was to have each sensor node discover the communication path to the data server. This would have been possible using Bonjour on the nodes, but the sensor lookup server provides some of the functionality, and acts as a small gateway. If a sensor data computing node can not use Bonjour, because it is running on the computer cluster, it will use the sensor lookup server to register.

The sensor lookup server runs outside the computer cluster and registers with the local network using Bonjour. In most cases the server was running on the author's MacBook. This makes the sensor lookup server available as a HTTP server to all Bonjour capable devices on the local network as shown in Figure 20. This gives any process or user access to a gateway to the whole sensor network emulation, without the need for further configurations of any device. No access control has been implemented.

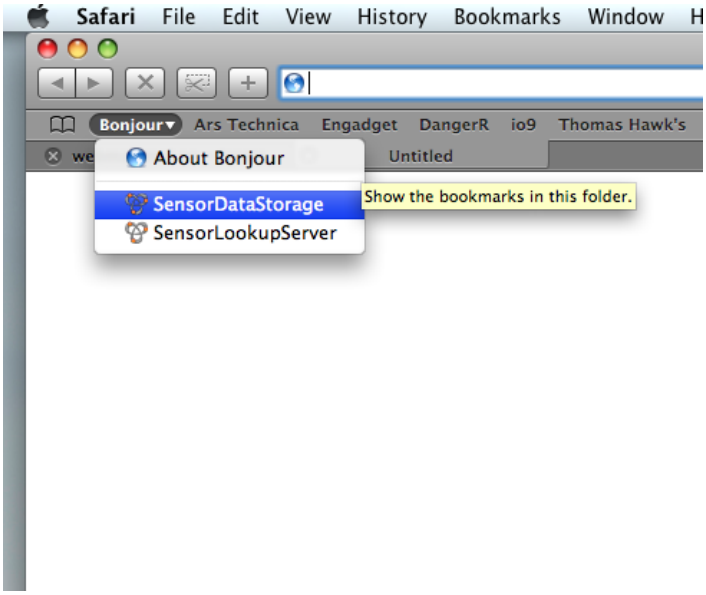

Figure 20: Sensor Lookup and a Data server node available through Bonjour

Selecting the "Sensor lookup" will access a small http server. This provides the user with further links to the individual sensor nodes as show in Figure 21.

On the sensor lookup server web page, simple commands can also be given to the server. The most frequently used is the "KILL" command for stopping the server.

The individual links to the nodes provides access directly to all nodes presently known to the lookup server. The "QUERY DATASTORAGE" link gives a simple text line with the known location of a Sensor Data Storage node. This functionality is mostly used by the data producing nodes to locate a data storage

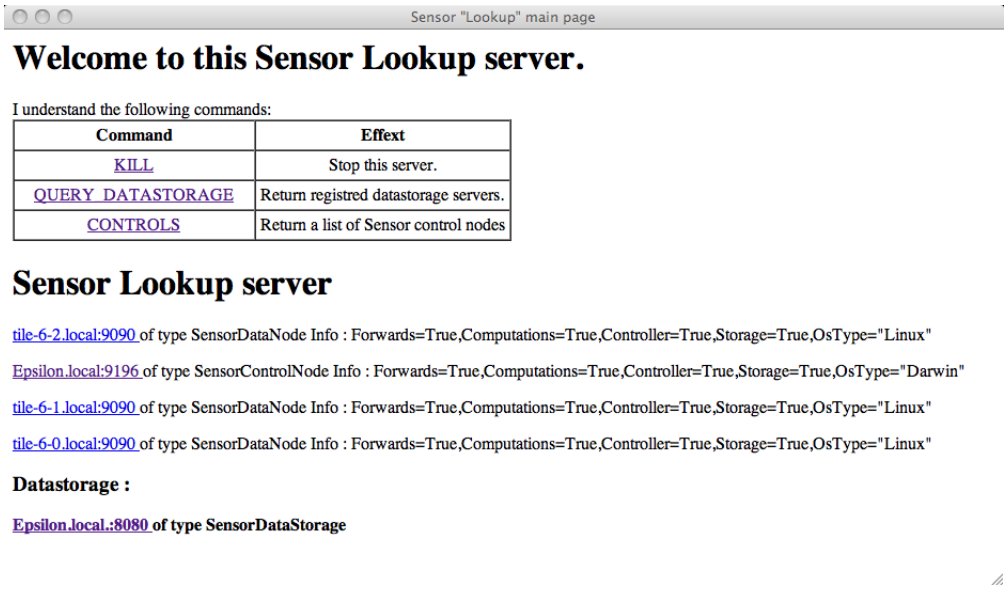

Figure 21: Sensor Lookup HTTP server

server, the data sink. The "CONTROLS" link returns a simple list of registered sensor control nodes.

The Sensor Lookup server has two separate modes of operation. Any node will broadcast a request for the address of the Sensor Lookup server using a "well known" port and UDP. This port was selected to be port **54666**. All communication with the Sensor Lookup server uses the same common routines placed in a small python module, and this keeps this port and the protocol only defined one place. The broadcast from the nodes just contains the word "*HEI*" and is not used for further reference. The Sensor Lookup server replies with the location and port number to use for further communication. The nodes will after this, use the simple http server on the lookup server for registering, de-registering and query of location of a data storage server node. The nodes may also register capabilities in form of a text string with name=value pairs describing the node.

The Sensor Lookup Server must run at all times and be available at all times. This makes this server a single point of failure. This is a weakness of the actual implementation. Using Bonjour for this service would distribute the functionality to all participating nodes, as information would be registered with all nodes and also cached. See also chapter 9.7.3 for some details and references.

The protocol used for registering a node on the Sensor Lookup server is illustrated in Figure 22 and a description of the details are shown in Table 2.

A more complete overview of the communication between various types of nodes is illustrated in Figure 23.

The nodes will query for the location of the Sensor Lookup server before information is retrieved from the server. This will handle a situation where the server has been restarted on a new node after crash or other stops. The

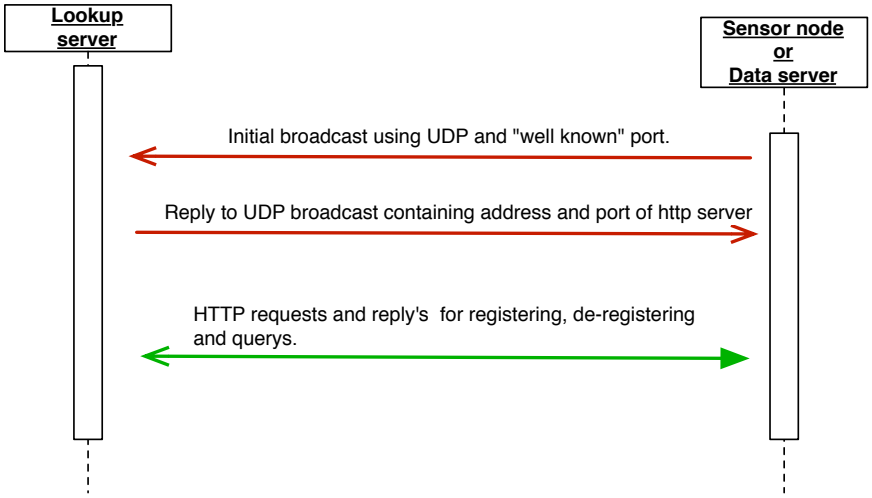

Figure 22: Sensor Lookup communications

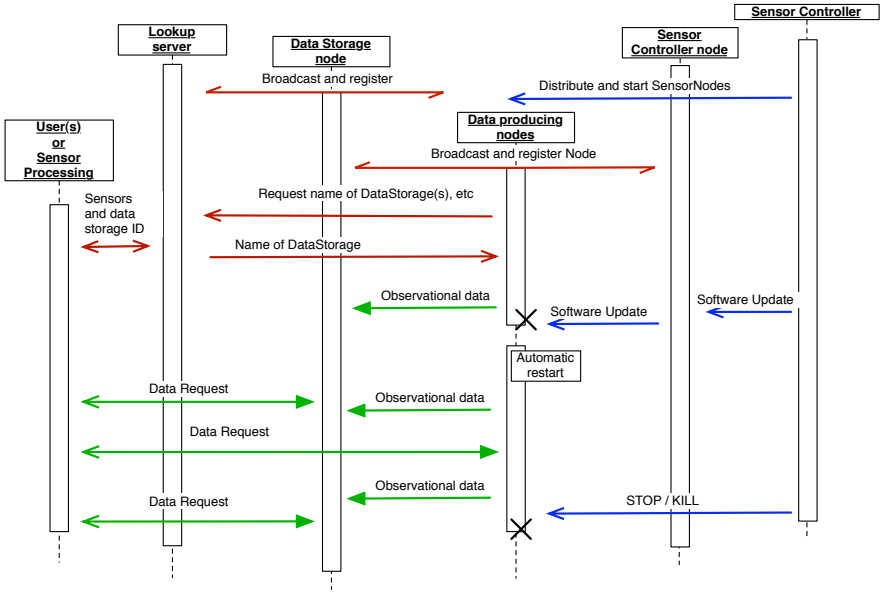

Figure 23: Messages exchanged with the Sensor Lookup server

Table 2: Sensor lookup protocol

Descriptor	Explanation
KILL STOP	Stops the sensor lookup server Stops the sensor lookup server Both these will halt the server.
REGISTER REMOVE	Register a node with the server Remove a registration with the server Both these will POST a python pickled dictionary.
QUERY DATASTORAGE	The address of the registered storage servers This will return lines with each node on one line.
CONTROLS	Query the address of registered sensor control nodes This will return lines with each node on one line

nodes have a limited number of retries before they give up and has to stop the computing on the node. In our emulation, 10 retries.

7.9 Node monitoring and structure of a data producing node

Each node consists of three running processes and a shared filesystem as illustrated in Figure 24

- Process monitor. Will start the other processes initially and after a crash or software update.
- SensorReader. Retrieves one observed item from the simulated sensor and stores this on the shared filesystem. The observing frequency is set using a configuration file.
- SensorDataServer. The Data server has the communication responsibility for the node. This server will at some frequency dispatch the latest observed value to a data storage and will respond to any users accessing the node directly. This node also handles software updates.

The shared filesystem in this dissertation is a simple file on the local disk on the node.

The process monitor differs between stopped and crashed software on the resulting error code the operating system returns. The process monitor is implemented as a separate program that starts and monitors that the sensor is running. When the sensor stops, two different options exist. The first is simply to restart the sensor program, the second is to terminate all computing at the data producing node if this is wanted. Since the process monitor has started all cooperating processes it also knows the process ID, and can stop them. All processes run as regular user processes.

The node does not report stopped or crashed software in our implementation. This could be implemented by having the Process monitor create a report file that the SensorDataServer reports as part of the registration. This report would also contain version numbers if this is available.

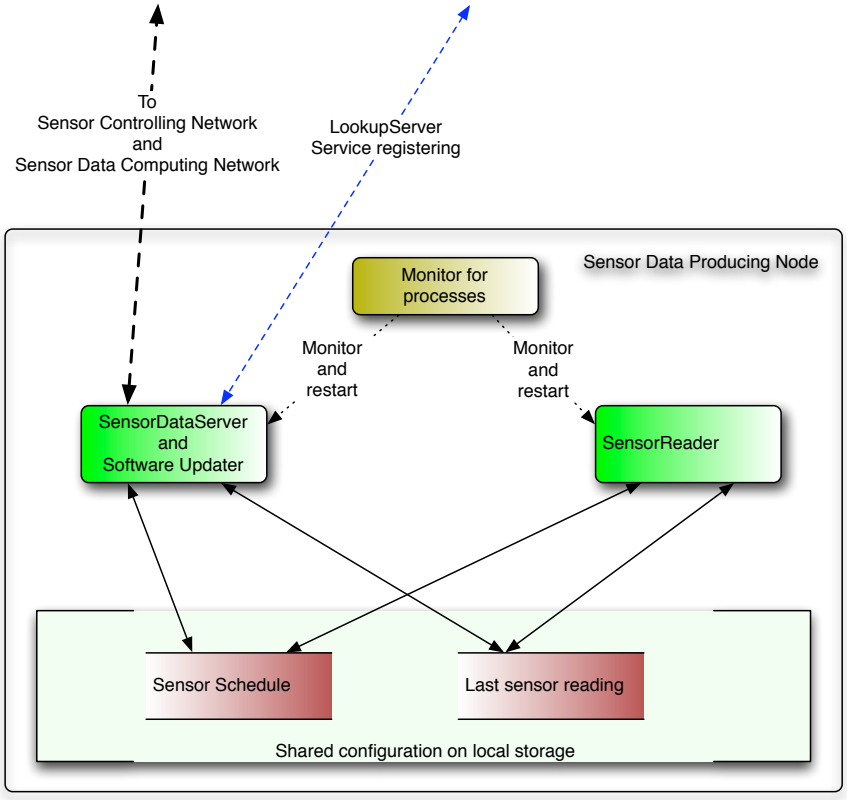

Figure 24: Data producing node with monitor

7.9.1 Size of the software at each node

The sensor data producing node is programmed in python, and the size of the different files to be transfered on deployment of the node adds up to a total size of around 40 Kb.

7.10 Design of Sensor Data Storage node

A sensor data storage node is part of the sensor data computing network. A simple use case analysis shows that the most common use is to store and retrieve single elements from the data storage. The data storage uses the following technical solutions:

- The "mydbm" python module is used for database-like storage. See chapter 7.4.1
- The stored data is a python pickled dictionary containing at least the following keys:
  - name, identity of the data producing node sending the data
  - timestamp, of the datavalues sent

- data, is the actual data element
- mime, is the mime "Content type" of the data element. "image/jpeg" is used for webcamera images.

The pickled dictionary is stored in the database under the key generated by concatenation the name and the timestamp. Any type of data value can be stored as long as it can be pickled. The storage does not know the type, it simply stores a long string. The dictionary is also human readable, to ease debugging. A binary representation would have made the storage requirements smaller, but this was not an issue in this emulation.

7.10.1 Communication

The data storage node provides an HTTP server for posting incoming data from data producing nodes and also from other cooperating data storage nodes. Data can also be retrieved from the data storage using the HTTP server. The protocol has been described in chapter 6.3.3.

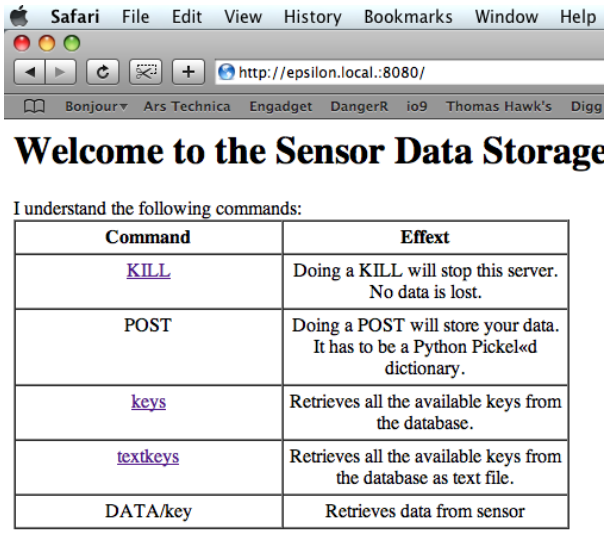

Figure 25: Sensor Data server, viewed from a browser

The HTTP Server is available from a standard web browser as illustrated in Figure 25. The HTTP server has a simple page-design and may also either show all available keys in the local data storage as a web page (Figure 26) or in a pure textual form that is more suitable for further computing by other applications. See chapter 7.13 for an example that uses this functionality.

For communications between nodes in the data computing network, a node needs to keep a list of all peers. Using the basic assumption that all data elements should be available from all nodes in the data computing network, a simple routing of data between nodes is needed. In an real-world sensor network this protocol would be much dependent on the characteristics of the

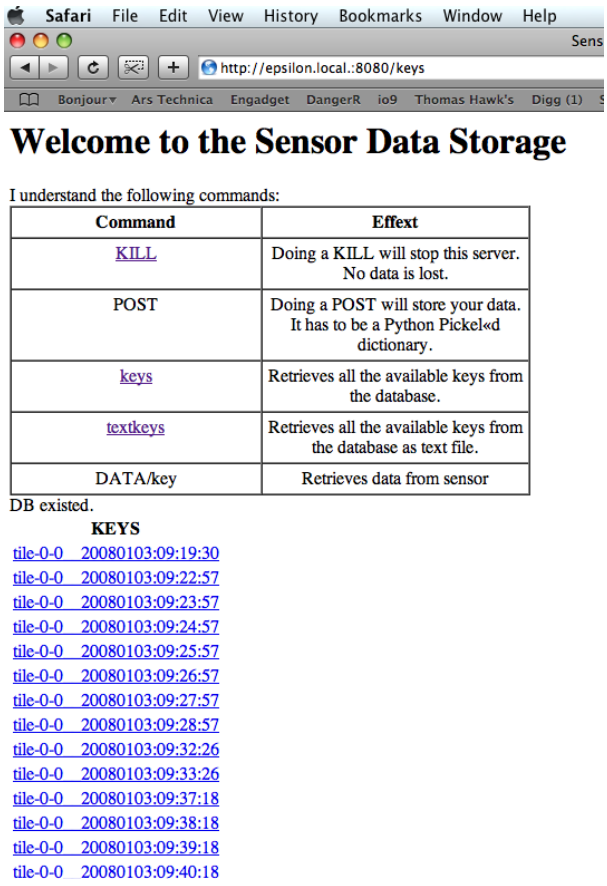

Figure 26: Sensor Data server with link to all stored items

actual network in use, and would have to be as efficient as possible. In our case we can implement a simple worm-like distribution, with the same functionality as the software distribution.

The data element is extended with a simple list of nodes to receive the data. This list is generated by querying the service discovery for nodes of the correct type using the protocol in chapter 7.8

7.11 Sensor controlling network

7.11.1 Emulation start, stop and update

A simple controlling GUI for starting and stopping the emulation on the computer cluster is created to run on a Macbook laptop. The program is created using python and TK for GUI. This makes the software portable to all platforms. This software is called the "controller". An running example is show in figure 27.

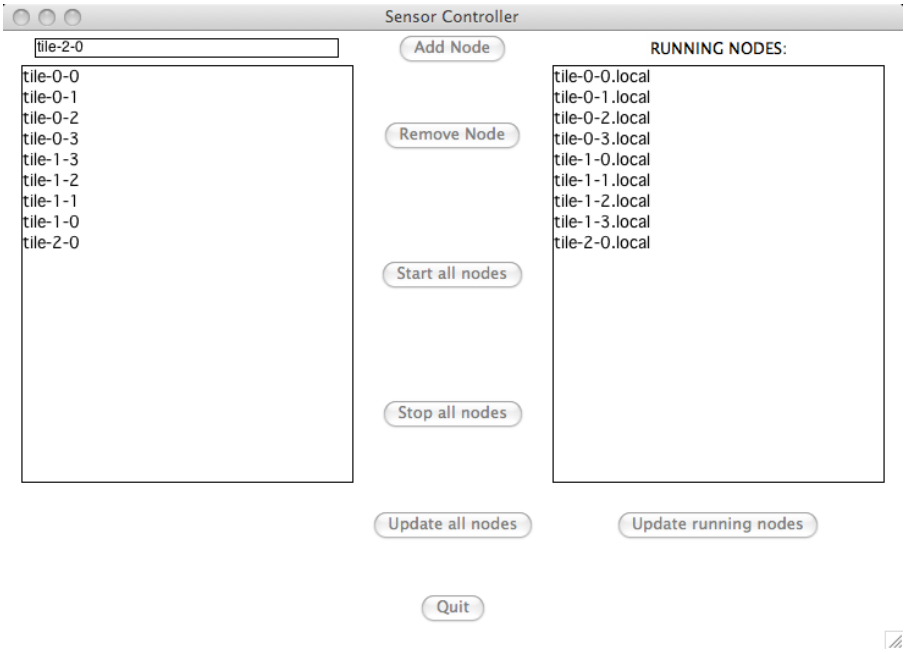

Figure 27: Sensor controller, Tk version

This controller has a limited number of responsibilities :

- Start sensor nodes
- Stop sensor nodes
- Update software on sensor nodes

To start a sensor node the controller distributes the software to a node in the computer cluster, and starts it as a local daemon. This is done using "scp" and

"ssh" commands. The sensor nodes are started in a demonized way to ensure continuous operation after the "ssh" commands ends.

The controller does not receive a heartbeat from the newly started nodes, instead the controller has to ask the Sensor Lookup for a list of running nodes. As can be seen in Figure 27 the reported node name in the column to the right, and the given name before startup in the column to the left, is not identical. This is because the node is collecting its own name from the cluster node that it is running on and therefore adds the ".local" domain suffix.

The sensor nodes is stopped by requesting a list of running nodes from the Sensor Lookup and issuing a "KILL" command to the HTTP server on each node.

The controller only starts the nodes in the sensor data producing network. The data computing network and sensor controlling network are started manually. Using the software update functionality nodes for these networks may also be started on other nodes.

7.11.2 "External" Software updates

Software updates are initiated externally viewed from the complete sensor network. Our controller is therefore a possible origin for such updates. Distributing a software update is achieved by sending one single file. This file may be a TAR archive containing Python scripts or configuration files. The controller will first request a list of running control nodes from the sensor lookup server, and thereafter send the update to the first sensor control network node on the list. The software update is packaged as a Python dictionary with the following content:

Key	Content
filename	The name of the file to be replaced
data	The content of the file to be replaced
nodelist	A simple list of all nodes to be updated
nodedict	A dictionary over nodes and ports to be updated.

Both the "nodelist" and the "nodedict" elements comes from the Sensor Lookup. The file may be a TAR archive, and if the filename end in ".tgz" the file will be unpacked. The files in the archive may replace existing files or add more files.

When using a "Worm" (see chapter 6.2.8) strategy we can view the described protocols as a mean to propagate in the network. In this implementation we will not use the newly updated code to decide on how the update is to be distributed, but will simply follow the provided list. To implement a better worm-like update strategy, the received software should be stored on disk, the node restarted, and the new software or configuration used to decide the following strategy.

7.11.3 Sensor control node

The sensor control node has basically two tasks in our implementation:

- Keep the sensor network running by monitoring the nodes.

- Distribute software and configuration updates

Monitoring the nodes could have been achieved by having the sensor control node either receiving a heartbeat from the other nodes, or having the sensor control node contacting the other nodes at regular intervals. But since the nodes are already contacting the sensor lookup server at regular intervals, we choose to simply have the sensor control node retrieve the list of active sensors from the sensor lookup service.

The distribution of software and configuration is externally initiated. In our emulation this is done by the sensor control described in chapter 7.11.1. The sensor control node simply creates a list of nodes that will receive the update, and it also generates a complete dictionary as described in the previous section, and sends it to the first node on the list. The update will be forwarded to the other nodes by the nodes themselves in a worm-loke fashion.

Upon reception of this data package the node will write the content of the file to the wanted file, pop its own name from the "nodelist" and "nodedict", and transmit the new package to the next node on the list. After successful file replacement and forwarding of the update, the sensor node will stop with an error code indicating that the Process Monitor should restart the node.

7.12 Startup sequence

In our emulated system, elements have to be started up in a sequence. This is simply to assure that all necessary services are running prior to its use by other parts.

1. Sensor Lookup server
2. Sensor data storage nodes.
3. Sensor control nodes
4. Data producing sensor nodes

The lookup server is used by all types of nodes and has to be running prior to anything else. This services is therefore best regarded as part of the general network infrastructure. The sensor data storage nodes has to be started before the data producing sensor nodes, as the producing nodes will start the delivery of data immediately.

7.13 Data Viewer

A simple application for looking at the sensor data was implemented. Since the emulated sensors delivers images, a simple image viewer is programmed using Python and TK to ensure portability and the ability to run on all available nodes. The viewer will retrieve the latest images from all the available sensors from the sensor data storage network and display these in one large window. If the window is resized and click on, all images will be resized. Whenever a click is registered in the window, the program will reload the images.

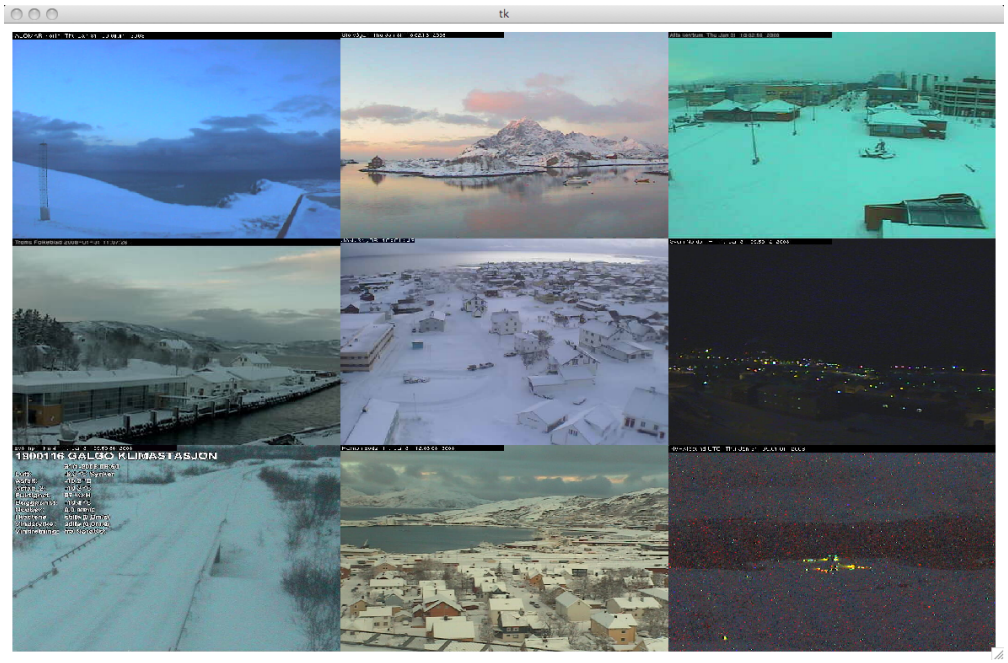

Figure 28: Simple data viewer for image sensor data

The data viewer will also store the last composite image to a file for other use.

The data viewer collects all available keys from a sensor data storage node using the pure "textkeys" method, and sorts this after node source. The last available image from each node is retrieved from the data storage, and a large composite image is created.

In Figure 28 an example is shown. Images from 9 different sensor data producing nodes have been collected for the sensor data storage node and are put together. The actual programming needed for doing this is very simple, and can be illustrated with some parts from the python program. Some actions and contents have been removed and are explained in the comments.

```
    # URL contains  address of sensor data storage node
    #
    url = URL + "textkeys"
    f = urllib.urlopen( url )
    lines = f.read()
    f.close()

    allkeys = lines.split("\n")

   # Some selection of keys from the "allkeys" tuple is performed and the remaining
   # keys is put into a "nodes" dictionary. The last key for each node may be selected.

    url = URL + "DATA/"
```

```
for k in sorted( nodes.keys() ):
    kk = nodes[k]
    try:
        f = urllib.urlopen( url + kk )
        i = f.read()
        f.close
        res.append( i )
    except:
        pass

 # "res" contains all images from the selected list.
```

Other languages have similar libraries to retrieve data over the Internet from URLs and this can also easily be done using web 2.0 type applications. This demonstrates that using HTTP servers for the data storage creates easy access in terms of programming requirements.

7.14 Items previously described but not fully implemented

The following items are described previously in this dissertation but is only partially implemented:

- Redundancy in the sensor data computing network or in the sensor controlling network. Both networks may have several nodes that registers with the sensor lookup server, but no detection of failure is done.
- Data forwarding between nodes in the sensor data computing network. All data is processed locally.

8 Experiments

The whole system was set up and run on the Display Wall lab at the Institute of Computer Science, with the different tasks distributed as shown in Table 3.

Table 3: Experiment setup

Where running	**Task**	**Instances**
On the Computer cluster	Sensor data producing nodes	up to 28
	Sensor Computing nodes (data storage)	1
	Sensor Controller nodes	1
On a Macbook laptop	Controller for the emulation	1
	Sensor Lookup server	1
On a Linux workstation	Data Viewer	1

Both the Sensor data computing nodes and the Sensor controller nodes may be started on more nodes, up to 28 on the cluster and also one each on the Macbook laptop. The limit is due to the use of TCP/IP ports on each computer. Only one process can use a specific port at a time.

The experiment was set up so that the nodes on the cluster delivered one observation item, typically an image, every 60 seconds. This gives a volume of observations in the order of 50 - 100 Kb/s coming to the data storage nodes so this represents a low volume network.

To test the experimental setup a simple stress-test of the Sensor computing node was performed. Two computers in the cluster has 20 threads each doing 1000 iterations of sending observations. The data provided was an image with a total size to be transmitted of 86 kB. Some results from the stress run on one of the computers follows in table 4. Several threads could not connect to the Sensor computing node in at least one iteration.

We can see that the throughput from one thread is in the order of 580 - 720 kB/s. In total with 20 threads on each of two computers, gives a throughput in the order of 20 MB/s. It must be noted that the Sensor Computing node has to store the observation in the local files system. This filesystem is a NFS share between all computers in the cluster. The network load is therefore significant from this part of the test.

Table 4: Througput in stress-test of Sensor Computing node

Speed in B/s
718088
714152
690691
.... (removed some rows) ..
584814
583821
579298

8.1 Super Sensor Network experiment

The experiment consisted of the following phases :

- Deploy the software to all nodes and start all processes
- Collect data from all sensor data producing nodes
- Issue a software update from the controller with one single-file update
- Issue a software update that adds a new process and updates the configuration. Multi-file update.
- Read data from the storage nodes and from the individual data producing nodes
- Accessing the data from a Sensor computing node from as program on an computer outside the cluster but on the same subnet.
- Stop all processes in the emulation.

All phases were successfully completed. The first software update consisted of a small change in the handling of images at the data producing nodes, where the newer version simply annotated the image with some informations. This is demonstrated in Figure 29 (before) and Figure 30 (after). The small difference is the small text at the bottom left corner. This update also added the functionality of each node to display the current image on the nodes own tile on the Display wall. The software update transmitted contains the updated file and some additional items and the size was 7103 B.

Updating all nodes in a 28-node emulation takes close to 8 seconds, where most of this time used for stopping and starting of processes. The local processing on the Sensor Data Producing nodes when receiving a software update is minimal before the node forward the update to the next node in the queue. This makes the distribution of the software between nodes efficient, even if it is done sequentially from node to node.

Using the data viewer, the update could be seen rapidly spreading in the data producing networks.

The second software update added a sensor control node to each sensor data producing node. This demonstrated adding functionality to any node. The software update consisted of two files packed in one packed TAR archive. This was also successfully done. Updating all nodes to add this new functionality takes the same time as a regular software update. This is as expected as most of the time for an update is in stopping and restarting the processes on the node. Adding one new process does not significantly increase the load time. This is most likely a function of the underlying operating system and file-system.

Figure 29: The original image from the sensor node

Figure 30: The updated image from the sensor node

9 Related work

9.1 Operating systems

A popular operating system used in current research is TinyOS [20]. This OS focuses on providing the platform for three goals in developing sensor networks and sensor nodes.

- Provide an extensible and developing platform.
- Allow diverse implementations in varying mixes of hardware and software.
- Be suitable for limited resources, concurrency intensive operations and robust application-specific development.

TinyOS provides a framework for defining different services and does not define a particular system/user boundary. TinyOS has been implemented on various hardware platforms on various sensor node systems. TinyOS provides also a complete over-the-air reprogramming mechanism using Deluge [31]. Deluge is a reliable data dissemination protocol for wireless sensor networks. TinyOS provides different tools for single-hop and multi-hop communications using active messages [32]. Active Messages are an event centric communication API. TinyOS provides also several network services like multi-hop broadcasting, routing, power management and time synchronization.

TinyOS has not traditionally used IP for networking. This means that only the edge of the sensor network was externally accessible and could use the large base of technology IP provides. This is changing and some implementations exist today. TinyOS is also optimized for small nodes. In this dissertation the focus was on large nodes and IP capable networking. But with newer implementation TinyOS could also be a viable platform for the architecture proposed here.

9.2 Virtual machines

Maté [8] is a small virtual machine designed for sensor networks. Maté has a very high level interface and is a byte-code interpreter that runs on TinyOS. Programs can therefore bee very small, typically less than 100 bytes. Maté has been extended to a framework for building application-specific virtual machines. When updating a sensor node a complete TinyOS image with a modified virtual machine is transmitted.

Using virtual machines to run multiple applications in parallel and with high degree of isolation from each other is very desirable. Maté could also have provided a basis for the architecture proposed here, and would have been a good candidate for running processes belonging to separate functional networks on the same node. Maté is based on TinyOS and the initial lack of IP capability did make it unsuited for this dissertation.

9.3 Security

For many sensor networks security is a major problem. One example can be networks used for surveillance and monitoring purposes. Unauthorized access to

the observations or the possibility of corrupt or incorrect data due to intruders is a serous problem.

In the survey by Walters et. al. [33], the challenges is divided into these aspects :

- The obstacles to sensor network security
- The requirements of a secure wireless sensor network
- Attacks
- Defensive measures.

The survey also contains a walk-through of the different problems and available solutions. They expect that as wireless sensor networks become more common, the use of public-key cryptography will meet the expectation of strong security.

One example of a communication systems that implements strong security in wireless sensor networks is the Dust Networks SmartMesh-XT [21]. This system implements 128-bit symmetric key encryption for end-to-end confidentiality in a totally meshed network [10]. The motes are interfaced with other types of equipment and may acts like NICs.

In this dissertation we do not study security concerns and assumes that some of the technologies referred above is present.

9.4 Middleware

Cascades [34] is a complete middleware for use in sensor networks with emphasis on video sources. It is developed based on the experiences with Panoptes [35] where low powers video sensors were used in a testbed for further research. One of the main features was the possibility of intermittent network disconnects where the node had to store data for later retrieval, and with memory constraints, had to do thinning on the video stream to allow new data.

Cascades uses Python scripts to provide high level access to powerful functions leading to small but efficient routines. The system composes a data flow using cascading "filters" where the individual filter are highly optimized for its task. Filters are connected using python interconnects that are both means of exchanging data on a computer, but also between computers on a network. Different types of filters handles different types of data, both traditionally on-parameter sensors and video streams are used. A separate filter is used for metadata and control data handling, so that missing video data can be detected. Filters can be updated during operations and will be reloaded when the update is detected. The means of updating is not described.

Cascades has a fixed structure of control and relies to some extent on a centralized model. This is different from what this dissertation have assumed, and Cascades was therefore not a good starting point for the architecture presented here. Cascades has a very efficient programming style and very high level abstractions are available for use. Also the possibility to dynamically update an

active application is impressive. In this dissertation the focus was on meteorological observations and not video. Cascades is still a good candidate for some types of meteorological networks, like monitoring for the occurrence of fog or other visibility-reducing phenomena at airports.

DAVIM [36] is a new service middleware for sensor networks implemented as a dynamic management of services on virtual machines. DAVIM uses a combination of native code and virtual-machine byte-code for applications and argue that this has been shown to be energy efficient. DAVIM is designed to meet some requirements.

- Services should be separate from the application using them.
- Managing available services should be dynamic and easy.
- Multiple applications running on the same sensor network should be isolated.
- The complexity of implementing recurring functionality should be lowered by the middleware.

The services and applications are separated at runtime as the applications are a byte-code script that runs in a virtual machine of their own, on top of the DAVIM core. This removes the static linking between VM instructions and operational libraries.

DAVIM was not available at the start of this dissertation, and the first publication was noticed around January 2008. DAVIM implements some of the same architecture features studied in this dissertation. Both control and computing functional networks may be implemented in DAVIM and run on the same node as the sensor data producing code. Using virtual machines these would have been separated from each other but at the same time monitored by the underlying VM. DAVIM would be a very interesting technology to built the studied architecture on.

9.5 REST

The REST [18] [19] style of organizing HTTP access and how to build up URLs, provides architectural constraints that may ease implementation of communication in heterogenous systems. The use of REST does not imply a need for any specific language or system used in the implementation.

One of the important aspects of REST is that it is stateless. This means that the participating entities does not have to keep state between sessions. This is helpful for smaller implementations. This also means that most results are cacheable. See also Appendix C for an example of use of REST.

The use of HTTP will not place restrictions on the type of data exchanged but will induce an overhead in the communication. HTTP is most often implemented on top of TCP/IP, but can be implemented on top of any protocol delivering reliable transport. HTTP can be used in a compressed mode that is supported by most clients if the size of the data stream is regarded as a problem.

Also simple HTTP servers are implemented in a number of languages with a very small memory and processor footprint.

REST is therefore a good mach for the systems used in this dissertation and was easily implemented.

9.6 Software update

Several good algorithms for distributing software updates in wireless sensor networks exist. A few examples are Deluge [31], MNP [37] and Freshet[38]. In this dissertation a highly efficient algorithms is assumed to exist, and as an example some parts of Freshet algorithm is described here.

The Freshet algorithm is designed to provide an energy efficient on-demand distribution on new code in large scale sensor networks. In Freshet some nodes act as originators of new code. These are typically gateways to the network. These originators will first start a "blitzkrieg" phase where a "warning" message with information on the availability of new code is pushed to all nodes in the network. As these "warning" messages traverses the network each node will rebroadcast it to the adjacent nodes after having incremented a hop-counter. For all nodes with a hop counter of less than 3 will start to listen after broadcast of the availability of new code. A node which hears an availability broadcast will request the code from the source, which will send the code to the requesting node. This availablity-request-code phase is called a three-way handshake. Any nodes also hearing the transmissions will receive the new code. Nodes with a hop count of 3 or larger will enter a waiting state with their radio receivers turned off to preserve energy. How long this waiting is, is dependent on how long distance from the originating node the node is. A node will act as a source of the new code as soon as the first parts is received. Several optimizations for receiving larger parts without the three-way handshake are also included. Also optimizations for new nodes needing large updates are part extensions of the basic algorithm.

The Freshet algorithm would be very interesting to implement in the software update part of the architecture studied in this dissertation. It is a minor problem that the state of a node is dependent of wether a software update has been announced or received.

9.7 Existing service discovery functionality in regular networks

Frank, Handziski and Karl [39] have a thorough review of different forms of service discovery in use today in wireless sensor networks. A simple list may include :

- Jini, Java's protocol-independent framework for heterogeneous networks and software components.
- Bonjour/Zeroconf/Avahi, User by Apple and in Linux systems for network configuration and service discovery.
- UPnP with SSDP. Used by Microsoft for communication between devices on a network.

- SLP, Service Location protocol has reached IETF Proposed Standard level. SLP is described in RFC 2608 and RFC 3224
- JXTA, A general purpose communication system that also includes peer discovery.

JINI and Bonjour technologies are the most relevant in this dissertation. In this dissertation the choice to use Bonjour was mostly based on the selection of programming language and os and hardware platform. Bonjour provides also a programming language independence.

9.7.1 JINI

JINI presents a comprehensive systems with integrated service discovery and code transparency and remote execution. The JINI systems is not protocol dependent as the systems allows for interchanging modules loaded on the fly using any wanted communication protocol. JINI supports service discovery via brokers available on the network.

To access a given JINI service the program locates the object that supports the wanted service. The object is used to download the code needed to access the service. This is done using a standard interface in Java. To facilitate this several services must exist on the local network.

- RMI Registry
- A JINI Lookup Service

The JINI services uses the Java security functions to access and control access to remote and local code. A simple use of JINI can be illustrated with the code in Appendix Appendix A

9.7.2 JXTA

JXTA [40] is a set of protocols for general-purpose computer-to-computer communication. JXTA is an open-source, community-based project originally initiated by Bill Joy and Mike Clary at Sun Microsystems. JXTA has defined a set of XML-based protocol and a Java reference API to provide a generic framework for P2P.

The JXTA Protocols Specification describes several protocols and functions (not exhaustive list):

- Discovery of the services peer and peer groups provide to the network
- Services including location, specifications and executable code for the service.
- Data transfer. JXTA provides the concept of a "pipe" for transferring data in an abstract fashion. Including one-to-one, one-to-may regardless of protocols.

- Message routing including rendezvous peers that may cache information.

JXTA is at present implemented in Java SE, Java ME and C/C++/C#. There is some work on Ruby and Python bindings but these are not complete at the time of writing.

9.7.3 Bonjour

Bonjour [23] is Apple's implementation of zero-configuration [24] over IP networks. Bonjour solves three problems :

- Addressing and assigning of addresses.
- Naming.
- Service discovery.

Bonjour uses a multicast DNS [41] [42] (mDNS) in the local IP network. Since all devices participating on the local network runs either a daemon mDNS responder for its services or a local mDNS responder on each computer, no common DNS server is needed. Also this implies that no DNS server has to know all available services.

The local application has only to register with the local mDNS responder for other applications to be able to locate the services provided. This relieves the application from handling all types of mDNS messages.

Registering with the local mDNS responder can be illustrated with the test code in Appendix Appendix B. This code registers a service at a given port on the local computer. More specific it registers a HTTP server to be visible as a service in a Bonjour capable browser.

The browsing is done in an asynchronous way. First a request is issued on the local network and then each reply is handled by the call-back routine. For each reply a simple DNS service resolve is done to retrieve the full host name of the service provider. The ".local" is used to indicate a name that should be looked up using a IP multicast query on the local IP network. The test code will locate any HTTP servers that are bonjour capable and have a bonjour responder daemon on the local network. In this example the programs looks for the service type (or application protocol) of "_http._tcp", that is HTTP over TCP. The responder can be either a thread in the local application or the mDNS responder on the computer. All responses from these responders look like standard DNS responses.

Bonjour uses an extension to the standard DNS system with the local multicast. This reuse of existing standards makes implementing such services simpler. Bonjour uses ".local" as an indicator for computers on the local network and will use a broadcast to find the wanted machine. This implies that Bonjour has a special semantic for the ".local" domain that departs from the DNS standard. The use of the ".local" domain is not routable over network partitions but are sufficient for accessing nodes within the same segment.

9.7.4 Avahi

Due to licensing concerns Bonjour was not originally implemented on Linux systems and a version that could be released under the LGPL license was created. This is named "Avahi". This is an implementation of the Bonjour protocols (mDNS) and also of the DNS-SD standard. It is commonly available on all newer linux distributions and is heavily used in the Gnome and KDE desktops.

As Bonjour, Avahi is also an implementation of Zeroconf. Zeroconf is the need for simple configuration of address, name and service discovery. This is in some respects an IPv4 problem, as IPv6 has local-link addressing. Service discovery is done using the standard DNS SRV records. This gives the user both the IP-address, the host name and also the port where the service is running. Additional information may also be available.

Avahi is available as a complete and comprehensive implementation.

9.8 Example of birds nests monitoring application

In cyclops [43] a complete system for deploying image sensors in a biological study have been developed. The small sensor nodes has a relatively low resolution (320x280) image sensor and has the Mico2 mote for radio communications. Images are collected each 15 minutes and are sent to a local data server that runs Linux and is connected to the Internet. The classification of the images may be done in the node and only a classification value sent. The study has also examined the balance between using powers at the node for image compression compared to the gains in energy consumption when transmitting the images over radio.

The nodes has limited powers supply (running around 15 days on batteries) and the software in the nodes are not on-site update-able. One of the interesting aspects is the detection of eggs in birds nests. This is done using a highly optimized algorithm on the node. One could speculate that the possibility for updating this algorithm without physical contact with the node may be of interest.

The example shows that location local computation on sensor nodes is in some cases highly desirable, but also demonstrates that even for small nodes, on-site wireless software updates may help. In this example the sensor nodes is fairly accessible, but physical presence may cause disturbances in the observed data.

9.9 Video surveillance

Palau et. al [44] has implemented an urban traffic monitoring system in use in Valencia, Spain. The system uses a high number of video cameras (more than 600) where MPEG4 video is streamed using MPEG-TS [45] over IP. All nodes are connected via high speed fiber optic network. To isolate different parts of the network different virtual networks, VLAN (IEEE 802.1Q) are defined and the video streams uses one VLAN while information from other sensors are sent through a separate VLAN.

Video streaming to end applications are handled by video servers that are deployed on selected points in the city. The role of the video servers is to capture the raw video signal, compress it using MGPEG4 and transmit the video stream using MPEG-TS. The whole of the system is designed and implemented in the COTS (Commercial off-the-shelf) philosophy. The playback system handles multiple simultaneous clients using IP multicast as this is available on the L 3 switches in the TMC (Traffic management centre) part of the core network.

One of the lessons learned was that possibility to easily change the operating software on the nodes in the network was a great benefit. Video encoding could be changed without physically handling devices and new and improved codecs could be installed. This was also one of the reasons for studying software updates in this dissertation.

9.10 Tapestry

Tapestry [46] is an overlay infrastructure designed for scalable and fault-tolerant applications in a wide-area IP network. Tapestry studies the access to objects that may exist in several copies on different locations in the network. Tapestry provides integrated location and routing using a strong focus on the locality of communications, where routing is decided on local available information only and not on a total network overview. This also provides fault tolerant routing and location. Tapestry uses both the soft-state, announce/listen approach that relies on TCP timeouts and periodic UDP heartbeats, and also explicit republishing of location information. The priority between these solution is based on the substantial network overhead using soft-state, so that explicit republishing is the preferred method and soft-state as the fall-back method.

Relative to this dissertation Tapestry provides an example that the underlying routing and distribution of information and services in a IP network can be handled as overlay networks with a small overhead.

Also Pottie and Kaiser [47] show that computation close to the source of the raw data can drastically reduce the communication and total computation cost for a complete system. This may also lead to better scaling networks.

10 Discussion

10.1 Failure Models

The most difficult failure model is when the device or node delivers wrong data. Due to the large natural variations in the meteorological conditions, these kinds of failures may be very hard to notice. The most common way to handle these types of failure is to black-list the node which removes any value of the node. Another solution is to try to correct the incoming data, if some common or systematic error can be detected. In this case some value is retained although the node still has errors. In this dissertation the nodes are assumed to deliver correct data at all times.

10.2 Communication

The use of IP in a sensor network gives access to a very large base of technology and methods. Very good and proven technology exist for a number of issues, like routing, security, applications support and service discovery. Many of these issues has to be revisited to handle constraints in a sensor network, but the basic framework exist. For further details see `http://en.wikipedia.org/wiki/6loWPAN`

Using HTTP based protocols was relatively simple to implement as there exist a large base of modules for HTTP communication. This also makes the small servers at the node and at the data storage program, simple and extendable. The data producing nodes and the data storage nodes can also be accessed from a standard web browser, which makes debugging much simpler.

The use of HTTP for communication and device setup has become an industry standard. All different kinds of devices have now a small and effecive HTTP server built in.

10.3 Routing

In this emulation the nodes must ask a known entity about other nodes. This represents a possible single point of failure and a possible congestion point. If no contact with the known controller is established, no contact with neighboring nodes is possible. In a environment with complete support for Bonjour/Zeroconf the structure is such that any nodes may listen in on the service discovery traffic and will cache names and services for its own use if it sees something new in the traffic.

10.4 Communication overhead

A single observation is delivered as a single POST to a HTTP server. The posted data is an pickled Python dictionary. The total network overhead can be divided into three parts. An TCP/IP part, an HTTP part, and a data packaging part.

To estimate the data packaging part we can compare one actual JPEG image (from figure 29) with the stored Python dictionary containing the same image in addition to the other small elements (see chapter 7.10) :

Mode	Content	Size in bytes
Uncompressed	Original image size	67093
	Pickled Python dictionary	193778
Compressed	gzip'ed original image	66752
	gzip'ed Pickled Python dictionary	89718

If the web server supports HTTP Compression (see RFC2616) some of the pickle expansion would be mitigated. One of the possible compression algorithms is gzip. As shown above the dictionary is closer to the original size even though it contains additional information compared to the original JPEG image. Still the method used in this dissertation generate an overhead of around 50 % compared to a gzip'ed version.

It should be noted that Pythons pickle.dumps function with the default protocol was used. This results in an ASCII string that is human readable. Using a higher protocol would result in smaller sizes with very little overheads. The choice of using the default protocol in stead of a more compact higher protocol was mostly to keep the network traffic human readable and network bandwidth was not a limiting factor in the emulation.

TCP/IP overhead and HTTP protocols has been studied and Frystyk Nielsen et. al. at W3C [48] has demonstrated that the TCP/IP overhead is between 5 -10 % and using HTTP compression the HTTP protocol does not increase the overhead much. This is mostly due to the compression algorithms. This overhead has to be weighted against power requirements by the network requirement to implement HTTP compression.

10.5 Coordination of control

On possible solution to the problem of having many sensor control nodes trying to update nodes with different generations of software (see Figure 13)is to have every item carry a "version" number. This version number has to be monotonously renumbered so that every new update has a higher number than any previous versions. If any control node sees a data producing node or data computing node with any item with a newer version, it will simply wait for the upgrade to diffuse to it. In the case of a mistaken update or an update with faults, "downgrading" to the older version is done by issuing a new update with a higher version number but containing the older software.

It is also possible to divide the data producing and data computing nodes into groups, and have each group controlled by one sensor control node. This simplifies the control issues, but has no redundancy in case of communication or other failures.

Also algorithms like Freshet (see chapter 9.6) may remove the described problems.

10.6 Service discovery

Using UDP for the initial broadcast to find the Sensor Lookup server had several problems connected to the possible loss of data packets in the network. Therefore a mechanism for retransmissions had to be implemented. In this dissertation this is done only on the data producing nodes as the Sensor Lookup Server does not demand an response on its reply. The data producing node will retry until it has a valid response.

In practice this may not be a large problems as at least some of the available commercial solutions delivers a network with a optimized and guaranteed delivery of network packages. One example is Dust Networks SmartMesh-XT [21].

Using Bonjour in this emulation was difficult because of the missing support in the computing cluster. This problem was solved using a combination of UDP broadcasts and HTTP. It is therefore satisfying to find other solutions using the same basic idea with UDP broadcasts and HTTP for detailed communications. In mRDP [49] some of the same ideas are used for a lightweight semantic service discovery protocol.

Bonjour has nevertheless a very attractive feature with the lack of centralized servers and local control. It also makes it simple to both browse and announce services.

10.7 The worm update

Using a worm-like update does a sequential update of all nodes and could be replaced with a more efficient algorithm. In the implementation in this dissertation any node receiving an update, simply forwards this according to the given list on nodes. In reality this could have been replaced with a process where the node is first updated and then does some processing to decide how, what and who to update. This creates the opportunity for selective updates based on information available on the local node.

A worm would logically be part of the Sensor Controlling functional network and this network can, in principle, be implemented as a worm with extended functionality for controlling other parts of the Super Sensor network.

10.8 Node state

In the implementation in this dissertation only a small part of node state information stored on the Sensor Lookup server. Only token information on operating system from each node is provided. Additional information should be easy to implement and would make it possible for the sensor controlling network to check that software updates are implemented and that any node is functioning correctly. This would also make it easy to take simple decisions on where to locate processing, data storage or other functionality.

Using Bonjour, node state and other information can be stored in the DNS records for each node. This mechanism is as an example used for storing information on printing capabilities for Bonjour capable printers.

Table 5: Estimated load in kB/s

nodes \ Frequency	60	30	20	10	5	1
20	4	8	13	25	50	250
40	8	17	25	50	100	500
80	17	33	50	100	200	1000
160	33	67	100	200	400	2000
320	67	133	200	400	800	4000
640	133	267	400	800	1600	8000
1280	267	533	800	1600	3200	16000
2560	533	1067	1600	3200	6400	32000

10.9 Bandwidth requirements vs. The number of Sensor Computation nodes

In table 5 we give a rough estimate of data traffic generated by the emulated nodes.

From Dust networks [10] we have the following calculations of available bandwidth :

```
.....  a TSMP implementation on 802.15.4 radios with 60
timeslots per second provides:

 16 channels x 60 slots/second = 960 transmissions/second

 Assuming an 80 B payload the effective total bandwidth is:

 960 transmissions/second x 80 B/transmission = 76.8 KB/second
```

Using this effective bandwidth in a wireless environment we can se from the table that with an image size around 100kB, we have to reduce the number of nodes or observing frequency or introduce a number of Sensor computation nodes for intermediate storage.

10.10 Conclusion

This dissertation has studied composing a super sensor network from the combination of three functional sensor networks. The communication was based on an IP network using HTTP for the main parts. The intention was to use Bonjour for service discovery, but a minor adjustment were needed for technical reasons.

This dissertation have demonstrated that the selected technology and architecture may handle some of the demands in a sensor network, and that the architecture gives new opportunities on how to handle updates and control.

The implemented system also demonstrates that using standard Internet protocols can make access to services in the sensor network easy. A web browser may become the preferred user interface for controlling and accessing all parts of the sensor network, as it has for controlling printers and simple network devices.

11 Future work

The proposed architecture is not fully studied or tested in this dissertation. Both control and processing have large untested parts and issues to be further studied. Also the concept of using a worm for software updates. A new approach could also be to use a worm for data processing, doing queries.

Further testing could be done using small portable computers. One possible setup where many different sides could be tested is suggested below :

11.1 Air quality and road monitoring

The aim of the sensor network is to monitor the air pollution and road status in the Tromsø area.

Two types of sensor nodes :

- Stationary, at strategic geographical positions
- Mobile. On the local Busses in Tromsø.

The stationary nodes monitors local conditions and some are connected to the Internet. The mobile nodes monitors the conditions at regular intervals as the busses moves around on the roads in Tromsø. Both kinds of nodes would be large nodes with significant computing and power capabilities available.

Parameters to be monitored :

- Air temperature
- Wind. On busses this is almost impossible.
- Air pollution by measuring PM10, PM2.5 and NO2 (very small particles and nitrogen for diesel engines)
- Road status by taking an image
- Position and speed

Todays situation is best described on `http://www.luftkvalitet.info/`. Tromsø have two measuring stations that takes point measurements with high frequency and good quality.

The described setup would have many interesting sides both for local air quality monitoring, local meteorology and computer science. Some of the computer science questions that could be answered may be:

- Is this dissertations architecture viable in networks with highly changing topography?
- Do IP work in a sensor network?
- What network technology is best to use for exchanging information with mobile nodes?

- Is P2P a viable technology for sensor networks ?
- How to keep a running query on the nodes to support alarm services?
- Can road status be inferred from images?

References

[1] P. Marrón and D. Minder, "Embedded wisents research roadmap," *informatik.uni-stuttgart.de*, 2006.

[2] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, "... : a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance," *Proceedings of the 6th international conference on ...*, Jan 2007.

[3] A. commitee on Computers and P. Policy, "The risks digest," *The risks digest*, vol. 19, p. 13, Feb 1998.

[4] W. World Meteorological Organization, "Global observing system, world weather watch." http://www.wmo.int/pages/prog/www/OSY/GOS.html.

[5] ECMWF, "The 4dvar analysis procedure." http://www.ecmwf.int/products/forecasts/guide/The_ 4DVAR_ analysis_ procedure.html.

[6] R. Brækkan, "Polar avs aa96," tech. rep., Norwegian Meteorological Institute, Feb 2008.

[7] Wikipedia, "Wikipedia, device bandwidth." http://en.wikipedia.org/wiki/List_ of_ device_ bandwidthsWireless_ networking.

[8] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *ACM SIGOPS Operating Systems Review*, Jan 2002.

[9] met.no, "Stasjonskart2007." http://met.no/Meteorologi/A_ male_ varet/Observasjoner_ fra_ land/filestore/stasjonskart2007.pdf.

[10] D. N. inc, "Technical overview of time synchronized mesh protocol," *White Paper*, 2006.

[11] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, Ltd, 2005.

[12] NOAA, "The argos system." http://noaasis.noaa.gov/ARGOS/.

[13] Wikipedia, "Intelligent vehicular ad-hoc network." http://en.wikipedia.org/wiki/Intelligent_ Vehicular_ AdHoc_ Network.

[14] A. D. Instruments, "Autiomatic weather station aws," *Product Description*, 2006.

[15] A. Instruments, "Recording current meter," *White Paper*, 2000.

[16] ScanMatic, "Scanmatic transport technology." http://www.scanmatic.no/Default.aspx?page=products13.

[17] QNX, "Qnx real time operating system." http://www.qnx.com/.

[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," *DISSERTATION*, 2000.

[19] R. Fielding and R. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, Jan 2002.

[20] P. Levis, S. Madden, D. Gay, J. Polastre, and R. Szewczyk, "The emergence of networking abstractions and techniques in tinyos," *usenix.org*, 2004.

[21] DUST, "Dust networks smartmesh-xt." http://www.dustnetworks.com/products/smartmesh-xt.shtml.

[22] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, Jan 2005.

[23] Apple, "Bonjour overview," *Apple technical documents*, May 2006.

[24] Wikipedia. http://en.wikipedia.org/wiki/Zeroconf.

[25] Avahi.org. http://avahi.org/.

[26] SitePlayer, "Siteplayer serial to ethernet." http://www.siteplayer.com/telnet/index.html.

[27] S. Cheshire, "Bonjour techtalks at google." http://video.google.com/videoplay?docid=-7398680103951126462q=Google+techtalks.

[28] M. Kuorilehto, M. Hännilkäinen, and T. D. Hämäläinen, "A survey of application distribution in wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, Jan 2005.

[29] J. F. Shoch and J. A. Hupp, "The worm programs, early experience with a distributed computation," *Commun. ACM*, vol. 25, no. 3, pp. 172–180, 1982.

[30] O. J. Anshus, J. M. Bjørndalen, and T. Larsen, "A scalable display wall using commodity components," *Presentation at the University of Tromsø*, 2004.

[31] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," *Proceedings of the 2nd international conference on Embedded . . .*, Jan 2004.

[32] P. Buonadonna, J. Hill, and D. Culler, "Active message communication for tiny networked sensors," *. . . the IEEE Computer and Communications Societies (INFOCOM'01 . . .*, Jan 2001.

[33] J. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security: A survey," *Security in distributed*, Jan 2006.

[34] J. Huang, W. Feng, N. Bulusu, and W. Feng, "Cascades: scalable, flexible and composable middleware for multi-modal sensor networking ...," *Proceedings of SPIE*, Jan 2006.

[35] W. Feng, B. Code, E. Kaiser, M. Shea, W. Feng, and L. Bavoil, "Panoptes: A scalable architecture for video sensor networking applications," *Proc. of ACM Multimedia*, Jan 2003.

[36] W. J. Wouter Horré, Sam Michiels and P. Verbaeten, "Davim: Adaptable middleware for sensor networks," *IEEE Distributed Systems Online*, vol. 9, no. 1, 2008.

[37] S. Kulkarni and L. Wang, "Mnp: Multihop network reprogramming service for sensor networks," *at the 25th IEEE International Conference on Distributed ...*, Jan 2005.

[38] M. D. KRASNIEWSKI, R. K. PANTA, S. BAGCHI, C.-L. YANG, and W. J. CHAPPELL, "Energy-efficient on-demand reprogramming of large-scale sensor networks," *ACM Transactions on Sensor Networks,*, vol. 4, p. 38, Mar 2008.

[39] C. Frank, V. Handziski, and H. Karl, "Service discovery in wireless sensor networks," Tech. Rep. TKN Technical Report TKN-04-006, Technical University Berlin, 2004.

[40] S. L. Daniel Brookshier and B. Wilson, "Jxta: P2p grows up." http://java.sun.com/developer/technicalArticles/networking/jxta/.

[41] S. Cheshire and M. Krochmal, "Dns-based service discovery," Mar 2006.

[42] K. Sekar, S. Cheshire, and M. Krochmal, "Dns long-lived queries," Mar 2006.

[43] S. Ahmadian, T. Ko, S. Coe, M. Hamilton, and M. Rahimi, "Heartbeat of a nest: Using imagers as biological sensors," *Center for Embedded Network Sensing*, Jan 2007.

[44] C. Palau, M. Esteve, J. Martinez-Nohales, and B. Molina, "Controlling urban traffic with flexible video streaming," *Potentials*, Jan 2006.

[45] ISO and I. 2000, "Information technology — generic coding of moving pictures and associated audio information: Systems," *ISO 13818*, 2000.

[46] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Computer*, Jan 2001.

[47] G. Pottie and W. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, Jan 2000.

[48] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prudhommeaux, H. W. Lie, and C. Lilley, "Network performance effects of http/1.1, css1, and png," *W3C note*, 1997.

[49] J. Vazquez and D. L. de Ipiña, "mrdp: An http-based lightweight semantic discovery protocol," *Computer Networks: The International Journal of Computer and ...*, Jan 2007.

Appendix A

```
import net.jini.core.entry.Entry;
import net.jini.core.discovery.LookupLocator;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.lookup.entry.Name;

import java.rmi.RMISecurityManager;

public class MyClient {
    public static void main (String[] args) {
        Entry[] aeAttributes;
        LookupLocator lookup;
        ServiceRegistrar registrar;
        ServiceTemplate template;
        MyServerInterface myServerInterface;

        try {
            System.setSecurityManager (new RMISecurityManager ());
            lookup = new LookupLocator ("jini://localhost");
            registrar  = lookup.getRegistrar();
            aeAttributes = new Entry[1];
            aeAttributes[0] = new Name ("HelloWorldServer");
            template = new ServiceTemplate (null, null, aeAttributes);
            myServerInterface = (MyServerInterface) registrar.lookup (template);
            System.out.println ("Calling sayHello()->" + myServerInterface.sayHello () + "<-");
        } catch (Exception e) {
            System.out.println ("client: MyClient.main() exception: " + e);
        }
    }
}
```

This program basically does:

- Set up security manager to access RMI/remote code.
- Find the JINI locator.
- Find the local registrar using the JINI locator
- Find the remote interface using the local registrar
- Call the remote function using local downloaded interface

Appendix B

```
import threading
import pybonjour

class RegisterBonjour(threading.Thread):
    """ Thread for registering and servicing bonjour/zeroconf requests."""
    def __init__(self, name, regtype, port ):
        threading.Thread.__init__(self)
        self.name    = name
        self.regtype = regtype
        self.port = port

    def register_callback(self, sdRef, flags, errorCode, name, regtype, domain):
        if errorCode == pybonjour.kDNSServiceErr_NoError:
            print 'Registered service:'
            print '  name    =', name
            print '  regtype =', regtype
            print '  domain  =', domain

    def run(self):
        """Runs the bonjour/zeroconf threads."""
        sdRef = pybonjour.DNSServiceRegister(name = self.name,
                        regtype = self.regtype,
                        port = self.port,
                        callBack = self.register_callback)
        try:
            while True:
                ready = select.select([sdRef], [], [])
                if sdRef in ready[0]:
                    pybonjour.DNSServiceProcessResult(sdRef)
```

```
                print "..... got request"
        finally:
            sdRef.close()

if __name__ == "__main__" :
    rb = RegisterBonjour("TESTREGISTER","_http._tcp",8080)
    rb.start()
```

Please note that this code will run in its own thread but that it has no other contact with its applications. This thread will keep the bonjour register alive as long as the application is running.

To browse for other services this code can be used :

```
import select
import sys
import threading
import pybonjour

timeout  = 5
resolved = []
class BrowseBonjour(threading.Thread):
    """ Thread for watching bonjour/zeroconf requests."""

    def __init__(self, regtype, dsQueue ):
        threading.Thread.__init__(self)

        self.regtype = regtype
        self.dsQueue = dsQueue

    def resolve_callback(self, sdRef, flags, interfaceIndex, errorCode,fullname,
                         hosttarget, port, txtRecord):
        global resolved
        if errorCode == pybonjour.kDNSServiceErr_NoError:
            print 'Resolved service:'
            print '  fullname   =', fullname
            print '  hosttarget =', hosttarget
            print '  port       =', port
            resolved.append(True)

    def browse_callback(self, sdRef, flags, interfaceIndex, errorCode, serviceName,
                        regtype, replyDomain):
        if errorCode != pybonjour.kDNSServiceErr_NoError:
            return

        if not (flags & pybonjour.kDNSServiceFlagsAdd):
            print 'Service removed'
            return

        print 'Service added; resolving'
        resolve_sdRef = pybonjour.DNSServiceResolve(0,
                                                    interfaceIndex,
                                                    serviceName,
                                                    self.regtype,
                                                    replyDomain,
                                                    self.resolve_callback)
        try:
            while not resolved:
                ready = select.select([resolve_sdRef], [], [], timeout)
                if resolve_sdRef not in ready[0]:
                    print 'Resolve timed out'
                    break
                pybonjour.DNSServiceProcessResult(resolve_sdRef)
            else:
                resolved.pop()
        finally:
            resolve_sdRef.close()

    def run(self):
        browse_sdRef = pybonjour.DNSServiceBrowse(regtype = self.regtype,
                                                  callBack = self.browse_callback)
        try:
            while True:
                ready = select.select([browse_sdRef], [], [])
                if browse_sdRef in ready[0]:
                    pybonjour.DNSServiceProcessResult(browse_sdRef)
        finally:
            browse_sdRef.close()

if __name__ == "__main__" :
    rb = BrowseBonjour("_http._tcp")
    rb.start()
```

Appendix C

In `http://www.xfront.com/REST-Web-Services.html` several examples on the REST style is given. Here follows some examples.

Since REST is more an architectural style, no "standard" exist. As an example we can look at another system using this style. The web site "yr.no" operated by the Norwegian Meteorological Institute and the Norwegian Broadcasting Corporation.

Lets se how we can access the weatherforecast for Bossekop area in the city of Alta. What we know :

- Bossekop is a place in the country of Norway.
- Bossekop is a place in the county of Finnmark.
- Bossekop is a place in the commune of Alta

Using this geographical drill-down we can construct several URLs to return different types of forecast. In REST terminology we will get different representational states transfered from the web site.

- First to get a nice graphical timeline with the forecast:
  - `http://www.yr.no/place/Norway/Finnmark/Alta/Bossekop/meteogram.png`
- To get a one-page PDF file with many elements :
  - `http://www.yr.no/place/Norway/Finnmark/Alta/Bossekop/forecast.pdf`
- To get the text forcast in XML format :
  - `http://www.yr.no/place/Norway/Finnmark/Alta/Bossekop/forecast.xml`

As we can se we are getting different representations using only minor differences in the URLs. All URLs are also easily human readable.

The "yr.no" web site has also published an API for retrieving weather related elements using a REST-like style. As an example one can retrieve an image from a weather radar using the following URL:

- http://api.yr.no/weatherapi/radar/1.1/?radarsite=rissa;time=2008-04-12T10:15:00Z;width=460

We can see that we are retrieving data using the `weatherapi` data, from a `radar`, using version `1.1` of the API. The remainder of the URL is not a typical REST style use, but is still fairly readable and easy to use.