Towards a File System Interface for Mobile Resources in Networked Embedded Systems

N.C. Audsley, R. Gao and A. Patil
Department of Computer Science
University of York
York, United Kingdom
neil, rgao, appatil@cs.york.ac.uk

Abstract— **Networks for real-time embedded systems are a key emerging technology for current and future systems. Such networks need to enable reliable communication without requiring significant resources, and provide an easy programming interface. This paper considers a file-system interface across all resources in a networked embedded system, ie. an application can access local, remote and mobile resources using a file interface. The approach is based on Styx [1, 2], part of the network protocol of the Inferno/ Plan 9 OS [1]. The Styx protocol provides file system level abstractions for ease of developing and management at an application layer. To this, we have added limited fault-tolerance and potential mobility for resources. To ensure applicability in a low-resource context, we have defined and implemented a (hardware) Styx IP-core Module[1], removing the need for a CPU and software overhead.**

I. INTRODUCTION

A key emerging issue in embedded systems is the provision of suitable networks that are low resource, and enable applications to be programmed easily. This paper considers a file-system interface to all resources within an embedded network. Such an interface is easy to program, and provides a uniform namespace across all resources, even if they are mobile. An example of such a network is a sensor network, containing a large number of sensor nodes. To design and manage these sensor nodes and associated applications is a multi-dimensional research issue [3], including network topology [4, 5], communication protocol [6, 7], fault tolerance [8], power consumption [9, 10] and cost. In this paper, we extend the Styx protocol [11] to provide file system abstractions across mobile resources; and provide a low-resource hardware implementation.

A file system abstraction, through which the resources on remote sensor nodes are presented as files, is unconventional – usual Operating Systems (OS) do not allow remote control of resources except via local proxy servers. Styx allows such control, reducing the need for complex OS on remote nodes, a distinct advantage for low-resource remote sensors / actuators (eg. as seen in sensor networks, or smart buildings). Applications see only a file system interface. For example, a remote sensor can be represented by two files on the file-like interface: "/sensor_1/control" and "/sensor_1/data". Writing a command to the "/sensor_1/control" file results in appropriate actions, such as turn on/off the sensor. Reading from the "/sensor_1/data" file obtains data from the sensor. Resources can be flexibly managed by "mount" and "unmount" to/from the different nodes (views) in the namespace, according to users application. Hence, the file system abstraction introduces flexibility to application development.

In order to effectively utilise the Styx protocol in embedded networks, two issues need to be addressed. Firstly, Styx needs to be extended to cope with mobility; secondly, a low-resource version is required. Styx is originally developed as a network protocol in Inferno OS, and it brings a considerably amount of overhead for small (especially low-power) devices. In previous work [13], a hardware Styx protocol stack has been developed as IP-cores for SoC. It is delivered in standard interface and proven to work with a wide range of devices, such as, CPU cores, robot servo controllers, voice synthesisers, etc., and popular wireless communication stacks, such as, WIFI [14], Bluetooth [15], and Zigbee [16]. In this paper, we present a power-area-optimised Styx IP-core.

The rest part of the paper is organised as follows. Section II gives the technical background. Section III describes the design of the Styx IP-core based low-power nodes. In section IV, we illustrate extensions to Styx for mobile resources in the context of sensor nodes in a wireless sensor network environment. Section V describes the further implementation and evaluation, with section VI providing conclusions.

II. BACKGROUND

A. Styx Protocol

Styx was developed for the Inferno OS, designed by Bell Labs and now is a product of Vita Nuova [17]. It is an application layer protocol that working over protocols such as TCP/IP [18], ATM [19], PPP [20], etc. Styx merely requires the underlying network provides an in-order and reliable data channel to send /receive data packet.

A Styx-enabled system can choose to be a client (to connect) or a server (to be connected). Fig.1.a shows the exchange of various Styx messages between a Styx client and server in order to establish an initial connection. The client sends a "Tversion" message containing its Styx protocol implementation version number. After verification, the Styx

[1] This work is part of the DEMOS project undertaken by the AMADEUS Research Centre, University of York, York, UK (http://www.cs.york.ac.uk/amadeus)

server responds by sending an "Rversion". The T/Rversion message also gives server/ client the information about the maximum length of a Styx message that the client/ server is capable of handling. The client then sends a "Tauth" message containing the user name and password required to connect to the Styx server. This information may also be encrypted using any encryption standard agreed upon by both client and server during implementation. The server verifies the username and password and responds with an "Rauth" message on success. Finally, the client issues a "Tattach" message requesting the server to attach it to the server's namespace. The server responds with a "Rattach" message that contains a handle, QID, which contains the unique identification to the root node of its namespace. Once the client receives a "Rattach" message from the server, it is then ready for communication (read/ write files) with the devices connected to the server.

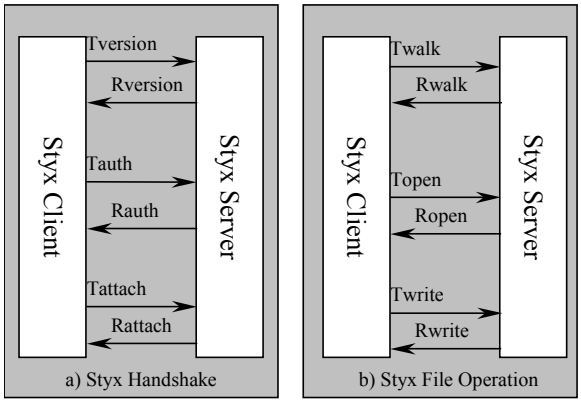

Figure 1. The Styx Protocol Basic Operations

Fig.1.b describes the Styx messages exchanged between client and server depicting a file-write operation by a Styx client. The client issues a "Twalk" message to navigate to the file that it wants to write to. The server changes the client's working location to the requested node if such a file exists and the client has sufficient permissions to access it. On success, the server acknowledges the client by sending an "Rwalk". The client then issues the "Topen" message containing the identification number (FID) of the file to be opened. This FID is unique and local to every client. On the server side, the FID is associated to a QID in its namespace. Thus, on the client end, it is possible that two or more FIDs point to the same QID on the server side. This is how every client has its own copy of the namespace by using their own FIDs and manipulating them as per their requirements. On receiving Topen message, the server associates the client's FID to the device files QID and replies with an "Ropen" message.

To write data (either command to the associated device, or mere data) to the opened file, the client now issues a "Twrite" containing the data to be written. It is possible for the client to issue more than one "Twrite" message if the length of data to be written is greater than the maximum Styx message length, which is implementation dependent. Each such message is tagged by a message identification to provide information to the server about the order in which it has to write the data to the file. The server may not literally write the data it receives. It will decode the data field for commands and carry out the required operation(s) on the actual device accordingly. If appropriate the server now changes the information present in the device's status file. The server sends an "Rwrite" message containing the information about the number of bytes written to the file for verification. When there were multiple "Twrite" messages sent by the client, the server also replies in equal number of "Rwrite" message carrying the same message tags.

Finally, if required, the client issues a "Tclunk" message to close the opened file. The server carries out the required operations (e.g. disabling a device, putting it to sleep, etc.) on the device and replies with an "Rclunk" message. For any error encountered by the server during its operation, a "Rerror" message is sent to the client describing the error.

B. Remote Access via the Styx File System Abstraction

Remote access via file system abstractions has been utilised in many OSs. However, this is conventionally restricted to true data files rather than devices. In typical Unix style implementations (e.g. SunOS, Linux, BSD [21, 22]) and Windows (i.e. Samba [23]), devices are not exported and hence are not available to remote applications via the virtual file system. The usual work around is the construction of local applications (or kernel level) servers to handle remote accesses to devices. In contrast, Styx represents each device/ resource on the network as a single or multiple file(s), so providing distribution transparency over all resources to applications. Remote clients can open / close / read / write remote devices represented by files. For example: a device, A can access a device/resource, R on the network in the form of file, F. The device can then simply use the other device/resource as if it were a local file. Any open/ read/ write/ close operations performed by the device on file, F directly affect the actual network device/resource, R.

Styx allows chaining device accesses across multiple nodes. Thus, an application can access a device via an intermediary node (acting as communication bridge). This feature removes the need for total direct connectivity of all nodes (and connected devices) in the system. Thus, if intermediary serves the third device's attached namespace via its Styx server, the application can access the device files of third device (via the intermediary) using the Styx protocol. Initially, it connects to the intermediary and then starts using the device files of the third device as its local files. The overhead is a two-level of indirection, which is inevitable without total connectivity.

C. Styx Hardware Implementation

A hardware implementation of the basic Styx protocol has been developed as a SoC component in previous research [13]. It has a simple command set to enable control by a CPU, using a register / interrupt interface. A lightweight variant of the Styx SoC component was developed to interface to devices directly. This allows remote clients to interact with the device via a Styx namespace without any CPU and software overhead.

III. DESIGNING LOW-RESOURCE REMOTE NODES

This section describes the design of low-resource remote nodes that utilize the Styx approach. As an example, sensor-nodes are considered.

A. Low-Resource Sensor Networks

A data centric sensor network processes the acquired data on central processors, while the sensor nodes only handles data acquisition, sensor management and transmit/ receive [24, 25]. This is illustrated in Fig. 2, comprising: sensor/ Analog-to-Digital converter (AD), sensor controller/ namespace, Styx server, transceiver and power unit. The AD converter feeds the digitised data collected from the sensor to the sensor controller, which may use a RAM-based data buffer if the sensor node requires large amount data storage. These data are presented as a file on the Styx server's namespace, for example "/sensor_1/data". Other resources are also mapped to the Styx server, and presented as files to the remote client. For example, the sensor controller and status register can be mapped as "/sensor_1/control" and "/sensor_1/status", respectively. Thus, the Styx server establishes a connection with file system abstraction, and do not require CPU, OS, and software.

The Styx server has three main features that contribute to low-power sensor management. Firstly, a hardware implementation requires less resources than software solutions. We note the hardware Styx server can be run at a lower clock frequency to achieve comparable performance a CPU; and will require less power. Secondly, RAM is only needed when the sensor requires data buffer, e.g. for an image / voice sensor, a fast-sampling sensor, etc. In most cases, the sensor only produces a small amount of data that can be buffered in FIFO, registers, or even sent by the transceiver in real-time. In the above case, the sensor node requires no RAM at all; therefore, the power consumption of RAM chip can be eliminated. Finally, the sensor, AD and sensor controller can be turned off from the power unit. This cuts the power dissipation from those devices, and the Styx server remains in idle mode, which only uses a small amount of power. In the extreme case (a timed sleep mode), even the transceiver and the Styx server itself can be powered off, leave only the real-time clock running to wake up the sensor node after a certain period. However, unlike a woken-up CPU, which boots and subsequently executes the software code, the Styx server needs only a few clock cycles to initialise devices, i.e. transceiver, A2D and sensor.

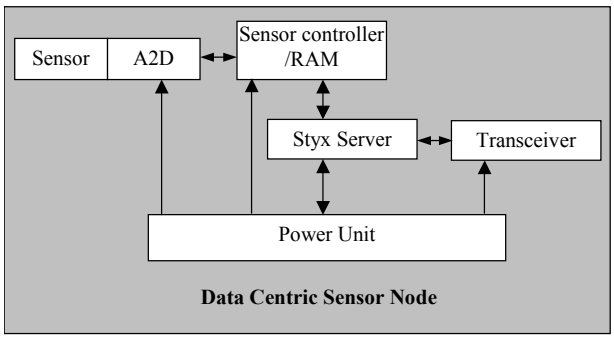

Figure 2. Conventional Sensor Node

B. Sensor Nodes for Distributed Sensor Networks

A distributed sensor network processes the acquired data on the sensor nodes [26]. Subsequently, it sends the results to the data centre or discards invalid data. Compared to data centric sensor node, distributed sensor nodes require more flexibility for data processing algorithm, e.g. Fast Fourier Transmission (FFT) [10], data compression, Motion Estimation (ME) [26], etc. Therefore, the sensor node still requires a general-purpose processor or corresponding architectures, e.g. a hardware FFT engine, to perform data processing. In this case, Styx IP-core can also provide further performance gain and power reduction. The low-power sensor node using Styx server is shown in Fig.3. The data acquired and the processing results can be presented in files, i.e. "/sensor_2/data" and "/sensor_2/results", respectively. The status read and sensor management can also be achieved by reading from "/sensor_2/status" file, and writing command to "/sensor_2/control", respectively. Interestingly, the CPU and RAM resources can also mapped to the namespace of the Styx server to allow debug and direct RAM management, such as remote debug.

Once the data to be transmitted is ready, the CPU simply writes data to the Styx server as a standard bus slave device, which subsequently sends the data via network transceiver. Upon the reception of a new message from the remote client, the Styx server notifies the CPU by interrupts or status registers, which is frequently checked by the CPU. Hence, there is only a small amount of software overhead of using a hardware Styx component on the CPU-based sensor nodes to provide file system abstraction and extra flexibility, i.e. CPU/ RAM management and debug on the fly.

In common with the data centric sensor node, we note that most of the devices, including sensor, AD, processor and RAM can be powered off. Compared to conventional CPU sleep mode, where the CPU checks for events periodically, the device power-off mode is more power-efficient. Once an event has occurred (a new message is received), the Styx server powers on the CPU, and while boot up the software, it can simultaneously respond to most requests, e.g. status check, acknowledgment, etc., because the Styx component works parallel to the processor. The timed sleep mode, in which the transceiver and Styx server are further turned off, is also supported to achieve maximum power save when the sensor node is not used.

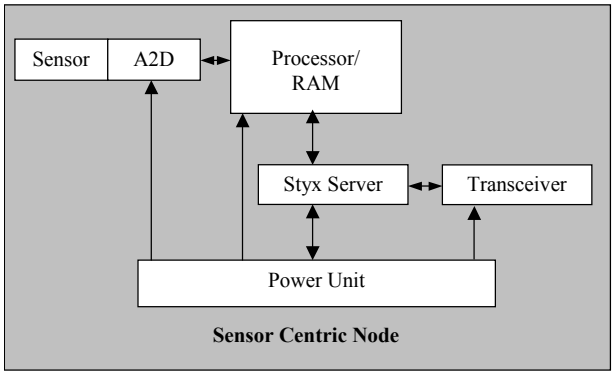

Figure 3. Low-power Sensor Node for Distributed sensor Network

C. Cluster Head for Sensor Network

The cluster architecture is an example architecture in sensor networks [5, 27]. A Cluster Head (CH) is a more complicated

sensor node that may be responsible for managing a group of similar sensor nodes and locally carry out data processing. Similar to other sensor nodes, the cluster head has processor, RAM, transceiver, Styx server and maybe a senor, as shown in Fig.4. Compared to ordinary sensor nodes, it requires more processing power and flexibility. In order to connect to the sensor nodes in the cluster for management, the CH also has a Styx client IP-core connected through a shared bus. Both Styx client and server are memory mapped to the processor, which instructs the Styx client/ server by macro commands, e.g. mount, read, write, etc. The actual Styx messages are hidden from the sensor managing programs that are run by the cluster head. If the transceiver requires software protocol stack, it can also be connected to the system bus to exchange data with the processor, as shown by the dash-line arrow in Fig.4.

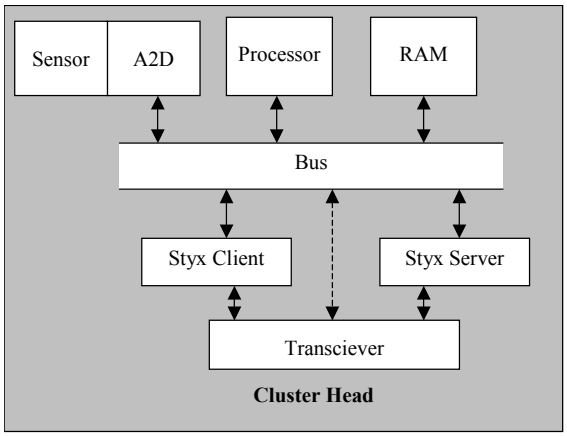

Figure 4. Cluster Head

D. Performance-cost Trade-off

Ideally, the Styx-server IP-core is fabricated in the transceiver's ASIC to give maximum performance with lowest power consumption. However, it is often expensive to build such an ASIC, nevertheless the dedicated Styx server is a tiny device that can be fitted into a small low-power FPGA or even a CPLD. This gives a cost-effective solution to low-power sensor nodes. For data centric sensor nodes, such a FPGA/CPLD with sensor controller can be used instead of an embedded CPU. For data distribution sensor networks, such a FPGA/CPLD can be used along with an embedded CPU, such as SA-1110 [28], Geode GX1 [29], or a built-in CPU core, such as MicoBlaze [30].

IV. MOBILITY IN STYX-BASED NETWORKS

The Styx hardware component supports a number of different sensor network architectures and modes of operation. In this section a number of modes of operation for a cluster based architecture are described to show how a basic cluster-head style architecture can function using Styx; how faulty cluster heads can be tolerated; how mobiles sensors can be incorporated by changing their cluster head allocation.

A. Architecture

In Fig.6, a basic sensor network configuration is given. The data center (DC) executes the application that processes the sensor data returned by the sensors via the cluster heads (CH). The intention of the CH nodes is to enable a single CH to interact with a number of sensor nodes via low-power wireless links, enabling the sensors themselves to turn off between reading the environment (where this is deemed necessary by the application). The CH nodes themselves can connect to the DC using higher-powered wireless links, and are intended to be active (or at least not completely powered down) for the duration of system execution. The DC itself is intended to be mains powered.

In terms of Styx nodes, the DC is a client; the CH nodes are client / server nodes; all sensors are servers. The system initiates by the DC interacting with the CH nodes via "Tversion" messages, with the CH nodes returning "Rversion" messages. Likewise, the CH nodes send "Tversion" messages to nodes, the nodes replying with "Rversion". All CH nodes are able to mount the namespace of their connected sensors (using a "Tattach" message). This namespace can be exported in turn to the DC. As far as the application is concerned, it is able to read sensors by merely reading appropriate files in its local filesystem.

One strength of Styx is the ability to provide different namespace views to reflect differing aspects of the architecture. This is shown in Fig.5 where three views are provided: data/application; physical and control. Essentially, the data/application view enables the DC application to open and read sensor values via the directory /sensornet/data/ where separate sensor directories exist, one per sensor: e.g. /sensornet/data/SN_1 contains Data and Status files, the former containing sensor readings, the latter current sensor status.

The physical view (/sensornet/phi/) provides ability to directly access the hardware of the CH and sensors (registers / memory). This is important for low-level status, debug or potentially upgrade. Similarly, the control view (/sensornet/control) provides access to the Styx internal registers and memory for both the CH and sensors.

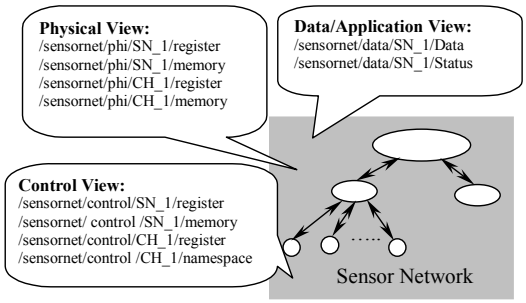

Figure 5. Different Views of Sensor Network Namespace

B. Basic Operation

To enable low-power operation, a sensor needs to power-down between successive readings. However, this is slightly problematic with a file-based protocol such as Styx, which needs the server to respond to client requests, rather than "come

alive" and asynchronously send a message to the server. Therefore, when the Tread message is sent from DC (via CH) to the sensor, it uses this as a key to sleep until the next time it requires a sensor reading; when the time occurs, it wakes up, makes the reading, and replies using a "Rread" message to the DC via the CH. The DC responds by issuing the next Tread. The sensor uses this as an acknowledgement that the DC has received the message; and as initiation of the next request-reply cycle of messages from the DC to the sensors. This is illustrated in Fig.6, where sensors only need to exchange messages with their allocated CH.

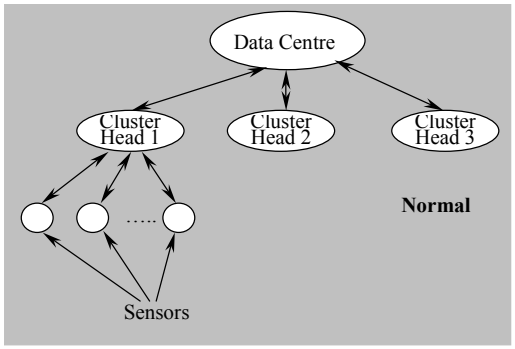

Figure 6. Basic Operations

C. *Coping with Faulty Cluster Head Nodes*

Tolerating CH node failures is key to providing a reliable sensor network [3, 8, 5]. The loss of a CH node (or the wireless link to the CH from the DC) can be tolerated with little additional overhead or change to the normal operation outlined above. Essentially, the fault needs to be detected by both DC and sensor(s), with the latter establishing contact with the DC via a different CH – this is illustrated in Fig.7.

Fault detection is achieved either by the DC noting the lack of Rread messages from a CH - a loss may indicate a faulty CH, multiple losses almost certainly indicate a problem. In either case, the DC can attempt to read the status of the CH, utilising the physical namespace. The CH should respond immediately unless it is faulty, or the communications link is down.

To recover from the fault, the sensor must also detect the fault. This is relatively easy, as both potential faults are apparent at the sensor when the DC does not acknowledge an "Rread" from the sensor with the next Tread message. At this point, the sensor notes that the route to the DC is faulty and waits for communication to be established via a different CH (if one is within communications range). From the DC, if a CH is determined unavailable, it needs to instigate the set-up of a new "route" to the sensor via a different CH. This proceeds by appending the list of connected sensors for the chosen CH (or CHs) with those of the failed CH. This results in the CH attempting to initiate contact with the sensor via a Tversion message (in the same manner as the normal operation detailed above).

The faulty CH (or DC-CH link) is polled ("Tversion" messages) to see if it recovers from a transient fault. If it does, then sensors that have moved to a different CH as part of fault recovery can be handed-back to the original CH via the mechanism below for mobile sensor nodes.

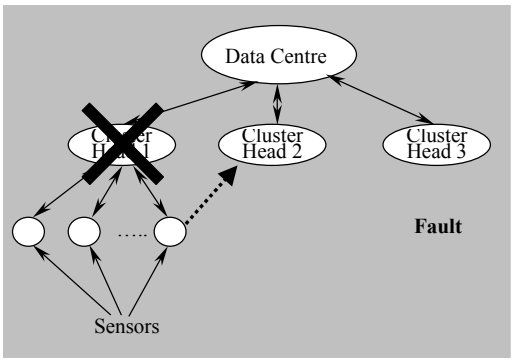

Figure 7. Coping With Faulty Cluster Head

D. *Coping with Mobile Sensor Nodes*

For sensor net architectures where sensor (and / or CH nodes) are mobile [3, 9, 31], a degree of dynamic reconfiguration of the network is required to ensure DC to sensor data is not lost, or at least any loss is minimised. Normal operation (described above) essentially has a fixed route from DC to a given sensor via a specific CH. If the sensor moves out of range of the CH (or vice versa), this route needs to be re-configured. Problems with mobility can be solved using a similar fault-tolerance mechanism to that outlined above.

From the DC, the sensor appears to be faulty, in that an "Rread" reply to the previous Tread does not return, even allowing for the time between successive readings (during which the sensor is powered down). However, the status of the CH can be read by the DC and so it can imply that the CH and CH to DC link is not faulty - this is different to the fault scenario outlined in the section above. The DC now instigates a reconfiguration of the sensor net (CH nodes and sensors) to recover the "missing" sensors.

Essentially, a new route needs to be established between the DC and sensor with minimal overhead or potential for lost data. This can be achieved transparently (from the perspective of the application on the DC) using Styx. This is illustrated in Fig.8, where the sensor moves out of range of CH1 (to which it was originally allocated) and into the range of CH2. By a simple exchange of namespace routing information between CH1 and CH2 (potentially via DC if CH1 and CH2 cannot communicate directly), the route is "handed-off" from CH1 to CH2 without having to re-initialise.

Note that both the CH nodes and the sensor are aware of the loss of route. However, the sensor is not able to determine whether the CH itself is faulty (as described in the previous section) or that the route is being reconfigured. Therefore, when the sensor notes the problem (i.e. timeout waiting for a Tread message) it waits for the reconfiguration to complete.

Reconfiguration of the route from a CH node is achieved by the CH writing the route into all other CH node route tables (under the /sensornet/control namespace). The other CH nodes interpret such a write as a need to send a Tread to the sensor.

The sensor, unless it has moved out of range of *all* CH nodes, will receive at least one Tread (depending upon how many CH nodes are within range). However, it will only respond to one, establishing the route (which could be with the original CH if it has moved back into the range of that sensor).

Note that to establish contact with a new sensor, the CH could utilise a "Tversion" message, i.e. re-initialise connection. However, this implies that the CH would have to instruct the DC to unmount the namespace associated with the missing sensor, and then re-mount it (i.e "Tattach") via the new CH. Whilst this is feasible within Styx and the architecture described above, the alternative and more efficient approach presented above, that is CH nodes exchanging Styx route configuration data, enables the namespace to be served by the new CH without the need for re-mounting by the DC. This implies that the reconfiguration is transparent as far as the application is concerned – noting that at most one sensor reading should be lost, i.e. the "Rread" message that was sent by the sensor, but never received by the (out-of-range) CH. Clearly, a simple resend by the sensor could circumvent this problem.

Clearly, this mechanism for handing-off namespace serving between CH nodes is powerful, and can cope with many scenarios, e.g. load-balancing between CH nodes.

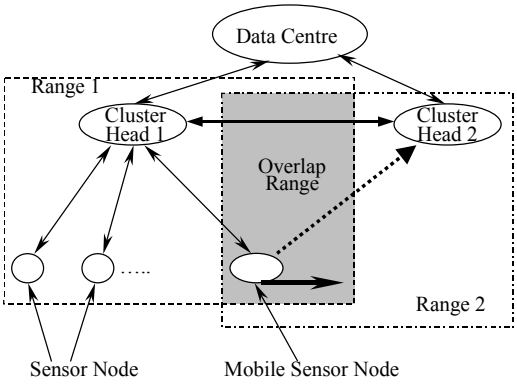

Figure 8. Coping with Mobile Sensor Nodes

E. Coping with Faulty Sensors

To cope with permanent and / or transient faults at a sensor node, the mechanisms described above can be utilized. From the CH and DC perspective, an excessive time is taken by the sensor to reply to send the "Rread" message (even allowing for the time the sensor is in low-power mode between successive readings). At this point, the CH and DC cannot tell whether the sensor has moved out of range, so requiring a reconfiguration of the route, or is indeed faulty. Hence, the CH will presume that the sensor has moved, and instigate a route reconfiguration. However, to ensure that if the sensor fault is transient, and it has *not* moved, the CH itself will send a Tread to the sensor (as per the reconfiguration protocol above). Clearly, if no CH is able to establish communication with the sensor, it is presumed permanently failed. This is established by allowing each CH to retry the Tread message a fixed number of times, with fixed timeouts between (both figures being application dependent).

F. Maintenance, Debug and Upgrade

In system development and operation, it is important that sensor node functionality can be changed or upgraded whilst sensors are in the field. Within the Styx sensor net architecture model considered in above, this is relatively straightforward. To change the operation of the sensor we can utilize the physical view namespace to alter hardware registers and/or memory.

V. IMPLEMENTATION AND EVALUATION

A. General Implementation Results

In order to verify the architecture and demonstrate the performance of the sensor node, a prototypeof the data centric network node was implemented featuring full Styx I/O support, i.e. file open, close read and write, small in size and standard parallel interface. The implemented sensor node is based on "9P2000" version Styx protocol, which is used in the current Plan9 / Inferno OS. It supports up to four files in a 32-byte register-based namespace. It only used 70% of the macro cells and 48% of the I/O pins of a 1.8V, 384-cell Xilinx CoolRunner2 CPLD (XC2C384-6-TQ144) [32]. It has been interfaced with a Zigbee transceiver to allow remote access from a client. It has been tested in both standard Inferno OS environment and standalone Styx software libraries on Windows / Linux environment. The results show the implemented Styx IP-core can provide file system abstraction to sensor nodes.

B. Performance Evaluation

In order to evaluate the performance of the Styx IP-core over the Styx software component, we conducted the following test cases on both the software (standalone) as well as hardware IP-core Styx components:

- use an Inferno shell running on a different machine connected to the test machine on serial line to connect to the Styx server. Typically, we use the command – "mount /dev/eia0/n/remote" to connect to a Styx server. This command makes the client send "Tversion" and "Tattach" messages to the server. The Styx server on authentication replies with the corresponding "Rversion" and "Rattach" messages.
- next we traverse through the mounted remote namespace and write to a file. This action makes the client generate "Twalk", "Topen", "Twrite" and "Tclunk" messages. Thus, the server needs to carry out any required action and reply the client with corresponding "R" messages. For better comparison we wrote to the file twice – initially with a short data (8 bytes) and then with considerably large data (256 bytes).

We record the time taken by the Styx server to decode each of the "T" messages from client and time taken to encode a reply ("R") message. The choice of the above test cases is particularly because they make the Styx component generate almost all the possible Styx messages allowing for detailed analysis. Comparing against the Styx standalone software

component gives us precise measures of the performance improvements gained by the hardware implementation of Styx.

1) Styx Software Component

This was implemented on a 300MHz Geode GX1 embedded processor with 64MB SDRAM memory. The design and implementation of this software component is exactly similar to the hardware Styx IP-core described in the previous sections. When compiled, the standalone Styx component is 59KB in size.

TABLE I. PERFORMANCE OF STYX SOFTWARE COMPONENT (μS)

Message Type	Length (bytes)	Decode Time	Encode Time	Misc. Time	Total Time
(T/R)version	19(T)/19(R)	4.91	8.47	3.35	16.73
(T/R)attach	24(T)/20(R)	5.42	6.77	3.07	15.26
(T/R)walk	17(T)/35(R)	50.22	7.77	4.41	62.40
(T/R)open	12(T)/24(R)	3.5	8.37	3.08	14.95
(T/R)write (8 bytes)	33(T)/11(R)	657.35	5.22	1.65	664.22
(T/R)write (255) bytes	281(T)/11(R)	7315.4	7.34	1.19	7323.9
(T/R)clunk	11(T)/11(R)	2.93	4.30	2.16	9.39

Table I shows the decode/encode time taken by the software only solution. Every received "T" message from the client must have a reply "R" message. Thus, each row in the table describes one complete cycle from "T" to "R" messages. The length field describes the lengths of the message received ("T") from client and the message sent ("R") to the client. The Decode time refers to the time taken by the server to decode the received ("T") message including the time needed to carry out the required operations (e.g. open/read/write a file/device). The Encode time refers to the time taken by the server to encode and prepare the reply ("R") message. The Misc. field refers to the time spent by the server in doing other miscellaneous activities like book-keeping, device access, etc.

2) Hardware Styx IP-core

We tested the Styx IP-core-based sensor node described in section IV.A, at 25MHz. Applying the same test criteria to the hardware Styx IP-core we obtained the results as shown in Tab. II. The similar length of messages in both software and hardware implementation confirms that it is compliant with the Styx protocol.

3) Performance Comparison

A comparison of the performance between the software and hardware implementation of Styx protocol stacks describe in the above two sections are given in Fig.9. It plots a graph of the total cycle time values in tables and to assess the performance of Styx software and the Styx IP-Core. It can be seen that compared to the Styx software implementation, significant improvement has been made by the hardware IP-Core in terms of speed. For example, the total cycle time to "walk" to a file is 62.4μs and 2.04μs respectively for the software and hardware versions. Also, it is clear from the performance tables in the above sections that the Styx IP-Core outperforms the software counterpart by several orders of magnitude in "Twrite" message.

TABLE II. PERFORMANCE OF STYX HARDWARE COMPONENT (μS)

Message Type	Length (bytes)	Decode Time	Encode Time	Misc. Time	Total Time
(T/R)version	19(T)/19(R)	0.84	0.84	0.08	1.76
(T/R)attach	24(T)/20(R)	1.04	1.04	0.08	2.16
(T/R)walk	17(T)/35(R)	1.00	0.96	0.08	2.04
(T/R)open	12(T)/24(R)	0.56	1.20	0.08	1.90
(T/R)write (8 bytes)	33(T)/11(R)	1.40	0.52	0.08	1.90
(T/R)write (255) bytes	281(T)/11(R)	11.32	0.52	0.08	11.92
(T/R)clunk	11(T)/11(R)	0.52	0.45	0.08	1.05

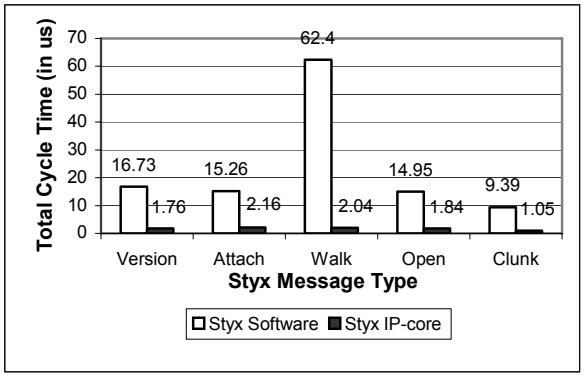

Figure 9. Comparison of Performance

C. Power Consumption Evaluation

The described sensor node is synthesised using Xilinx CoolRunner2 CPLD technology. The integrated power tool, XPower, gives the following power consumption results, as shown in Tab. III. The results include the power consumption in four different scenarios: worst case mode, normal mode, idle mode and sleep mode. In the worst-case mode, we assume the circuit has a 100% switch rate at 25MHz. Clearly, for sensor nodes it represents the power consumption in full-processing power peak. In the normal mode, we adopted 12.5% switching rate, which is often used as a common benchmark for power estimation. This could address the combinational power consumption in a long-term operation. In idle mode, most parts on the sensor node remain quiescent, leaving only the transceiver interface to detect valid input from the transceiver. Finally, in sleep mode, the sensor node is completely powered off, except the real-time clock.

TABLE III. POWER CONSUMPTION RESULTS

	Power Consumption (*mW*)
Worst Case	150.87
Normal	37.53
Idle	11.59
Timed-sleep	0.04

We compared the power consumption of the proposed Styx-IP based sensor node to a 3.3V 206 MHz Intel Strong Arm embedded processor (SA-1110), as shown in Fig.9. The

average power consumption of the processor is from related documents [33]. Also, note the power consumption in sleep mode is considerably less then others, and cannot be visibly represented in the figure. It can be seen that in the worst case, the power consumption has been reduced to 15% (150/1000) of the embedded processor solution. It has been reduced to 9.25% (37/400) and 11% (11/100) of the embedded processor solution, in the normal and idle mode, respectively.

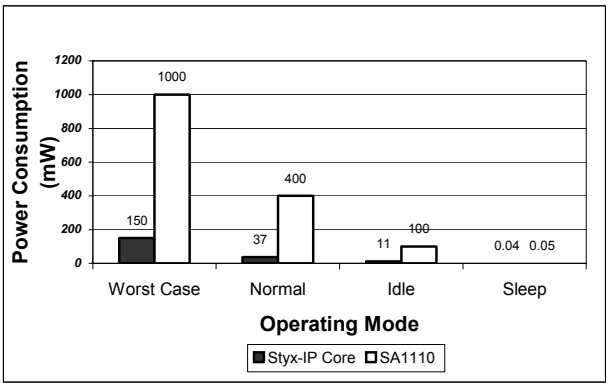

Figure 10. Comparison of Power Consumption

VI. CONCLUSION

This paper has described an approach for implementing sensor networks based upon the Styx file protocol. Styx is a powerful approach for sensor networks as it provides a simple file namespace over the entire sensor network; and also different views of that namespace to allow applications to deal simply with sensor data; and the system infrastructure to control faulty nodes, and provide facility for the architecture to change – i.e. mobile sensors – without changing the simple filesystem view of the sensor network by the application.

The paper describes a hardware implementation of Styx that enables low-power sensors without the overhead of a CPU and OS. The implementation results shows our hardware low-power sensor node is several orders of magnitudes faster than software implementations (11.92 : 7323.9, T/Rwrite 225-byte,Tab.I and Tab.II), and it only uses 9.25% (Fig.9) of the power consumption of a conventional embedded processor based sensor node.

REFERENCES

[1] S. Dorward, R. Pike, D.L. Presotto, D.M. Ritchie, H. Trickey, and P. Winterbottom, "The Inferno Operating System", Bell Labs Technical Journal, 1997.

[2] N. C. Audsley, and A. Patil, "DEMOS - implementing Operating System communication components on FPGA", Proc. Embedded Real-Time Systems Implementation Workshop, Lisbon, Portugal, July, 2004

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," IEEE Comm. Mag. pp.102–114, Aug. 2002.

[4] A. E. Cerpa, and D. Estrin, "ASCENT: Adaptive self-configuring sensor eetwork eopologies", IEEE Trans.on Mobile Computing, 3(3), 2004.

[5] C. Schurgers, V. Tsiatsis, and M. Srivastava, "Stem: topology management for energy efficient sensor networks," Proc. IEEE Aerospace Conference, pp. 78-89, Mar. 2002.

[6] J. Broch, D.A. Maltz, D.B. Johnson, Y. Hu, and J. Jetcheva, "A performance comparison of multihop wireless Ad Hoc network routing protocols," Proc. ACM/IEEE Int'l Conf. Mobile Computing and Networking (MobiCom '98), pp. 85-97, Oct. 1998.

[7] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for Wwreless sensor networks," Proc. IEEE Computer and Comm. Soc. (INFOCOM), pp. 1567-1576, June 2002.

[8] G. Hoblos, M. Straoswiecki, and A. Aitouche, "Optimal design of fault tolerance sensor networks," IEEE Int'l. Conf. Cont. Apps., Anchorage, AK, pp. 467-472, Sept. 2000.

[9] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie , "Protocols for self-organization of a wireless sensor network," IEEE Personal Communications, 2000.

[10] R. Min, M. Bhardwaj, S. Cho, E. Shin, A. Sinha, A. Wang, and A. Changdrakasan, "Low-power wireless sensor networks", Proc. Conf. On VLSI Design, pp 205-211, 2001.

[11] S. Tilak, B. Pisupati, K. Chiu, G. Brown, and N. Abu-Ghazaleh, "A file system abstraction for sense and respond systems," Proc. USENIX workshop on End-toEnd, Sense-and-respond systems, Applications, and Services, pp 1-6, June, 2005.

[12] C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," IEEE Pers. Commu., pp. 52-59, August, 2001.

[13] N.C. Audsley, R.Gao, and A. Patil, "The Styx IP-core for ubiquitous network device interoperability," IEE Proc. Perspectives in Pervasive Computing, October, 2005.

[14] IEEE.org, "IEEE 802.11b/g specifications," online: http:// grouper. ieee. org /groups/802/11/

[15] Bluetooth.org, "Bluetooth specifications," online: https: //www. bluetooth. org/spec/

[16] Zigbee.org, "Zigbee specifications," online: http:// www.zigbee. org/ en/ index.asp

[17] Vita Nuova, online: http://www.vitanuova.co.uk

[18] G.R. Wright, and W.R. Stevens, TCP/IP Illustrated, Volume I and II, 1995.

[19] ATM specification, online: http:// wwwcs. cern.ch /public/ projects/ atm/ specs.html

[20] Point-to-Point Protocol Specification, online: http://www.rfc-editor.org/rfcxx00.html

[21] A. Tanenbaum, Modern Operating Systems, Prentice-Hall, 2001.

[22] A. Silberschatz, P. Baer, and G. Gagne, Operating System Concepts, John Wiley, 2005.

[23] A. Silberschatz, and V. van Steen, Distributed Systems: Principles and Paradigms, John Wiley, 2001.

[24] B. Krishnamachari, D. Estrin, and S Wicker, "Modelling data-centric routing in wireless sensor networks" IEEE INFOCOM, 2002.

[25] B. Krishnamachari, D. Estrin, and S. Wicker , "The impact of data aggregation in wireless sensor networks," ICDCS Workshops, 2002.

[26] A Chandrakasan, R Amirtharajah, S Cho, J Goodman, "Design considerations for distributed microsensor systems," Proc. IEEE 1999 Custom Integrated Circuits, 1999.

[27] S. Banerjee, and S. Khuller, "A clustering scheme for hierarchical control in multi-hop wireless networks," IEEE Infocom, Anchorage, Alaska, April 2001.

[28] Intel SA-1110 Processcer, online: http:// developer. intel. com/ design/ pca/ applicationsprocessors/ 1110_brf.htm.

[29] AMD Geode Solutions, online: http://www.amd.com/us-en/ ConnectivitySolutions/ ProductInformation/0,,50_2330_9863,00.html.

[30] MicroBlaze Sofe Processor Core, online: http:// www.xilinx. com/xlnx/ xebiz/ designResources/ip_product_details.jsp?key=micro_blaze.

[31] J. M. Dolan, M.Saptharishi, C.S. Oliver, C. P. Diehl, A.Soto, and P.K. Khosla, "Network of collaborating mobile and stationary sensors," Proceedings of SPIE, 2003.

[32] CoolRunner II Datasheet, online: http://www.xilinx.com/

[33] Xilinx Application Notes, "Low-power Design with CoolRunner-II CPLDs v1.0 (5/02)" online: http:// www.xilinx.com /bvdocs /appnotes/ xapp377.pdf