Middleware Support for Dynamic Reconfiguration in Sensor Networks

Paul Grace, Danny Hughes, Barry Porter, Paul Alcock, Geoff Coulson, Gordon Blair
Computing Department, Lancaster University, UK
Email: {gracep, danny, barry.porter, {p.alcock@lancaster.ac.uk}, geoff, gordon}@comp.lancs.ac.uk

Abstract—**The operational environments of sensor networks will alter over time, often due to hazardous conditions or fluctuating resource availability; however this important characteristic of sensor networks has yet to be fully addressed by current middleware solutions. In this article, we present a reflective middleware solution for co-ordinated dynamic reconfiguration of middleware behaviour across nodes in a sensor network. We evaluate this approach in a real-world case study; firstly, we demonstrate how dynamic reconfiguration optimises the performance and resource consumption of a sensor application, and secondly we illustrate that the costs of reconfiguration are not prohibitive in this domain.**

I. Introduction

Wireless sensor networks are now employed in a wide range of application domains including, for example, environmental monitoring and disaster management. The role of sensor middleware is to support the application developer, and shield her from: i) the complexity of developing applications on heterogeneous low-level hardware such as MICA Motes, Gumstix, or bespoke sensor technology, and ii) routing messages across heterogeneous ad-hoc networks e.g. Bluetooth, 802.11b, Zigbee, and others. Initial sensor middleware solutions have concentrated on common communication abstractions (e.g. event subscription [1], database-style [2], or tuple-spaces [3]) over a variety of ad-hoc message routing strategies [4]. Some proposals have additionally considered the management of resources within sensor networks e.g. power consumption [5]. However, the *dynamic change* of a sensor network's operational environment and general context has yet to be considered. For example, sensor network middleware used in disaster scenarios will typically need to react to increasingly hazardous conditions-e.g. flooding in a monitored location may cause the network to perform poorly or fail. Therefore, we believe that sensor middleware should be inherently adaptive.

In this paper we examine two key aspects of our Gridkit middleware [6], which support dynamic reconfiguration in sensor networks:

- *Pluggable routing protocols, and overlay networks.* In Gridkit, sensor networks are created as virtual network topologies (i.e. overlay networks), which can be selected to support different application requirements. For example, an overlay that conserves power may be most appropriate for one application type, whereas an overlay that focuses on robustness might be better in a different setting. Different overlays optimise for different characteristics, primarily by differing in terms of their topology and in how messages are routed between nodes.
- *Co-ordinated Dynamic Reconfiguration.* To support reconfiguration of overlay behaviour, co-ordinated adaptation of the per-host middleware across all nodes in the sensor network is required-e.g. the routing protocol on every node must be updated. Gridkit introduces the concept of *distributed frameworks* for this purpose; these consist of a set of sensor nodes and a set of distributed reflective meta-protocols that allow i) the inspection of the current network-wide middleware configuration, and ii) the coordinated reconfiguration of software elements across nodes.

To evaluate our middleware we use Gridkit to deploy a real-world sensor application, involving the management of flooding in a river valley in the North-West of England. We concentrate in particular on how our middleware can perform system wide adaptations of the overlay networks to react to changing conditions such as increasing water flow, and sensor failure caused by flooding. We demonstrate that these reconfigurations are beneficial to the operation of the sensor network compared to static deployments, and they do not come at a prohibitive cost in this type of application.

The results in this paper build upon previous work in [7] where we reported on our general approach to supporting distributed reconfiguration in sensor networks. This paper significantly extends this work, with a quantitative evaluation of our approach on the GumStix embedded computing platform and in a real world WSN-based flood predictions scenario. We also in [8] reported on how the GridStix hardware and software platform has evolved over time, however this paper focuses more upon a detailed evaluation of the 'production' GridStix 1.0 platform as was deployed at our River Ribble site from 2005 - 2007.

II. Gridkit Sensor Middleware

Gridkit [6] is a generalized middleware framework that can be specialized to operate in diverse application types (e.g. Grid computing, pervasive computing, and mobile computing). Here, we examine the specialization to the domain of sensor middleware. Fundamentally, Gridkit is constructed as a set of components developed using the OpenCOM v2 [9] component model. This employs a minimal runtime that supports the loading and binding of lightweight software components at run-time.

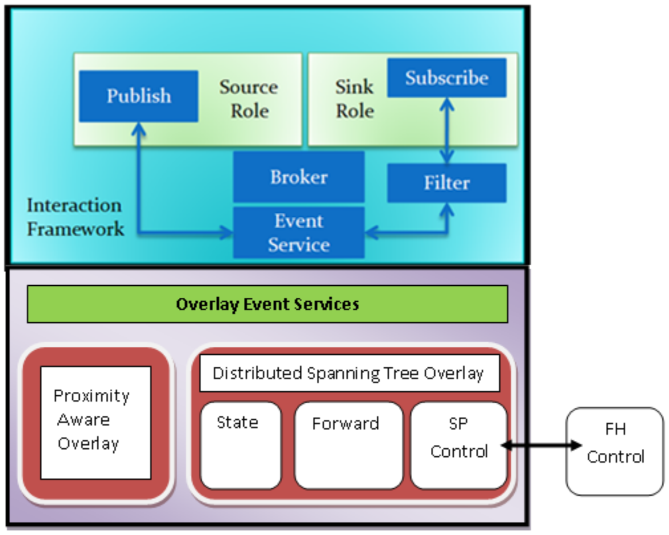

Fig. 1. Gridkit specialised for Sensors

Figure 1 illustrates the key elements of Gridkit's wireless sensor network profile. Applications use the *interaction framework* that contains a customizable event service. A node can be just a publisher, just a subscriber, or, optionally, it can act as a broker in the event service. This is then layered above a core distributed framework known as the *overlays framework*. This hosts, in a set of distributed overlay framework instances, a set of per-overlay plug-in components, each of which embodies i) a *control* element that cooperates with its peers on other hosts to build and maintain some virtual network topology, and ii) a *forwarding* element that routes messages over its virtual topology. Different overlays can be employed here depending on the conditions, e.g. a distributed spanning tree overlay, or a proximity aware overlay. Each overlay can be dynamically reconfigured to react to changing conditions (by changing the control and/or forwarding component). For example, figure 1 shows an example reconfiguration of the distributed spanning tree overlay; we can reconfigure the topology of the overlay from a shortest-path tree to a fewest hop tree by replacing the control component. This process is performed across all nodes who are members of the overlay using the procedure described in the following section.

III. Dynamic Reconfiguration

There are two important dimensions in the dynamic reconfiguration of sensor networks: *local* and *distributed*. Local adaptation is the dynamic reconfiguration of software elements on an individual node; whereas distributed adaptation is the adaptation of the behaviour across a sensor network. Therefore, a distributed adaptation consists of a series of local adaptations. In Gridkit, dynamic reconfiguration is based around software architecture elements known as *local* and *distributed component frameworks*.

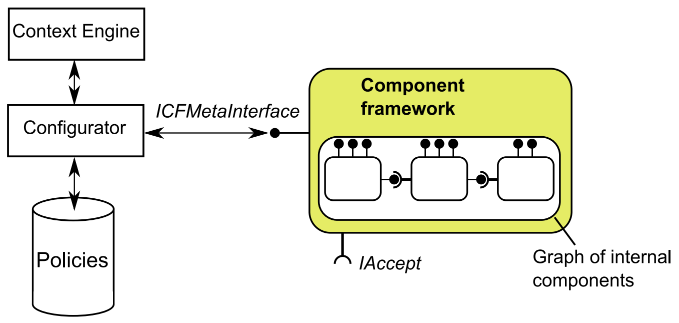

Fig. 2. Local Component Frameworks in Gridkit

A. Local Component Frameworks

The local component framework model (illustrated in figure 2) is based on the concept of composite components as proposed in the OpenORB project [10]. Each framework has a reflective meta-interface (*ICFMetaInterface*) that enables inspection and dynamic adaptation of the local 'architecture' of the composite component in terms of its local components and connections. Additionally, the integrity of each framework is maintained in the face of dynamic change, using developer specified architectural rules plugged into the component framework (through the *IAccept* interface).

The second aspect of the local component framework model is the use of the configurator pattern [11] as illustrated in figure 2. A configurator is assigned to each framework instance, and acts as a unit of autonomy for making decisions about when and how to change the framework. Each configurator maintains a set of local policies for its framework. It is connected with the Gridkit context engine [6] to receive relevant environmental events; and communicates with its host framework through the meta-interface. Gridkit policies use the Event-Condition-Action pattern. When an event is detected, it triggers the corresponding action, which is a reconfiguration script of component inserts, deletes, disconnects, connects, or replaces.

B. Distributed Component Frameworks

A distributed component framework (illustrated in figure 3) is a set of local component framework instances of the same type located across a set of co-ordinated devices, typically providing co-ordinated middleware functionality. The design of the distributed framework model follows the same basic themes as for local frameworks-i.e. the use of reflection to support inspection and adaptation of software, and configurators to enforce autonomic actions. The distributed component framework model must be inherently more flexible than the local model, as there are many more constraints that disallow a single fixed model being utilised. We now discuss the important elements of distributed frameworks in turn.

- **Meta-Object Protocol & Reification Strategies**. Each distributed framework maintains a basic meta-object protocol (MOP) that reifies the information about the global contents of the framework; this can be in terms of node

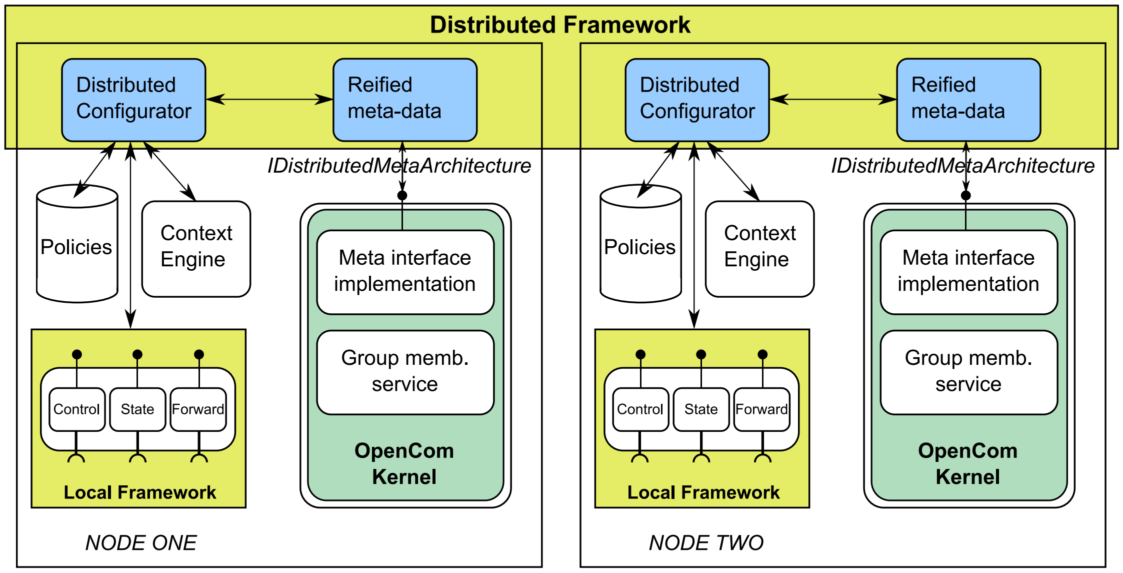

Fig. 3. Distributed Frameworks (per host components)

members, and also the component configurations on each host. The MOP provides operations (through IDistributedMetaArchitecture in figure 3) for the insertion and deletion of local framework elements into/from a given distributed framework, and the inspection of information about the set of individual participants.

- **Group-based membership support for the MOP**. We use a lightweight group membership service as the base mechanism for distributing meta-data and reconfiguration events; this data then builds the view of the system wide architecture. The protocol is customisable: typically different group membership overlays will suit different sensor models-e.g. a sensor network with high node mobility will require a different group membership overlay from a more static network topology. So far, we have used the scalable membership protocol SCAMP [12] to maintain meta-data between members of a distributed framework.
- **Configurators**. Distributed configurators (as seen in figure 3) follow the same pattern as for local frameworks. They receive events about changing environmental conditions, select policies and then perform distributed reconfigurations using the MOP. However, individual frameworks may have more than one configurator (e.g. there could be one on every node). Therefore, consensus protocols are used to ensure that all members of the framework agree on the action to perform. Our evaluation has so far focused on single configurators; however, we are also investigating the introduction of selectable and replaceable consensus algorithms into our distributed frameworks.
- **Quiescence**. For safe dynamic reconfiguration it is important to ensure that updates complete atomically and do not impact the integrity of the network. There are two parts to this: i) making the framework safe to adapt, i.e. placing it in a quiescent state, and ii) ensuring that the reconfiguration is complete and correct. In our current implementation, local framework instances maintain a readers/writers lock to place it into a quiescent state: any standard interface call or meta-inspect is a reader, any meta-write is a writer. Hence, no reconfiguration can take place locally while a thread is executing in the framework. Currently our distributed frameworks use this capability; each local framework is placed into a quiescent state through a command propagated via the meta-group service. Once locally quiescent a notification is returned to the configurator. Upon the condition that all members are in a quiescent state then the reconfiguration continues. After reconfiguration, like with local frameworks (*IAccept* plug-in), the update can be checked through inspection of the meta-data to validate the integrity of component updates across multiple nodes. The disadvantage of this approach is that it may be too resource intensive, and may not scale suitably for large numbers of hosts. Therefore, we are investigating replaceable decentralised strategies for safely updating components.

IV. Case-study Based Evaluation

A. *Background*

We now discuss the use of the Gridkit sensor middleware in an implemented real-world scenario: wireless sensor network-based real-time flood forecasting in a river valley in the north west of England. This scenario is described in additional detail in [13].

In this scenario, a wireless sensor network (WSN) comprising of 20 nodes has been deployed to monitor depth and flow conditions along a 2.5KM stretch of river in the Yorkshire Dales in North-West England. The system monitors water depth using pressure sensors and flow-rate using a combination of image-based flow measurement and ultrasound flow measurement. Sensor data is collected in real-time at one or more designated 'root' nodes and forwarded from there via GPRS to a prediction model that runs on a remote computational cluster. Each sensor node (known as 'GridStix') comprises a 400MHz XScale CPU, 64MB of RAM, 16MB of flash memory, and Bluetooth and WiFi networking hardware (the root nodes are also equipped with GPRS). Each GridStix is powered by 4 watt solar array and a 12V 10Ah battery. They run Linux 2.6, version 1.4 of the JamVM Java virtual machine and the Gridkit version 1.5 WSN profile (figure 1). The following section now discusses the role of reconfiguration in this scenario.

In the scenario we used Gridkit with the distributed spanning tree overlay plug-in, which is used to disseminate sensor data between a large number of sensor nodes and a small number of root nodes. This plug-in is configured to operate in two modes; fewest hop and shortest path:

- *Shortest Path* (SP) spanning trees are optimised to maintain a minimum distance in edge weights from each node to the distinguished 'root' node; in our case edge weights are derived from the power consumption of each pairwise network link; SP trees tend to consume less power than FH trees, but offer poorer performance.
- *Fewest Hop* (FH) spanning trees are optimised to maintain a minimum of hops between each node and the root; FH trees minimise the data loss that occurs due to node failure, but are sub-optimal with respect to power consumption.

The different properties of these overlays make them more suitable for different environmental conditions. During quiescent conditions, when the criticality of sensor data is low, the system is configured to use a shortest path spanning tree. Conversely, during flooding conditions, when the criticality of sensor data and risk of node-failure is high, the system reconfigures to use a fewest hop spanning tree which is more resilient and offers better performance.

B. *Evaluating the Role of Reconfiguration*

The benefits of reconfiguration have been evaluated primarily through simulation of the Gridkit middleware in operation in an example sensor deployment. The *GridStix simulator* models the low-level properties of each node (available CPU, available Battery, solar panel power production) and each pairwise network link (round-trip-time, power-consumption, bandwidth, delay, jitter, loss). This low-level data has been measured empirically on the real-world system, which makes the simulator highly accurate for this scenario. The visualisation sub-system of the simulator is shown in Figure 4 illustrating FH (left) and SP (right) overlay configurations.

The simulation was configured as follows: The simulation period is 24 hours (midnight-to-midnight). Each node enters the simulated period with a battery at 50% charge. Flood conditions begin at 12PM and last until 6PM (the approximate mean duration of a flood event at the site). Dawn occurs

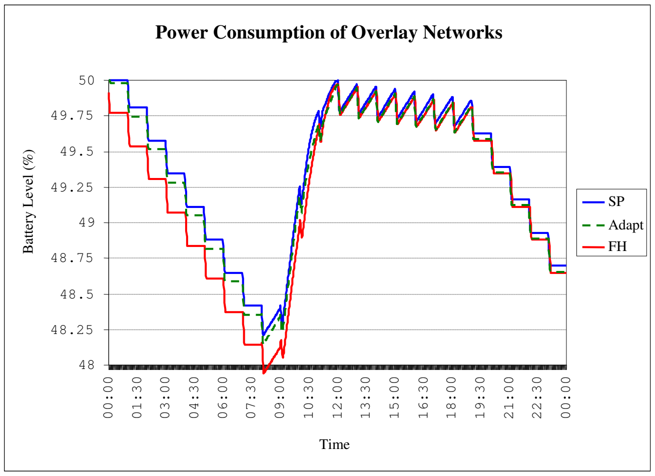

Fig. 5. Power Consumption of Overlay Configurations

at 8AM from which time solar power production is set to WINTER_SUN, when flood conditions begins at 12PM, solar power production is set to HEAVY_CLOUD, finally night falls at 8PM (approximately mimicking late winter conditions, when flooding is most prevalent). All nodes were programmed to wake for one minute in every hour. During quiescent conditions nodes transmit sensor readings at a rate of one per minute. During flooding condition, nodes transmit sensor readings at a rate of one per second. This is commensurate with the increased requirements of performing real-time flood modelling.

Throughout the simulated period, the system's performance has been evaluated in the context of three key metrics:

1) *Performance*: We measure this in terms of the latency with which messages can be relayed from each sensor node to the root node.
2) *Resilience*: The resilience of the network is a function of the extent to which the failure of a given node reduces the overall connectedness of the network. We measure this as the number of viable routes between each node and the root.
3) *Power Consumption*: Although the GridStix are equipped with solar panels, power consumption is still an extremely important factor. We infer this from the per-node battery power consumed throughout the test.

In all cases we measure and plot each of these metrics averaged for all nodes in the network, at intervals of one minute throughout the duration of our trace. Figure 5 shows the battery life of each node in the system using an SP configuration, an FH configuration and an adaptive configuration, wherein when flooding is detected (at noon), the middleware reconfigured from using a low-power SP tree to a high-performance FH tree.

Figure 5 shows that SP trees ensure that nodes maintain the highest possible battery life throughout the test. Conversely, FH trees result in the greatest battery power consumption

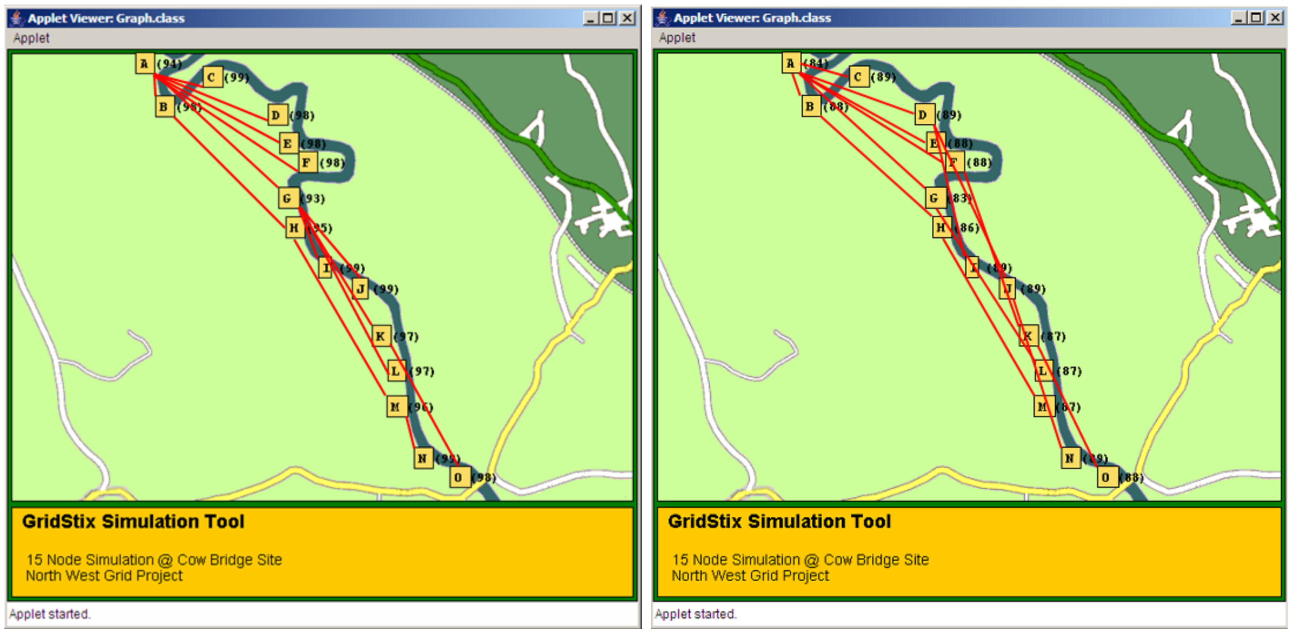

Fig. 4. FH (left) and SP (right) Spanning Trees in the Gridstix simulator

(though at the expense of performance and resilience as shown below). As one might expect, where the system reconfigures at flood time from an SP to a FH configuration ('adapt', shown in green), battery power is maximised during quiescent conditions (i.e. approximating SP), while increasing during flooding conditions (i.e. approximating FH) - though at the same time providing better performance and resilience, as shown in figure 6 and 7 respectively. Finally, the graph illustrates that the power consumed by reconfiguration, both in the transmission of reconfiguration messages and in CPU-usage is acceptably low. In fact, it is too small to be noticeable in figure 5.

Figure 6 shows the mean reporting latency throughout the test period. As one would expect, aside from normal jitter, FH and SP configurations remain relatively constant throughout the test. FH configurations demonstrate a mean reporting latency of 11ms, while SP offers a reporting latency of 28ms (though consuming significantly less power, as shown in figure 5 above). Finally, where the system reconfigures from an SP configuration during quiescent conditions to an FH configuration during flood conditions ('adapt', shown in green), performance is correspondingly low during quiescent conditions, but high during flood conditions (though at the expense of power).

Figure 7 illustrates the resilience of the system to node failure, based upon the mean number of routes that are affected by node failure. As FH trees have a typically lower node degree they tend to be more resilient to node failure, while SP trees have a typically higher degree and are therefore significantly more vulnerable to node failure. As one would expect, during quiescent periods, where nodes reconfigure between overlays during flooding, system resilience matches

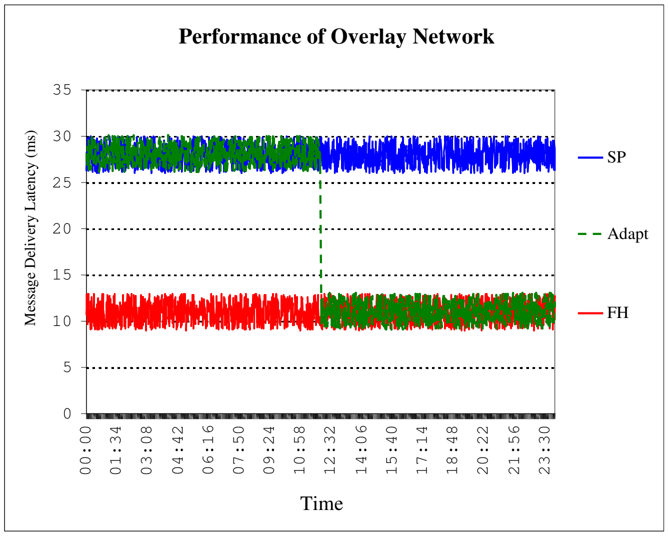

Fig. 6. Performance of Overlay Networks

SP during quiescent periods and FH during flooding periods (i.e. it is more resilient, but at the expense of power).

In summary, testing on a real-world system illustrates the benefits of overlay reconfiguration. By configuring to a low-power overlay during normal conditions and a high performing and resilient overlay during flood conditions, battery life is extended, while maintaining system functionality during critical conditions. We also illustrate that the cost of reconfiguration in terms of additional power consumed, and reconfiguration time does not significantly affect the power costs and performance

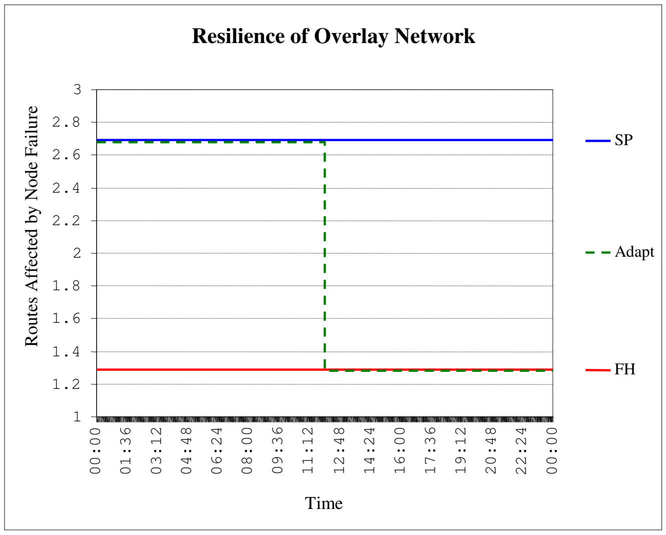

Fig. 7. Resilience of Overlay Networks

of the sensor network.

V. Related work

There are a number of existing sensor middleware, some of which were described in the introduction. These are important solutions in identifying the key characteristics required by sensor middleware, namely easy to use abstractions, suitable routing strategies, and resource management policies. However, none of these pieces of work considers dynamic reconfiguration and customisability to the same degree as Gridkit.

Outside the domain of sensor networks there are a set of technologies related to distributed reconfiguration. Kramer and Magee [14] describe algorithms for distributed reconfiguration that inspired our approach. k-Components [15] offers a decentralised agent-based approach; i.e. each node makes local decisions about adaptation, until a global consensus is reached. Gridkit supports both centralised and decentralised approaches, and in that manner can be tailored to resource-constrained domains such as sensor networks. NecoMan [16] offers an alternative approach to dynamic reconfiguration. It supports safe, co-ordinated updates of distributed services, typically related to network protocols. It has not yet been applied to multiple reconfiguration domains to illustrate its flexibility; however, it presents many interesting ideas that could be applied within our frameworks. Ensemble [17] is a micro-protocol stack framework that is able to adapt its configuration dynamically; however, the reconfiguration mechanism is closely coupled to the micro-protocol implementation, and isn't as re-usable as Gridkit.

VI. Conclusions

In this paper, we have proposed the introduction of dynamic reconfiguration into sensor middleware to ensure that the behaviour of the sensor network is optimised in the face of fluctuating conditions. We have demonstrated how the flexible Gridkit middleware platform can be tailored specifically to the sensor domain. In addition, we have introduced the concept of reflective distributed frameworks to manage the reconfiguration of software components across the sensor network. The approach was evaluated in a real-world sensor application, illustrating the benefits of reconfiguration, which come with acceptable costs.

There are a number of interesting future areas of research inspired by this work. Firstly, the creation of higher-level declarative languages that can be used by both middleware and application developers to describe reconfiguration actions on the sensor network, and also deal with potential conflicts that may arise from multiple policies. Secondly, the introduction of security measures to the distributed framework to ensure only authentic nodes can join a framework, and only members of the framework can make reconfigurations.

References

[1] B. Jiao, S. Son, and J. Stankovic, "GEM: Generic event service middleware for wireless sensor networks," in *2nd International Workshop on Networked Sensing Systems (INSS)*, San Diego, California, USA, June 2005.

[2] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," *IEEE Personal Communications*, vol. 7, no. 5, pp. 10–15, October 2000.

[3] A. Murphy and G. Picco, "Transiently shared tuple spaces in sensor networks," in *Workshop on Middleware for Sensor Networks*, San Francisco, USA, June 2006.

[4] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," in *1st ACM Int. workshop on Wireless sensor networks and applications*, June 2002, pp. 22–31.

[5] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *IEEE International Conference on Communications*, Helsinki, Finland, June 2001.

[6] P. Grace, G. Coulson, G. Blair, and B. Porter, "Deep middleware for the divergent grid," in *ACM/IFIP International Middleware Conference*, Grenoble, France, December 2005.

[7] P. Grace, G. Coulson, G. Blair, B. Porter, and D. Hughes, "Dynamic reconfiguration in sensor middleware," in *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*. ACM, 2006, pp. 1–6.

[8] G. Coulson, D. Hughes, G. Blair, and P. Grace, "The evolution of the gridstix wireless sensor network platform," in *International Workshop on Sensor Network Engineering (IWSNE '08)*, June 2008.

[9] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama, "Opencom v2: A component model for building systems software," in *IASTED Software Engineering and Applications (SEA'04)*, Cambridge, MA, USA, November 2004.

[10] G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski, "The design and implementation of open orb 2," *IEEE Distributed Systems Online*, vol. 2, no. 6, September 2001.

[11] F. Kon, "Automatic configuration of component-based distributed systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, May 2000.

[12] A. Ganesh, A. Kermarrec, and L. Massoulie, "Scamp: Peer-to-peer lightweight membership service for large-scale group communication," in *3rd Int.Workshop on Networked Group Communication*, London, UK, November 2001, pp. 44–56.

[13] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven, "An intelligent and adaptable flood monitoring and warning system," in *5th UK E-Science All Hands Meeting (AHM'06)*, Nottingham, UK, September 2006.

[14] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1293–1306, 1990.

[15] J. Dowling, "The decentralised coordination of self-adaptive components for autonomic distributed systems," Ph.D. dissertation, Trinity College, Dublin, 2004.

[16] N. Janssens, S. Michiels, T. Holvoet, and P. Verbaeten, "Necoman: Middleware for safe distributed service deployment in programmable networks," in *ACM/IFIP International Middleware Conference*, Toronto, Canada, October 2004.

[17] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr, "Adaptive systems using ensemble," *Software Practice and Experience*, vol. 28, no. 9, pp. 963–969, August 1998.