SLIM: Service Location and Invocation Middleware for Mobile Wireless Sensor and Actuator Networks

Gianpaolo Cugola and Alessandro Margara
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy.
[cugola, margara]@elet.polimi.it

April 12, 2010

Abstract

One of the main obstacles to the adoption of Wireless Sensor Networks (WSNs) outside the research community is the lack of high level mechanisms to easily program them. This problem affects distributed applications in general, and it has been replied by the Software Engineering community, which recently embraced Service Oriented Programming (SOP) as a powerful abstraction to ease development of distributed applications in traditional networking scenarios. In this paper we move from these two observations to propose SLIM: a middleware to support service oriented programming in mobile Wireless Sensor and Actuator Networks (WSANs). The presence of actuators into the network and the capability of SLIM to support efficient multicast invocation within an advanced protocol explicitly tailored to mobile scenarios, makes it a good candidate to ease development of complex monitoring and controlling applications. In the paper we describe SLIM in details and show how its performance easily exceeds those obtainable by using traditional approaches to service invocation in mobile ad-hoc networks.

INTRODUCTION

After an initial period of research and experimentation, *Wireless Sensor Networks* (WSNs) (Sohraby, Minoli, & Znati, 2007) and their siblings: *Wireless Sensor and Actuator Networks* (WSANs) (Akyildiz & Kasimoglu, 2004) are entering a more mature phase, with several commercial companies offering products to support a wide range of application domains concerned with monitoring and control. On the other hand, to hold the promise of bridging the gap between the physical and the virtual world, WSANs have still to overcome a major limitation: the difficulty of developing applications on top of a given WSAN platform.

Indeed, usually WSAN programming has to be carried out in a very low-level, system-dependent way. This requires programmers to have a strong technical background in several domains, from system to network, while also resulting in very long development and testing phases: something that is currently limiting the adoption of WSAN technology.

To overcome this limitation, the research community proposed various approaches to raise the level of abstraction for WSAN programming (Picco & Mottola, to appear) but none of them has been widely adopted. Meanwhile, the Software Engineering community is proposing Service Oriented Programming (SOP) as a powerful approach to ease development of complex distributed applications. While others have already proposed using SOP in WSANs (see our section on related works), to the best of our knowledge none has offered a complete and integrated solution to bring the SOP world into the tiny scale of WSANs. In this paper we fill this gap by presenting SLIM: a middleware infrastructure to support service-oriented programming in *mobile WSANs*.

Issues in Mobile WSANs. Several applications of WSANs involve monitoring and controlling mobile entities like people, animals, or goods, through

access and control points that can be also mobile, like PDAs in the hands of operators. This brings the need of considering mobility as a key aspect of a sensor network: an aspect such important because it has a great impact on the communication and coordination layers. Indeed, as the research on Mobile Ad-Hoc Networks (MANETs) shows, the communication layer of a mobile application has to include effective mechanisms to cope with unreliable links and routes. Similarly, ad-hoc coordination mechanisms are required to cope with nodes that may quickly become unreachable due to network partitions, while other nodes may appear as they approach the coordination area. Even the application layer is impacted by mobility, as the location, and more in general the *context* of nodes, becomes an important issue to consider in choosing which nodes to contact.

SLIM addresses these issues by adopting an advanced routing protocol explicitly designed to manage unreliable links and dynamic group membership, while it decouples the service matching policy from the communication layer, allowing applications to choose and install into the middleware the matching component that better fits their needs (e.g., one including contextual information as part of service descriptions).

SLIM also provides an efficient, peer-to-peer service discovery protocol fitting the needs of a mobile network in which nodes (including a registry) may easily become unreachable. It also covers the typical needs of a network designed for sensing, by providing two forms of invocation: unicast and multicast. The former allows service consumers to get data from a single sensor or to send a command to an actuator, while the latter allows to invoke all the services that satisfy a given query at once, e.g., to gather the data produced by a set of sensors in a single step.

Organization of the presentation. In the remainder of the paper we describe SLIM in details. In particular, the SLIM architecture and API is the topic of the next section, then we describe the routing protocol behind SLIM, while in section "Evaluation" we discuss SLIM performances and compare them with those attainable with more classical solutions for SOP. Finally, the "Related Work" section surveys other research results related with SLIM, while in the last section we draw some conclusions and describe our future research plans.

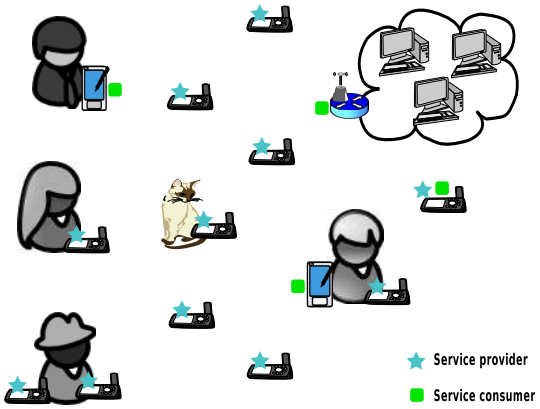

Figure 1: Reference scenario

THE SLIM ARCHITECTURE

The reference scenario for SLIM (see Figure 1) is that of a *Wireless Sensor and Actuator Network* (WSAN), composed of a set of nodes, including fixed or mobile sensor motes, fixed or mobile actuators, PDAs in the hand of operators, and gateways toward a traditional network (a LAN or the Internet). In a SOP style, every node may act as a *service provider*, a *service consumer*, or both. Service providers offer one or more *services* to consumers, each described through a *service descriptor*. Service consumers (or simply *clients*) locate the services they need by passing a *query* to a *lookup service*, and send *messages* to invoke them, receiving other messages back as replies.

As mentioned, SLIM clients may also send messages in multicast, to reach all those services that match a given query. This form of communication, which is not common in the panorama of SOP middleware, not only has the potential of reducing the cost of invocation, through an intelligent use of multicast routing, but it also allows avoiding the service location phase for all those cases, which are common in WSANs, in which the service consumer is interested in invoking all the service providers that offer the same service, e.g., that of temperature and humidity reading in a vineyard.

The kind of applications that may benefit of SLIM include environmental monitoring and actuating, e.g., for precision agriculture (Sikka et al.,

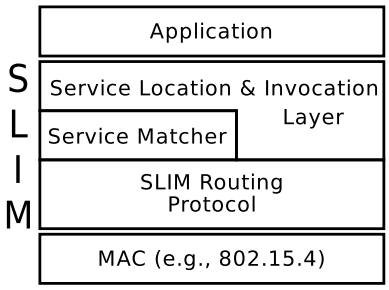

Figure 2: The SLIM internal architecture

2006); monitoring people at home or in hospitals, e.g., for elderly care (*Proc 3rd Int. Conf. on Pervasive Computing Technologies for Healthcare*, 2009); monitoring animals in farms (Andonovic et al., 2009); or monitoring goods moving around, e.g., for supply chain management (Evers & Havinga, 2007).

SLIM supports these scenarios through a middleware explicitly designed to be embedded in small scale nodes like sensor motes, which supports both service location (i.e., discovery) and invocation. Figure 2 shows the internal architecture of the SLIM middleware implemented into each node. On top of the MAC protocol (e.g., IEEE 802.15.4), SLIM includes an advanced routing protocol explicitly conceived to support the SLIM functionalities in a mobile, multi-hop WSAN. Next section describes this protocol in detail. Here we focus on the other layers that compose the SLIM middleware starting from the *Service Matcher*, which implements the logic that allows service descriptors to be matched by service queries. This is a generic component, which can have multiple implementations for different deployments of SLIM. This way SLIM is not forced to use a single service description and query language, but can adapt to multiple languages, from the simplest ones, in which services are described by a small set of attributes as in nanoSLP (Jardak, Meshkova, Riihijrvi, Rerkrai, & Mahonen, 2008), to the most complex ones, like those using XML descriptors and queries. Since SLIM performs service discovery using a query message that reaches every node in the network (more on this later), the matching is solved at each node separately. This reduces the effort that each node has to spend (minimizing power consumption) and simplifies the issue of coding the

```
registerSvc(SvcDescr s, void (*callback)(void *msg,
                int msize, void *reply, int *rsize))

SvcDescr *lookup(SvcQuery q)

void sendMsg(SvcDescr s, void *msg, int msize)

void mcastMsg(SvcQuery q, void *msg, int msize)
```

Figure 3: The SLIM API

Service Matcher, which has only to check if the incoming query matches the services exported by the local node.

On top of this is the *Service Location & Invocation Layer*, which offers a simple API to register a service, discover the services of interest, and invoke them. The main operations provided by this API are shown in Figure 3 using the C syntax, as most of the WSAN platforms use this language (or a dialect, like NesC). The first two operations allow applications to register a service into the SLIM runtime and to locate the services they need. The `sendMsg` operation is used to send messages to single services, chosen among those returned by the `lookup` operation. Finally, the `mcastMsg` operation sends the same message, in multicast, to all those services that satisfy the query `q`.

THE SLIM PROTOCOL

Previous experience with routing protocols (Cugola & Migliavacca, 2009; Baldoni, Beraldi, Cugola, Migliavacca, & Querzoni, 2005; Cugola, Murphy, & Picco, 2006) convinced us that using traditional routing tables, holding the next hop for each destination, together with unicast link-layer transmission to forward packets, hop-by-hop, from the source to the destination, is a bad idea in mobile, pervasive scenarios, like those we target. Accordingly, SLIM uses a radically different approach based on link-layer broadcast transmission, opportunistic forwarding, and soft state.

Brief summary of the protocol. More specifically, each SLIM node keeps a *distance table* storing the distance from those nodes it heard about recently (information in this table expires after a short period of time). During the service location phase the distance table of each node is filled

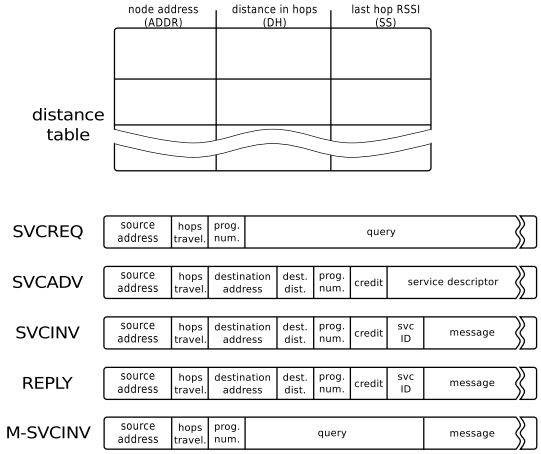

Figure 4: SLIM distance table and packets

with the distance (in hops) from the service consumer who originate the query. This information is subsequently used to bring service advertisements back. At the same time, during this "return" phase, the distance table of the nodes along the route toward the service consumer is filled with information about the service providers who replied. Finally, both information (i.e., the distance from service consumers and service providers) are used to route request and reply messages back and forth. During this message exchange, tables of nodes along the route are renewed, while new nodes have the opportunity of discovering their distance from the communicating peers.

Service location phase. The traditional approach to service location, which uses a single node acting as a registry to store information about all the services available in a given domain, is hardly applicable to a dynamic, pervasive scenario like the one we address in this paper. Starting from this consideration, the SLIM location protocol adopts a different, peer-to-peer approach, which does not rest upon the existence of a specific registry node.

In particular, when a service consumer C invokes the `lookup` primitive, its MAC address and the query describing the services it is interested in are encapsulated into a `SVCREQ` packet (see Figure 4), which is sent out in broadcast (link-layer), while a timeout is started to determine the maximum time to wait for collecting results.

Upon receiving a `SVCREQ` packet, each node:

1. uses the *service matcher* component to check if the query matches one of the services it registered;

2. adds a new record to its distance table (see Figure 4) with the address of C as the first field, the number of hops to reach C (i.e., the hops traveled by the `SVCREQ` packet so far) as field DH, and the RSSI (Received Signal Strength Indication) of the packet as field SS;

3. starts a timer that is inversely proportional to the RSSI of the packet and the remaining battery of the node.

If, during this waiting period, the same packet (each packet is identified through the address of the source and a progressive number) is heard again, the node cancels retransmission; otherwise, when the timer expires, it forwards the packet in broadcast. This algorithm is repeated at each hop with two results:

1. the `SVCREQ` packet floods the network being retransmitted only by those nodes that are farther by the previous forwarder (thus having the opportunity of covering the largest new area, minimizing the number of retransmissions) and whose battery is more charged;

2. the distance table of each node in the network is filled with the distance DH (in hops) from the service requester C, plus the signal strength SS of the last hop traveled.

When the `SVCREQ` packet reaches the provider P of a service that matches the request, a `SVCADV` packet is created, which holds P's MAC address, the address of the requester C (i.e., the source of the `SVCREQ` packet), the distance of C from P, and the descriptor of the matching service.

Like `SVCREQ`s, `SVCADV`s are also sent out in broadcast, but routing is different. Indeed, when a node N hears a `SVCADV` packet:

1. it updates its distance table from P;

2. if its distance from the destination C (field DH of the distance table) is greater or equal to the distance included into the packet itself, N drops the packet;

3. otherwise it updates the distance from the destination recorded into the packet and starts a timer inversely proportional to a combination of DH, SS, and the remaining battery.

As for `SVCREQ`s, if during this waiting period the same packet is heard again, the node cancels retransmission, otherwise it forwards the packet in broadcast.

Under ideal conditions, this protocol results in an efficient, greedy forwarding of `SVCADV` packets, which chooses opportunistically, as forwarders, those nodes that are closer to the destination C (small DH) and whose link from the probable next forwarder are stronger (large SS).

Mobility and local minima. On the other hand, mobility may break this protocol, by introducing local minima in the distance field toward the service requester C. This occurs whenever a node N has a wrong estimate of its distance from C, e.g., because it was once closer to C but now moved in a region where the real distance is higher. Nearby nodes will not forward packets generated by N because of the (wrong) smaller distance it puts in those packets.

To solve this issue we complemented the basic forwarding algorithm above with a *retransmission mechanism*. After transmitting a packet (either as a source or as a forwarder), a node N puts it in a *retransmission queue*. If a predefined *timeout of retransmission* expires without hearing the same packet again, this time including a lower distance from C (this happens when no one re-forwards the packet toward C), the node N increases the distance into the packet by one and transmits it again.

The consequence of this mechanism is twofold: on one hand it increases the protocol's resistance to collisions, which is good since link-layer broadcast is particularly subject to collisions. On the other hand, increasing the distance for packets that where not forwarded by neighbors, it also allows overcoming local minima, by increasing the set of potential forwarders for the retransmitted packet.

Unfortunately, asymmetric links may trigger the retransmission mechanism even when it was not required, thus increasing the network traffic without any positive effect on delivery. To limit this problem we allow each node to retransmit each packet at most once. Moreover, we also introduce in SLIM a mechanism of *credits*, which further reduces retransmission. When a `SVCADV` packet is created, it is assigned a predefined credit (an integer), which is decremented each time the retransmission timeout expires at a node and the packet is retransmitted. When the credit value goes to zero the retransmission mechanism is not used anymore, i.e., the initial credit of a `SVCADV` packet represents the maximum number of times the retransmission mechanism may fire along its route from the service provider to the service requester.

Invocation phase. When a service consumer C invokes the `sendMsg` operation, a `SVCINV` packet is created holding the address of the recipient P, the identifier of the service (both are part of the service description), and the message to be sent.

Once this packet has been created it must be forwarded. If the distance table of C does not hold any information about P (because it expired and had been removed) then the `SVCINV` packet is routed using the same approach adopted for `SVCREQ`s, i.e., using our "smart" flooding protocol. Otherwise it is routed using the same approach used for `SVDADV`s. In the first case it is very likely that the packet will reach its destination: flooding is robust and it rebuilds the distance field toward C that is easily followed back by replies. As a consequence, in this case we do not adopt any mechanism to further increase delivery. Conversely, if the packet is sent using the `SVCADV` approach, it may happens that it gets lost. To cope with this risk, C starts a timer just after sending the packet. If the timer expires without receiving a reply, the `SVCINV` packet is sent again, this time using the flooding approach.

Finally, replies are encoded into `REPLY` packets, which are routed using the same approach adopted for `SVCADV`s.

The last case we have to consider is that of a node C invoking the `mcastMsg` operation. In this case we build a `M-SVCINV` packet, similar to `SVCREQ`s, but including the service query provided by C as an implicit destination address. This packet is routed as `SVCREQ`s, flooding the network in search of matching services. A timeout is used to set the maximum time to wait for collecting results.

EVALUATION

To provide an accurate and replicable analysis of SLIM under different conditions we used the OMNeT++ (Varga, 2001) network simulator. To sim-

ulate a WSAN with mobile nodes we used the Mobility Framework (*Mobility Framework for OMNeT++*, 2009) in its most recent incarnation, which includes a fairly accurate model of a CC2420 card and the 802.15.4 MAC. For the channel, we used a path loss model taking into account interferences from parallel transmissions to compute (at runtime) the SNR of each frame.

To put the SLIM results in context we compared it against a solution based on DYMO (Chakeres & Perkins, 2008), a well known protocol (the successor of AODV) for unicast routing in MANETs. Since SOP usually adopts unicast invocation (using a unicast transport on top of an IP network) and being interested in a solution for mobile scenarios, the choice of DYMO was the most natural one. This also accounts for the fact that the IETF, while proposing IPv6 for WSAN (Hui & Culler, 2008), has not yet chosen which protocol to use for routing but is considering unicast protocols only.

DYMO is an *on-demand* routing protocol. It creates routes between nodes only when they are required for communication. Routes are generated by flooding the network with *route request* packets that create entries in the routing tables of receiving nodes. Entries are temporary: they are trusted and used for a limited period of time and then expire; after expiration a new route creation process is needed for further communication. After routes are created they are followed by forwarding packets in unicast, hop-by-hop. Notice that to offer a fair comparison with SLIM, we adapted DYMO to WSAN and to the specific application service we are interested in, by reducing the number and length of fields in packet headers.

Reference scenario. We considered a reference scenario in which a single node behaves as a client, periodically sending service requests and invocations, while all other nodes act as service providers. We chose this simplified scenario after observing how the results in scenarios including several clients (not included here for space's sake) did not exhibit remarkable differences w.r.t. the scenarios involving a single client issuing a large number of invocations (which we discuss below).

In particular, our reference scenario includes 75 service providers moving (with the client) in an area of 0.25 Km^2 ($500m \times 500m$) at a maximum speed of 3 m/s (minimum is 0 m/s). Each service provider exposes a single service and client queries are matched by 10% of services on average. The exact behavior of the client is the following: it periodically sends a lookup request, chooses one service provider among all responding ones and sends 10 invocation messages to it, with an average delay among an invocation and the following one of 10 seconds.

Starting from this reference scenario, we analyzes the performance of SLIM under different conditions, changing, one by one, the main parameters of the scenario: the density of the network (number of nodes per Km^2), the area in which nodes are located, the maximum speed of nodes, the frequency of service invocation, and the number of credits used by the SLIM routing protocol.

Service invocation. Our first analysis focuses on service invocation, forgetting about the performance and cost of service location. In particular, we measured *delivery* as the percentage of service invocations actually receiving a reply from the service provider, and *traffic* as the average traffic (in KB/sec) generated to obtain this result. For SLIM, the latter includes all the traffic generated by `SVCINV` (either when sent in unicast or in flooding) and `REPLY` packets, while for DYMO it includes the traffic generated to build routes and to transport messages forth and back.

Actual results are shown in Figure 5: we run each simulation 20 times, varying the seeds of the random number generators used in our models and plotting the sample mean we measured and its 95% confidence interval. Generally speaking, we observe that our protocol delivers much more messages that DYMO, especially when using at least one credit, while generating significantly less traffic.

In particular, first row of graphs in Figure 5 shows how SLIM and DYMO behave while changing the density of nodes. We observe how both protocols present higher delivery at higher densities, which is reasonable since a lower density increases the probability of partitioning the network. At our reference density of 300 nodes per Km^2, the delivery of SLIM with one or two credits is very close to 100% while DYMO still misses some packets. As for traffic, we notice how the SLIM forwarding mechanism, which opportunistically privileges long links when available, results in using a more or less constant number of hops to reach the interested ser-

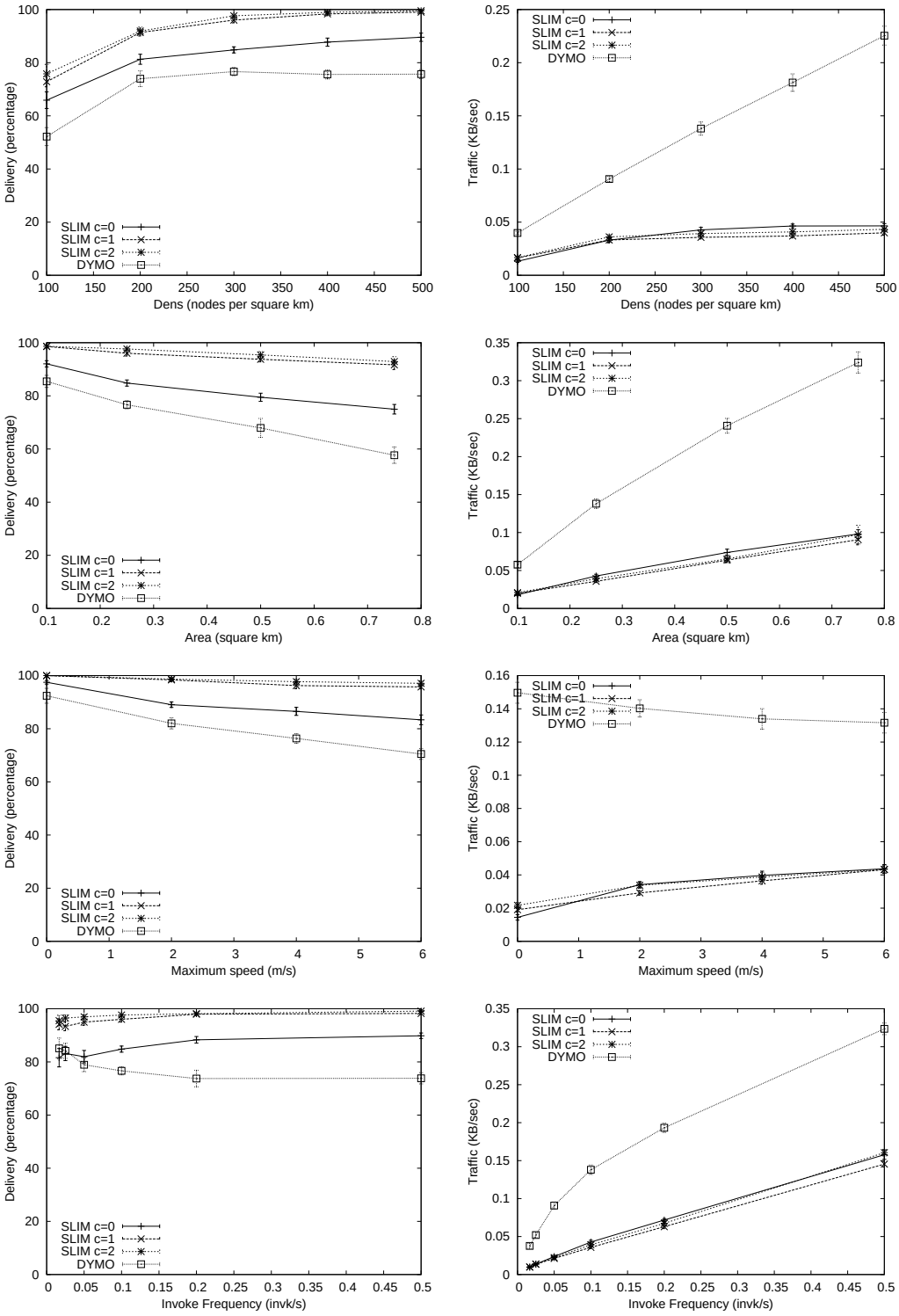

Figure 5: Performance analysis

vice provider, and hence generates the same traffic, independently from the density of the network. On the contrary, DYMO does not include any mechanism to filter hops based on the distance between nodes: as a consequence its traffic grows linearly with the number of nodes.

A similar behavior can be observed when increasing the area of simulation (with a fixed density), as shown by the second row of graphs. Both protocols decrease their overall delivery while the area grows, as packets have to travel a higher number of hops to reach their destination; but SLIM limits this trend due to its robust forwarding mechanism, which does not choose forwarders deterministically as in DYMO, while also adopting ad-hoc mechanisms like packet retransmission and renewal of routes through overhearing, which reduce packet loss and help keeping distance tables in sync even in presence of mobility. As for the traffic, also in this case SLIM behaves better than DYMO thanks to its opportunistic selection of retransmitting nodes during packet forwarding.

If we focus on the impact of credits on performance, we may observe that credits increase delivery with a minimal impact on traffic. Actually, increasing the number of credits from 0 to 1 reduces the overall traffic since the presence of credits increases the chance for a packet to reach its destination, thus reducing the need of reissuing the same packet again. Further increasing credits (e.g., from 1 to 2) slightly increases traffic as the retransmission mechanism may wrongly fire due to the presence of asymmetric links. In conclusion, we may observe that the ideal number of credits to use depends on the area of the network (see second row of graphs), the greater the area the more beneficial to delivery is increasing the number of credits, but in common scenarios like those we considered (a maximum area of 0.75 Km^2 with 225 service providers), two is the maximum number of credits worth using.

As our default scenario takes into account node mobility, we also studied the impact of the maximum speed of nodes (third row in Figure 5). Both protocols reduce delivery when maximum speed grows, as fast mobility contributes to a faster invalidation of routing information. On the other hand, DYMO suffers this problem much more than SLIM, as it chooses routes deterministically and it does not include any mechanism for retransmission or routing information renewal (apart rebuilding the route entirely). Notice that DYMO cannot reach a 100% delivery even in a fixed scenario with no mobile nodes. While this appeared strange to us at first, a more in-depth study has revealed that the packet loss is once again due to the route creation mechanism: in particular, DYMO does not include any mechanism to filter out long hops while flooding route requests to build the routing tables to be used for packet forwarding. As a result, especially in presence of asymmetric links (our channel model includes them) it may create routes that are hard to follow back. As for the traffic, SLIM uses more bandwidth when the speed increases, as packet loss causes retransmission of packets at each hop (using credits) or end-to-end (resending `SVCINV` packets in flooding if replies do not come back in time). On the contrary DYMO does not include any mechanism to deal with packet loss: as a consequence, once a packet is dropped it does not generate further traffic. This explains why we measured a moderate decrease in traffic as speed of nodes increases.

Finally, we measured the impact of the application behavior by varying the frequency of invocations from the reference value of 0.1 (1 every 10s). When the frequency is very low, routing information becomes invalid between two consecutive invocations and need to be updated. Both DYMO and SLIM use flooding in this phase, but our mechanism produces less traffic, minimizing the number of nodes that have to retransmit the packet, while keeping a similar delivery. At higher rates both protocols may use the same routing information for multiple invocations; however, a failure in the unicast transmission directly brings to packet loss when using DYMO while SLIM may opportunistically leverage different routes, while also including a retransmission mechanism when credits are greater than zero.

Service discovery. After measuring the performance of service invocation we were interested in measuring how service location behaves. The traditional approach to service location uses a single node acting as a *registry*, which stores information about all services exposed by service providers. On the contrary, SLIM uses a peer-to-peer approach gathering information by contacting every possible service provider to obtain service advertisements back. To compare these two approaches we sup-

posed the availability of a node acting as the registry and we compared the cost of our discovery mechanism with a single request to the registry sent through a unicast DYMO route. Notice that to further favor DYMO we did not consider the cost to populate the registry, which can be relevant in a dynamic scenario in which nodes may come and go while moving around.

SLIM c=0	SLIM c=1	SLIM c=2
33.27%	49.78%	56.16%

Table 1: Traffic for discovery (w.r.t. DYMO)

To concentrate on service discovery, in this case we built our simulations with the client periodically sending out service requests without performing any service invocation; Table 1 shows the overall traffic generated by SLIM w.r.t. DYMO. For space reasons we only provide results for the reference scenario, as all other ones reflect the trends already presented in Figure 5. Notice that, even if DYMO uses a unicast route to invoke the registry, it still has to undergo the route creation process: this results in an overall cost that is about twice SLIM's one, even when using credits.

SLIM c=0	SLIM c=1	SLIM c=2
78.39%	93.24%	94.19%

Table 2: Discovered services

Besides the costs, we were also interested in the effectiveness of the discovery phase. Accordingly, we measured the overall delivery for SLIM, i.e., the percentage of services the client receive information about w.r.t. to the total number of services matching the query. Table 2 shows the results obtained in the standard scenario: with zero credits less than 80% of matching services are discovered; however, only one credit is enough to bring this percentage to a very good value of 93.2%.

Multicast invocation. As already mentioned, the SLIM API includes a `mcastMsg` operation to contact all the services that satisfy a given query. Unfortunately, DYMO does not support multicast communication so we cannot use it for a direct comparison, but we can still make some considerations.

In particular, we notice that multicast service invocation in SLIM behaves exactly as service discovery, the only difference being a slight increase in the size of used packets (to account for the size of the message to be sent, as shown in Figure 4). With the payloads we used in our simulation, this results in multicast service invocation generating less than 5% greater more traffic than service discovery.

Starting from this number and remembering the results reported in Table 1, we can say that the cost of multicast invocation is still significantly lower than that of a single, unicast invocation in DYMO, as deduced by looking at the cost for DYMO to interact (in unicast) with the registry.

Along the same line, Table 2 demonstrates that a flooded request receives the vast majority of expected replies. The same result still holds when dealing with multicast invocation (we have done all the tests in the different scenarios, including several repliers for each request). More than 93% of expected replies are received using a single credit.

RELATED WORK

Different works have proposed the service oriented approach as a suitable abstraction for application developers to access the resources of WSANs. Most of them, however, consider an entire network of sensors as a single service provider and focus on the interaction between external (possibly remote) applications and a gateway server, which acts as a bridge between the sensor network and the outside, service oriented world. One of the first proposals in this direction is (Golatowski et al., 2003), which presents, at a very high level, a simple model in which an external server is adopted to communicate with the WSAN. A similar approach is described in (Avilés-López & García-Macías, 2009), which focuses on the abstraction layer offered to programmers. None of these works, however, provides details on how communication takes place between nodes.

As opposed to these approaches, SLIM does not introduce a full-fledged macro-programming style for a WSAN (Picco & Mottola, to appear); instead, it moves services directly into the WSAN, thus making the task of programming distributed applications simpler. Some works take the same approach.

In (Leguay, Lopez-Ramos, Jean-Marie, & Conan, 2008) authors propose a protocol stack, WSN-SOA

which reproduces the architectural concepts and information exchange of a regular SOA implementation. A topic based publish-subscribe communication paradigm is used to receive desired information from the network. Each service is registered as a topic on a gateway and applications choose which topics they are interested in. A unicast routing protocol is used to forward data from sensors to the gateway. Authors observe its lack of efficiency and scalability and promise further investigations on this aspect. The same communication paradigm is adopted in (Delicato, P. F. Pires, Pirmez, & Carmo, 2003), using Directed Diffusion (Intanagonwiwat, Govindan, & Estrin, 2000) to exchange data with the sensor nodes. OASiS (Kushwaha, Amundson, Koutsoukos, Neema, & Sztipanovits, 2007) proposes a programming paradigm in which developers define *logical objects* which are mapped to observed properties in the sensor network. Each sensor cooperates in the management of a logical object according to its capabilities, exposed as services. Authors propose requests flooding to locate services but don't provide implementation details. SYLPH (Tapia, Fraile, Rodríguez, Paz, & Bajo, 2009) proposes a layered architecture which can be adopted on top of heterogeneous devices. Gateways are used to connect devices that belong to different networks and use different communication protocols. None of the works above takes into account mobility, which instead is typical in various application scenarios for WSANs.

A Service Location Protocol for pervasive embedded devices, nanoSLP, has been proposed as part of the nanoIP protocol stack (Jardak et al., 2008). As SLIM, nanoSLP uses query flooding to locate matching services. The same approach is adopted by the Simple Service Discovery Protocol (SSDP), used in the UPnP protocol (Jeronimo & Weast, 2003). DEAPspace (Hermann et al., 2001) investigates completely decentralized discovery solutions, by continuously pushing information from node to node, so that all devices hold a list of all known services; as a consequence, discovery is performed locally. Similarly, Konark (Helal, Desai, Verma, & Lee, 2003) and PDP (Campo, Munoz, Perea, Marin, & Garcia-Rubio, 2005) use an hybrid push/pull communication style to exchange information about known services. Such information is cached in devices, so that a limited number of hops has to be travelled for discovery. In (Sailhan & Issarny, 2005), the authors propose a service discovery protocol aimed at large scale mobile ad-hoc networks, in which multiple directories are used to store known services. The nodes where directories have to be located are chosen dynamically at run-time, according to given parameters, involving resources of nodes and environmental variables. In (Bromberg & Issarny, 2005), the authors focused on the integration of heterogeneous discovery protocols, to provide interoperabiliy and flexibility.

The main benefit of SLIM w.r.t. these proposals is the full integration of service discovery and invocation with the routing protocol. Discovery, in fact, has the effect of populating the routing tables that are exploited to propagate service invocations and replies.

The SLIM protocol shares many of its core mechanisms with CCBR (Cugola & Migliavacca, 2009), but applies them to a radically different communication paradigm. Indeed, CCBR is a data-aware routing protocol that allows sinks to express their interest in data, which is subsequently pushed by sensors when they read it. This enables the traditional "publish-subscribe like" communication paradigm found in many protocols for WSNs. Conversely, SLIM adopts a pull style of interaction toward sensors, while also supporting commands to be sent to actuators. Moreover, some of the mechanisms that were originally developed for CCBR, have been modified in SLIM (e.g., by making larger use of RSSI) to benefit from the experience we gained in testing CCBR both in simulation and in real scenarios.

Finally, several protocols have been proposed so far to opportunistically find the next hop forwarder in a wireless network (Biswas & Morris, 2005; Zorzi & Rao, 2003; Jain & Das, 2008; Choudhury & Vaidya, 2004; Li, Sun, Ma, & Chen, 2008). They differ in the way this choice is made and how potential forwarders coordinate to limit multiple routes forwarding. With respect to these issues, the SLIM protocol adopts a very simple but efficient mechanism based on overhearing to let nodes coordinate, using a combination of hop distance from the destination (when available), link length and quality (through the RSSI), and remaining energy to elect the next forwarder. It also couples these mechanisms with a unique retransmission protocol to increase delivery while keeping overhead under control. Moreover, differently from the protocols

above, which use opportunistic forwarding to provide unicast multi-hop routing, SLIM uses its own opportunistic mechanism within a complex protocol, which jointly supports service location and invocation on a mobile WSAN.

Conclusions

In this paper we presented SLIM, a middleware to support service oriented programming (SOP) in mobile Wireless Sensor and Actuator Networks (WSANs). SLIM brings SOP directly inside the sensor network, allowing each node to act as a service provider, a service consumer, or both. While this holds the promise of simplifying the issue of programming large and complex WSANs, the kind of services each node may offer are those tailored to a sensing and actuating scenario, like closing a valve, getting the temperature and humidity sensed by a specific node, or collecting and averaging the values measured in an area. These services will seamlessly integrate with more complex, application-level services offered by PCs located in a standard network.

SLIM exposes a simple yet powerful API to applications, allowing service registration, discovery, and invocation. To better address the needs of the typical applications for WSANs, SLIM supports both unicast and multicast invocations. Moreover, the layered architecture of SLIM enables the usage of virtually every language for service descriptions and queries.

SLIM includes an advanced routing protocol explicitly designed for a mobile, multi-hop WSAN. The results we measured comparing this protocol with a more traditional one, DYMO, which defines and manages unicast routes between communicating nodes, shows that SLIM outperforms DYMO, by providing higher message delivery at a significantly lower cost.

Our plan for the future is to implement SLIM in TinyOS in order to apply it to real scenarios, like those we are considering in the WASP (`http://www.wasp-project.org/`) project.

Acknowledgments

This work was partially supported by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom; and by the Italian Government under the projects FIRB INSYEME and PRIN D-ASAP.

References

Akyildiz, I. F., & Kasimoglu, I. H. (2004). Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks*, *2*(4).

Andonovic, I., Michie, C., Gilroy, M., Goh, H., Kwong, K., Sasloglou, K., et al. (2009). *Wireless sensor networks for cattle health monitoring.* Springer.

Avilés-López, E., & García-Macías, J. A. (2009). Tinysoa: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, *3*(2), 99-108.

Baldoni, R., Beraldi, R., Cugola, G., Migliavacca, M., & Querzoni, L. (2005, Dec). Content-based routing in highly dynamic mobile ad hoc networks. *Int. Journal of Pervasive Computing and Communications*, *1*(4), 277–288.

Biswas, S., & Morris, R. (2005, Oct). Exor: opportunistic multi-hop routing for wireless networks. *ACM SIGCOMM Computer Communication Review*, *35*(4).

Bromberg, Y.-D., & Issarny, V. (2005). Indiss: interoperable discovery system for networked services. In *Middleware '05: Proceedings of the acm/ifip/usenix 2005 international conference on middleware* (pp. 164–183). New York, NY, USA: Springer-Verlag New York, Inc.

Campo, C., Munoz, M., Perea, J. C., Marin, A., & Garcia-Rubio, C. (2005). Pdp and gsdl: A new service discovery middleware to support spontaneous interactions in pervasive systems. In *Percomw '05: Proceedings of the third ieee international conference on pervasive computing and communications workshops* (pp. 178–182). Washington, DC, USA: IEEE Computer Society.

Chakeres, I. D., & Perkins, C. E. (2008, February). *Dynamic MANET On-demand Routing Pro-*

tocol. IETF Internet Draft, draft-ietf-manet-dymo-12.txt (Work in progress).

Choudhury, R., & Vaidya, N. (2004, Jan). Mac-layer anycasting in ad hoc networks. *ACM SIGCOMM Computer Communication Review*, *34*(1), 75–80.

Cugola, G., & Migliavacca, M. (2009). A context and content-based routing protocol for mobile sensor networks. In *Ewsn '09: Proceedings of the 6th european conference on wireless sensor networks* (pp. 69–85). Berlin, Heidelberg: Springer-Verlag.

Cugola, G., Murphy, A., & Picco, G. (2006). The handbook of mobile middleware. In (chap. Content-Based Publish-Subscribe in a Mobile Environment). CRC Press.

Delicato, F. C., P. F. Pires, F. P., Pirmez, L., & Carmo, L. F. (2003). A flexible web service based architecture for wireless sensor networks. In *Icdcsw '03: Proceedings of the 23rd international conference on distributed computing systems* (p. 730). Washington, DC, USA: IEEE Computer Society.

Evers, L., & Havinga, P. (2007, Oct). Supply chain management automation using wireless sensor networks. In *Proc. ieee int. conf. on mobile adhoc and sensor systems* (pp. 1–3). Pisa, Italy: IEEE.

Golatowski, F., Blumenthal, J., H, M., Haase, M., Burchardt, H., & Timmermann, D. (2003). Service-Oriented Software Architecture for Sensor Networks. In *In proc. int. workshop on mobile computing (imc03* (pp. 93–98).

Helal, S., Desai, N., Verma, V., & Lee, C. (2003, march). Konark - a service discovery and delivery protocol for ad-hoc networks. In (Vol. 3, p. 2107 -2113 vol.3).

Hermann, R., Husemann, D., Moser, M., Nidd, M., Rohner, C., & Schade, A. (2001). Deapspace: transient ad hoc networking of prevasive devices. *Comput. Netw.*, *35*(4), 411–428.

Hui, J. W., & Culler, D. E. (2008, July-August). Extending IP to Low-Power, Wireless Personal Area Networks. *Internet Computing, IEEE*, *12*(4), 37-45.

Intanagonwiwat, C., Govindan, R., & Estrin, D. (2000). Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobicom '00: Proceedings of the 6th annual international conference on mobile computing and networking* (pp. 56–67). New York, NY, USA: ACM.

Jain, S., & Das, S. (2008). Exploiting path diversity in the link layer in wireless ad hoc networks. *Ad Hoc Networks*, *6*(5), 805–825.

Jardak, C., Meshkova, E., Riihijrvi, J., Rerkrai, K., & Mahonen, P. (2008). Implementation and performance evaluation of nanoip protocols: Simplified versions of tcp, udp, http and slp for wireless sensor networks. In *Wcnc 2008, ieee wireless communications & networking conference, march 31 2008 - april 3 2008, las vegas, nevada, usa, conference proceedings* (p. 2474-2479). IEEE.

Jeronimo, M., & Weast, J. (2003). *Upnp design by example: A software developer's guide to universal plug and play.* Intel Press.

Kushwaha, M., Amundson, I., Koutsoukos, X., Neema, S., & Sztipanovits, J. (2007, Jan.). Oasis: A programming framework for service-oriented sensor networks. In *Communication systems software and middleware, 2007. comsware 2007. 2nd international conference on* (p. 1-8).

Leguay, J., Lopez-Ramos, M., Jean-Marie, K., & Conan, V. (2008, October). An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks. In *33rd ieee conference on local computer networks, 2008. lcn 2008.* (p. 740-747).

Li, L., Sun, L., Ma, J., & Chen, C. (2008, Jun). A receiver-based opportunistic forwarding protocol for mobile sensor networks. In ACM (Ed.), *The 28th international conference on distributed computing systems workshops.*

Mobility Framework for OMNeT++. (2009). mobility-fw.sourceforge.net.

Picco, G., & Mottola, L. (to appear). Programming wireless sensor networks: Fundamental concepts and state-of-the-art. *Comp. Surv.*.

Proc 3rd int. conf. on pervasive computing technologies for healthcare. (2009, April). London, UK: IEEE.

Sailhan, F., & Issarny, V. (2005). Scalable service discovery for manet. In *Percom '05: Proceedings of the third ieee international conference on pervasive computing and communications* (pp. 235–244). Washington, DC, USA: IEEE Computer Society.

Sikka, P., Corke, P., Valencia, P., Crossman, C.,

Swain, D., & Bishop-Hurley, G. (2006). Wireless ad-hoc sensor and actuator networks on the farm. In *Proc. of the 5th int. conf. on information processing in sensor networks* (pp. 492–499). New York, NY, USA: ACM.

Sohraby, K., Minoli, D., & Znati, T. (2007). *Wireless sensor networks: Technology, protocols, and applications*. J. Wiley.

Tapia, D. I., Fraile, J. A., Rodríguez, S., Paz, J. F. de, & Bajo, J. (2009). Wireless sensor networks in home care. In *Iwann '09: Proceedings of the 10th international work-conference on artificial neural networks* (pp. 1106–1112). Berlin, Heidelberg: Springer-Verlag.

Varga, A. (2001, June). The OMNeT++ Discrete Event Simulation System. *Proceedings of the European Simulation Multiconference (ESM'2001)*.

Zorzi, M., & Rao, R. (2003). Geographic random forwarding (geraf) for ad hoc and sensor networks: multihop performance. *IEEE Trans. on Mobile Computing*, *2*(4), 337–348.