# Hitch Hiker: A Remote Binding Model with Priority Based Data Aggregation for Wireless Sensor Networks

Gowri Sankar Ramachandran,
Wilfried Daniels, José Proença,
Sam Michiels, Wouter Joosen, and
Danny Hughes
iMinds-DistriNet, KU Leuven,
3001 Leuven, Belgium.
gowrir@cs.kuleuven.be

Barry Porter
School of Computing and Communications
Lancaster University,
Bailrigg,
Lancaster LA1 4YW,
United Kingdom.
b.f.porter@lancaster.ac.uk

## ABSTRACT

The aggregation of network traffic has been shown to enhance the performance of wireless sensor networks. By reducing the number of packets that are transmitted, energy consumption, collisions and congestion are minimised. However, current data aggregation schemes restrict developers to a specific network structure or cannot handle multi-hop data aggregation. In this paper, we propose *Hitch Hiker*, a remote component binding model that provides support for multi-hop data aggregation. Hitch Hiker uses component meta-data to discover remote component bindings and to construct a multi-hop overlay network within the free payload space of existing traffic flows. This overlay network provides end-to-end routing of low-priority traffic while using only a small fraction of the energy of standard communication. We have developed a prototype implementation of Hitch Hiker for the LooCI component model. Our evaluation shows that Hitch Hiker consumes minimal resources and that using Hitch Hiker to deliver low-priority traffic reduces energy consumption by up to 15%.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

## Keywords

Wireless Sensor Networks; Components; Aggregation

## 1. INTRODUCTION

Wireless Sensor Networks (WSN) must operate for long periods on limited power supplies and research has shown that wireless communication is the primary source of energy consumption in WSN [15]. It is therefore critical to minimise radio transmissions. *Data aggregation* has been widely applied to tackle this problem [8, 9]. By combining multiple messages addressed to a common destination into a single datagram, transmissions are reduced and energy is conserved. Furthermore, less frequent transmissions result in fewer collisions and therefore retransmissions. This can significantly lower WSN power consumption.

This paper focuses on data aggregation, which is achieved through the efficient merging of application traffic flows, rather than algebraic in-network aggregation [12]. Contemporary approaches to lossless data aggregation may be classified as either *application dependent* or *application independent* [5]. Application dependent approaches [6,8,12] support the creation of optimal network-wide data aggregation structures, but restrict the topology of the distributed application. In contrast, application independent approaches [1, 8] embed generic aggregation functionality in the underlying network stack, but do not consider application requirements, and therefore do not achieve optimal performance.

A new approach is needed that allows developers to build custom application communication structures, while providing support for efficient data aggregation. To tackle this problem, this paper introduces the *Hitch Hiker component binding model*, with support for *multi-hop data aggregation* based on priority information associated with bindings.

Hitch Hiker extends binding models to distinguish between *high-* and *low-priority* bindings. The Hitch Hiker binding model allows developers to specify *high-priority* remote bindings that generate radio transmissions, or *low-priority* remote bindings which communicate exclusively using the data aggregation overlay and therefore result in no additional transmissions. Using component meta-data, Hitch Hiker constructs a multi-hop Hitch Hiker overlay network from the free payload space of high-priority bindings. Low-priority bindings use the Hitch Hiker overlay network, and therefore avoids additional radio transmissions between remote components. By routing low-priority traffic over this data aggregation overlay, Hitch Hiker significantly reduces energy consumption. Furthermore, low-priority bindings provide developers with a low cost, lightweight and easy to use approach for data aggregation. To the best of our knowledge, Hitch Hiker is the first component binding model that provides support for data aggregation.

A prototype of Hitch Hiker has been implemented for the LooCI component model [7] running on the Contiki OS [4] and for the OMNeT++ [17] simulator. Our evaluation shows that: (i.) the resource consumption of Hitch Hiker is minimal and (ii.) by using Hitch Hiker to transmit low-priority traffic, energy consumption is significantly reduced.


*CBSE'15,* May 4–8, 2015, Montréal, QC, Canada.

http://dx.doi.org/10.1145/2737166.2737179.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 introduces the Hitch Hiker binding model. Section 4 introduces and evaluates prototype implementations of Hitch Hiker. Finally, Section 5 concludes this paper.

# 2. RELATED WORK

Section 2.1 discusses work in the area of data aggregation. Section 2.2 discusses component and binding models.

## 2.1 Data Aggregation Schemes

He et al. [5] describe two classes of data aggregation approach: Application Dependent Data Aggregation (ADDA), which requires knowledge of application-level traffic flows and Application Independent Data Aggregation (AIDA) which performs aggregation in a generic fashion without application-specific information. We discuss both classes of aggregation in Section 2.1.1 and Section 2.1.2, respectively.

### 2.1.1 Application Dependent Data Aggregation

ADDA approaches use network-wide application information to optimise the manner in which information is collected and routed across the network. These efforts focus upon the network and application layers of the communication stack.

At the *Network Layer*, Intanagonwiwat et al. introduce Directed Diffusion [8], which provides data-centric routing, in-network caching and aggregation. At the *Application Layer*, Madden et al. contribute the Tiny AGgregation (TAG) [12] service, which allows users to specify SQL-like queries, which are multicast to relevant sensor nodes using a tree that is rooted at the base-station. Network-flow based data aggregation protocols [9, 19] take an orthogonal approach, modelling the sensor network as a graph and, based upon application-level traffic flows, calculating and configuring an optimal aggregation structure. Network-flow based approaches offer efficient calculation of an optimal data aggregation structure, for static networks where network-wide data flows are known, but these approaches are unsuitable for dynamic networks which support runtime reconfiguration. It can be seen that contemporary approaches are either inherently static as in network flow models [8], or otherwise restrict developers to a single application interaction model [12] or routing topology [9, 19]. In contrast, application-independent approaches provide a more *generic* aggregation approach, discussed below in Section 2.1.2.

### 2.1.2 Application Independent Data Aggregation

AIDA schemes provide a one size fits all approach to data aggregation that is independent of application requirements. These approaches typically operate at the network and data link layer. At the *Network Layer*, well known approaches to aggregation include the Shortest Path Tree (SPT), wherein a single, network-wide aggregation tree is centrally calculated and configured and the Greedy Incremental Tree (GIT) which approximates a shortest path tree, but is constructed in an incremental and decentralised fashion [10]. However, such approaches are poorly suited to WSN scenarios where energy resources are unevenly distributed. At the *Data Link Layer*, He et al. contribute AIDA [5], which takes advantage of queuing delay and the broadcast nature of wireless media to implement application independent data aggregation. AIDA aggregates multiple packets into single frames prior to transmission, resulting in significant savings in terms of energy and latency. While AIDA uses data from the network layer, it treats the application layer as a black box and therefore cannot exploit patterns in application traffic flows. Furthermore, as AIDA operates at the data link layer, it is unable to perform multi-hop data aggregation.

## 2.2 Remote Component Binding Models

Hitch Hiker combines aggregation with a lightweight remote binding model. In this section, we review component binding models and discuss opportunities for aggregation. Contemporary remote binding models typically offer either event-based or Remote Procedure Call (RPC) semantics. *RPC-based binding models* allow remote functionality to be called using the same semantics as local procedures, thus lowering the overhead on component developers. RPC-based models are request-reply and therefore bidirectional in nature. In a WSN context, May et al. [13] contribute RPC calls with support for unicast and anycast, wherein exactly one neighbouring node responds to the call. Where component models support remote reconfiguration, bindings may be modified at runtime.

*Event-based binding models* provide simple unidirectional communication between software modules. Event-based approaches are attractive in resource-constrained scenarios, as they are lightweight and do not cause software modules to block while waiting for responses as in RPC. The Active Messages [18] protocol provides remote bindings for the NesC component model. A unique reference to an application handler is embedded in each active message and is used to dispatch incoming messages to the appropriate handler component. The LooCI binding model [7] provides unreliable event-based binding using a decentralised publish-subscribe *event bus* communication medium. LooCI bindings may be remotely modified at runtime in order to enact reconfiguration.

Considering opportunities for cross-layer optimisation, all of the binding models discussed above [7,18] provide explicit meta-data that can be used to determine traffic flows and therefore optimise aggregation functionality. Despite this opportunity, current component models typically treat the network layer and below as a black box, resulting in suboptimal communication. In the following section, we describe the Hitch Hiker binding model.

# 3. THE HITCH HIKER BINDING MODEL

This section describes the design of the Hitch Hiker component binding model and its associated network stack. Section 3.1 introduces prioritised bindings. Section 3.2 describes how route information is extracted from bindings. Section 3.3 describes the Hitch Hiker stack.

## 3.1 Prioritised Bindings

Figure 1 shows a part of the *smart building* case study, used in our evaluation. Here, a temperature component, deployed on sensor node $N_1$, samples temperature data once every 30s and sends the data to the comfort level component. The comfort level component on $N_2$, analyses sensor data, and sends the result to a manager located on $N_3$ every 30s. These three components communicate via standard bindings, depicted as  . For the remainder of this paper, we refer the standard bindings as *high-priority bindings*.

Hitch Hiker then introduces the concept of *low-priority bindings*, depicted as  in Figure 1. Low-priority bind-

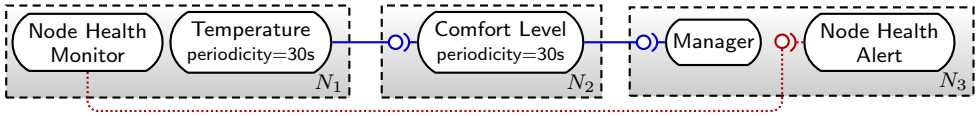


**Figure 1:** Application view of a deployed application.

ings are used by non-critical applications, the definition of which is left to the developer. The use of a low-priority binding indicates that the developer is willing to trade communication performance for energy efficiency. Low-priority bindings are realised in Hitch Hiker by routing messages via the data aggregation overlay network, referred to as *Hitch Hiker overlay network*. In our example, a node health monitor component deployed on $N_1$ is connected to a node health alert component deployed on $N_3$ via a low-priority binding.

**High-Priority Bindings** Hitch Hiker extends the LooCI binding model described in [7]. Conceptually, a LooCI binding is a connection between a source and a destination component, with an associated event type and a reference to the network *link* (which is potentially a multi-hop route) that connects the nodes hosting the two components.

Definition 1 (Binding). *A* binding *is a tuple* $b = \langle C_s, C_d, Type, Link\rangle$*, where* $C_s$ *is the source component,* $C_d$ *is the destination component, Type is the type of the events sent through the binding, and Link is the remote connection between the nodes where the binding is deployed, defined below.*

Definition 2 (Remote Connection). *A* remote connection *is a tuple* $\ell = \langle N_s, N_d, MTU, Bw, D\rangle$ *describing the communication channel between two network nodes, where* $N_s$ *is the source node,* $N_d$ *is the destination node, MTU is the* maximum transmission unit *between* $N_s$ *and* $N_d$*, Bw is the* bandwidth *of the remote connection, and D is the* expected delay *of the remote connection.*

A LooCI binding is realised as an *outgoing binding* entry on the sending node and an *incoming binding* entry on the receiving node, which is established by issuing `bindTo` and `bindFrom` calls to the sender and receiver, respectively. These bindings are created at runtime and stored in a *binding table* that is used to dispatch events. High-priority component binding's communication is mediated by the transmission of events using the network stack of the host operating system.

**Low-Priority Bindings** Hitch Hiker introduces the concept of a *low-priority* binding, depicted as ···○··· in Figure 1. In our example, a *node health monitor* component gathers the local node status such as battery level and radio link quality, and transmits this data to a *node health alert* component running on a server. We selected node health monitoring as an example of a low-priority application because this functionality is typically less important than the core WSN mission of gathering environmental data. However, it should be noted that developers are free to define which components are high-priority and low-priority in their application context. In our example (Figure 1), the low-priority binding connecting the *node health monitor* to the *node health alert* component is formally represented as $\langle$Node Health Monitor, Node Health Alert, `Status`, $\ell_{status}\rangle$, where `Status` is an event type with node status information and $\ell_{status}$ is the remote connection of the overlay network.

High-priority and low-priority bindings have the same artefacts: a source component, destination component, event type (Definition 1) and a remote connection (Definition 2). Low-priority bindings are realised in LooCI by adding a separate set of *binding tables* to each node.

The overlay routes necessary to support low-priority bindings are established reactively, as it required to support low-priority bindings. As with all data aggregation approaches, low-priority bindings can only be established where sufficient high-priority traffic exists to support them.

## 3.2 Component Probe Extracts Network Data

The component *probe* extracts data from the high-priority application to create the remote connections of the Hitch Hiker network. It *intercepts binding acknowledgment* messages containing the source component $C_s$, the source node $N_s$, the destination node $N_d$, and the binding event type *Type*, and builds a remote connection for the Hitch Hiker network. Recall that a remote connection is formally a tuple $\langle N_s, N_d, MTU, Bw, D\rangle$ (Definition 2). The *MTU* is calculated based on the event type, which has an associated payload size. Hitch Hiker extracts periodicity information by querying source components for their *periodicity* property using the standard LooCI API. Hitch Hiker distinguishes between *periodic* and *non-periodic* components: the former send values at a fixed rate (e.g., a temperature reading every 10 s), and the latter exhibit unpredictable behaviour (e.g., an alert generated when a window is opened).

Formally, we write $\Pi(C)$ to denote the periodicity of a component $C$, defined below, which returns the special symbol $\perp$ when $C$ is non-periodic.

$$\Pi(C) = \begin{cases} r & \text{if } C \text{ is periodic with rate } r; \\ \perp & \text{otherwise.} \end{cases}$$

Based upon the information intercepted in the binding acknowledgment—the source component $C_s$, the source node $N_s$, the destination node $N_d$, and the event type *Type*—and the periodicity $\Pi(C_s)$, the probe calculates the remote connection for the Hitch Hiker network as follows.

1. Get the payload size *ps* associated with the *Type*.
2. Get the *MTU* $m$ of the remote connection between the source ($N_s$) and destination ($N_d$).
3. Define $hd$ to be the size of the headers used by the data-link, network and transport layers of the host protocol stack.
4. Define $MTU_{HH}$ to be the unused payload size, calculated as $m - ps - hd$.
5. If $\Pi(C_s) = \perp$ then return the remote connection $\langle N_s, N_d, MTU_{HH}, \perp, \perp\rangle$, otherwise return the remote connection $\langle N_s\,, N_d\,, MTU_{HH}\,, MTU_{HH}/\Pi(C_s)\,, \Pi(C_s)\rangle$.

## 3.3 Hitch Hiker Network Stack

Figure 2 shows the Hitch Hiker network stack for a single embedded sensor node. The Hitch Hiker protocol stack is

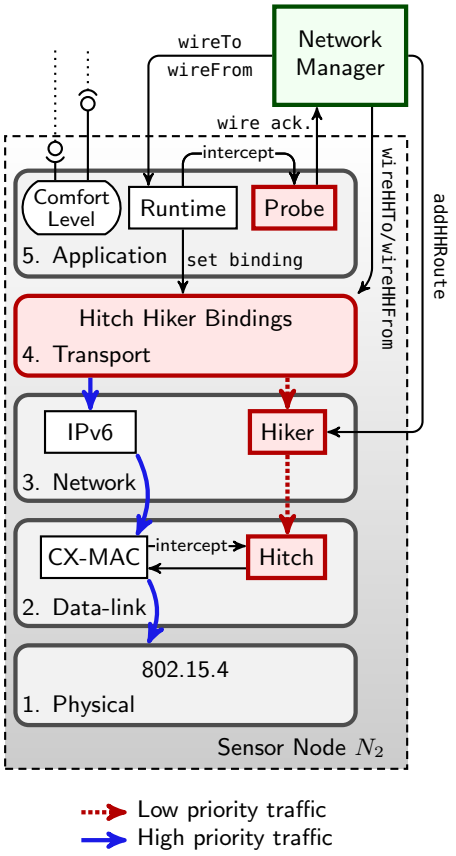


**Figure 2:** Architecture view over the node $N_2$.

composed of two protocols, the *Hitch* Virtual Medium Access Control (MAC) protocol (Section 3.3.1) and the *Hiker* routing protocol (Section 3.3.2).

### 3.3.1 Hitch Virtual MAC Protocol

Hitch is a virtual MAC protocol that manages and provides access to Hitch Hiker overlay links.

**Link Data Structures:** Hitch manages the set of virtual data links that are available on each sensor node. Each virtual data link maps to a remote overlay connection (Definition 2) that is composed of: a destination address, MTU, delay and bandwidth. Virtual links are created by the *probe*. A First In First Out (FIFO) queue is maintained per link where packets are buffered until they can be aggregated with high-priority traffic and dispatched. If the buffer reaches its capacity, the oldest frame in the queue is discarded, resulting in packet loss. Hitch is a best-effort protocol, which provides no reliability guarantees. Where reliability is required, it should be implemented by the transport layer.

**Aggregation:** The Hitch protocol intercepts outgoing packets as they are passed to the host MAC protocol. If the virtual link queue associated with the destination of an intercepted packet is not empty, the available payload size is filled with packets from the queue, until either the available payload space is exhausted or the buffer is empty. The modified packet is then returned to the host MAC protocol to be transmitted.

**Disaggregation:** The Hitch protocol intercepts incoming frames in the host MAC protocol, and disaggregates all encapsulated Hitch packets. The disaggregated packets are then passed to the network layer.

### 3.3.2 Hiker Overlay Network Protocol

Hiker is a multi-hop overlay routing protocol that operates efficiently with the Hitch data link protocol.

**Route Data Structures:** Hiker maintains a minimalist *routing table* on each node. This routing table begins empty, and routes are reactively configured by the network manager to support low-priority bindings. Each route is comprised of a *remote destination*, the *virtual link* that represents the next hop on the route to this address and a *route-MTU* which denotes the maximum packet size that can traverse the complete route.

**Routing:** When an incoming Hiker packet is received, the destination field of the packet is checked. If the destination is the local sensor node, it is passed to the transport layer. If the destination matches a known route, it is transmitted on the appropriate link using the `transmit(frame,link)` method of Hitch. Otherwise, the packet is discarded.

DEFINITION 3 (ROUTE). *A* route *is a multi-hop remote connection (Definition 2) obtained by composing a non-empty sequence of remote connections, such that for every consecutive $\ell$ and $\ell'$ the destination node of $\ell$ matches the source node of $\ell'$. Given a sequence of $n$ remote connections:* $\langle N_{s,1}, N_{d,1}, MTU_1, Bw_1, D_1\rangle, \ldots, \langle N_{s,n}, N_{d,n}, MTU_n, Bw_n, D_n\rangle$ *its composition yields the route* $\langle N_s, N_d, MTU, Bw, D\rangle$, *where*

$$N_s = N_{s,1} \qquad MTU = \texttt{min}_{i=1}^{n}\, MTU_i \qquad D = \textstyle\sum_{i=1}^{n} D_i.$$
$$N_d = N_{d,n} \qquad Bw = \texttt{min}_{i=1}^{n}\, Bw_i$$

**Route Discovery** *Hitch Hiker* provides efficient centralised route creation. Hiker assumes that a single LooCI *network manager* is running for the entire network. This network manager enacts all management and reconfiguration. This information is exploited to create the Hitch Hiker overlay network as follows:

1. Overlay links are discovered based upon extended binding acknowledgements. This information is provided by the component probe as described in Section 3.2.
2. The network manager assembles discovered overlay links to form a network graph, wherein each link is labelled with its associated delay, MTU and bandwidth.
3. When the user requests the establishment of a low-priority binding $b$:
   (a) The graph is pruned to remove all links which have an insufficient MTU to support the specified data type.
   (b) The Dijkstra algorithm is used to calculate the *shortest path* between the source and destination, using either delay or bandwidth as the link cost. Our evaluation uses delay as the link cost.
   (c) The network manager configures the *shortest path* overlay route, or responds with an exception where no overlay route is possible.
   (d) Finally, the network manager configures the route required by the low-priority binding $b$, by sending route-creation messages to all involved nodes.

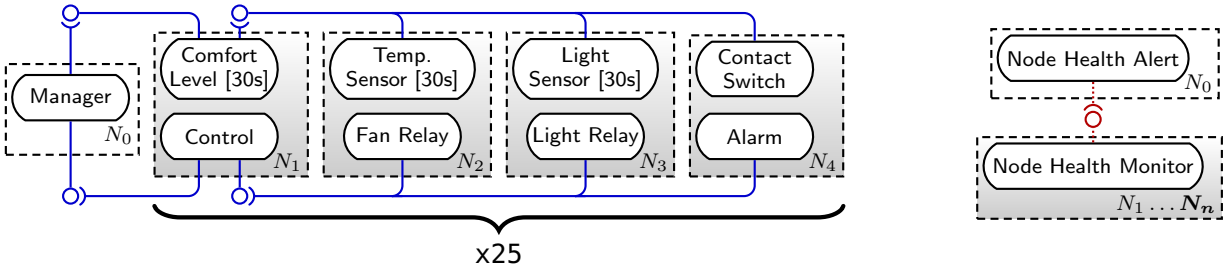


**Figure 3:** Component bindings of the smart building application (left) and for the monitoring application (right).

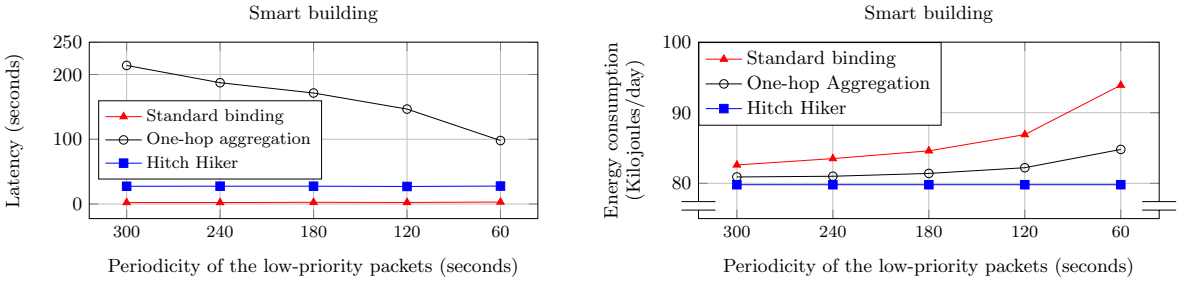


**Figure 4:** Latency (left) and the energy consumption (right) of the node health monitoring application overlaid on the smart building with Hitch Hiker, one hop aggregation and standard bindings.

# 4. IMPLEMENTATION AND EVALUATION

We have developed prototypes of Hitch Hiker for the OMNeT++ simulator [17] and the Zigduino mote [11].

*OMNeT++ settings:* The physical layer is a CC2420 IEEE 802.15.4 radio [16]. We use B-MAC [14] as a representative Low Power Listening (LPL) protocol.

*Zigduino configuration:* Zigduino is an Arduino-compatible mote based on the ATmega128RFA1 [2], which offers a 16 MHz MCU, 16 KB of RAM, 128 KB of Flash and an IEEE 802.15.4 radio. We use ContikiOS v2.6, Contiki X-MAC (CX-MAC) [3] and LooCI v2.0 [7] extended with Hitch Hiker. The parameterisation of CX-MAC uses the default Contiki values.

We compare Hitch Hiker against (i.) transmission of standard messages (referred to as *Standard* binding) and (ii.) an optimally configured one-hop data aggregation scheme using an optimal aggregation buffer size of 3 (referred to as *One-hop aggregation*). Values reported below represent averages taken over one week.

## 4.1 Case Study Applications

**Smart Building Application** The *smart building* application reduces energy consumption by sensing environmental conditions and controlling relevant appliances (left part of Figure 3). Sensor nodes ($N_2$, $N_3$ and $N_4$) monitor: temperature, light and if a window is open or closed every 30s. Sensor data is transmitted to a comfort level calculator running on the cluster-head sensor node ($N_1$) and forwarded to a manager running on a server ($N_0$), every 30s. This manager issues commands to a control component on the cluster-heads ($N_1$), which activates or deactivates relay switches on nodes $N_2$ to $N_4$ to control: lighting, ventilation and a window alarm. This application is realised using high-priority bindings. The payload size of sensor data is 4 B, the payload size of comfort data is 12 B and the size of relay commands is 4 B. In terms of network topology, the scenario covers 25 offices. Each office contains three sensor nodes and a cluster-head node. Sensor nodes communicate with cluster heads, which in turn communicate with a single server for management of comfort level. This approximates a 101-node tree, rooted at the server.

**Node Health Monitoring Application** The *node health monitoring* application (right part of Figure 3) is a low-priority application. This component monitors battery level, memory use and the radio link quality. The application consists of a health monitor component that runs on all sensor nodes ($N_1$ to $N_n$) and sends node health information to an alert component running on the server node ($N_0$). Node health monitoring is overlaid on the smart building application using low-priority bindings. The payload size of node health monitor data is 18 B.

## 4.2 OMNET++ Simulation Results

Figure 4 shows the results of our simulation. The sampling frequency of the node health monitoring application was set to 10% to 50% of the base application frequency. The $y$-axis shows the power consumption of low-priority Hitch Hiker bindings, standard bindings and one-hop data aggregation.

**Latency** As expected, the node health monitoring application exhibited a higher latency when using low-priority bindings than with standard bindings due to packets waiting for aggregation at each hop. However, the latency of low-priority bindings is lower than the one-hop aggregation scheme due to the exploitation of multi-hop routes.

**Energy** The results shown in Figure 4 confirm the expected savings when using Hitch Hiker to route low-priority traffic. Energy consumption is reduced by up to 15% in the smart building scenario. The energy consumption of Hitch Hiker is also lower than that of one-hop data aggregation.

## 4.3 Zigduino/Contiki Implementation Results

This section reports the performance timings of route configuration and message transmission as well as energy con-

sumption and memory overhead for the Contiki/Zigduino implementation.

**Route Creation:** Hitch Hiker requires approximately 86ms to configure a single low-priority binding. Each additional hop that must be configured adds 30 ms to the configuration overhead. Route configuration is thus lightweight; creating all of the Hitch Hiker bindings required for the smart building takes less than 3 s. However, this generates four transmissions per Hitch Hiker binding, which costs 36.5 mJ.

**Message Transmission:** Enqueueing, dequeueing and encapsulating a single Hitch Hiker packet within a host frame requires on average of 12.27 mJ, while a standard frame transmission using CX-MAC requires 21.41 mJ, a saving of 57.4% compared to standard transmission.

**Memory:** The implementation of Hitch Hiker adds 3% of ROM and 8% of RAM to the LooCI component model. Each routing table entry uses an additional 6 B of memory. We believe that this low overhead is reasonable in light of the energy savings reported in Section 4.2.

# 5. CONCLUSIONS

This paper introduced *Hitch Hiker*, a novel remote binding model for WSN which supports prioritised bindings and multi-hop data aggregation. This model provides developers with a low-effort mechanism to manage data aggregation. To the best of our knowledge, Hitch Hiker is the first component binding model to embed data aggregation support.

Simulation shows that using Hitch Hiker to route low-priority traffic consumes less energy (up to 15%) than explicit transmissions or one-hop data aggregation. Our prototype implementation for the Zigduino mote [11] validates that Hitch Hiker consumes minimal memory and that aggregation costs a small fraction of the energy that is required for a standard radio transmission.

As a future work, we will extend Hitch Hiker to support decentralised route discovery and provide Quality-of-Service guarantees for Hitch Hiker bindings.

# 6. ACKNOWLEDGEMENTS

This research is partially supported by the Portuguese FCT grant BPD/91908/2012, and by the Research Fund, KU Leuven, the IWT and iMinds (a research institute founded by the Flemish government). The research is conducted in the context of the following projects: ICON-COMACOD, IOF-TRANSITION and GOA-ADDIS.

# 7. REFERENCES

[1] T. Aonishi, T. Matsuda, S. Mikami, H. Kawaguchi, C. Ohta, and M. Yoshimoto. Impact of aggregation efficiency on git routing for wireless sensor networks. In *Int. Conf. on Parallel Processing Workshops*, pages 8 pp.–158, Aug. 2006.

[2] Atmel Corporation. *ATmega128RFA1 datasheet*, 2012.

[3] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *4th Int. Conf. on Embedded Networked Sensor Systems*, pages 307–320, 2006.

[4] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE Int. Conf. on Local Computer Networks*, pages 455–462, Nov 2004.

[5] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Aida: Adaptive application-independent data aggregation in wireless sensor networks. *ACM Transactions on Embedded Computing Systems*, pages 426–457, May 2004.

[6] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *33rd Annual Hawaii Int. Conf. on System Sciences*, page 10 pp., Jan 2000.

[7] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, J. Del Cid, W. Horre, C. Huygens, S. Michiels, and W. Joosen. Looci: The loosely-coupled component infrastructure. In *IEEE Symposium on Network Computing and Applications*, pages 236–243, 2012.

[8] C. Intanagonwiwat, R. Govindan, D. Estfin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11:2–16, 2003.

[9] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. *Computer Networks*, 42(6):697 – 716, 2003.

[10] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *22nd Int. Conf. on Distributed Computing Systems Workshops*, pages 575–578, 2002.

[11] Logos Electromechanical. *Zigduino Manual*, 4 2014. Rev. 2.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, pages 131–146, 2002.

[13] T. D. May, S. H. Dunning, G. A. Dowding, and J. O. Hallstrom. An RPC design for wireless sensor networks. *Int. Journal of Pervasive Computing and Communications*, 2(4):384–397, 2007.

[14] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *2nd Int. Conf. on Embedded Networked Sensor Systems*, SenSys '04, pages 95–107. ACM, 2004.

[15] V. Raghunathan, C. Schurgers, S. Park, M. Srivastava, and B. Shaw. Energy-aware wireless microsensor networks. In *IEEE Signal Processing Magazine*, pages 40–50, 2002.

[16] Texas Instruments. *CC2420 datasheet*, 2014.

[17] A. Varga. OMNeT++. In K. Wehrle, M. Günes, and J. Gross, editors, *Modeling and Tools for Network Simulation*, pages 35–59. Springer Berlin Heidelberg, 2010.

[18] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: A mechanism for integrated communication and computation. In *19th Annual Int. Symposium on Computer Architecture*, pages 256–266, 1992.

[19] Y. Xue, Y. Cui, and K. Nahrstedt. Maximizing lifetime for data aggregation in wireless sensor networks. *Mob. Netw. Appl.*, 10(6):853–864, Dec. 2005.