An Easy to Use Bluetooth Scatternet Protocol for fast Data Exchange in Wireless Sensor Networks and Autonomous Robots

Rico Möckel, Alexander Spröwitz, Jérôme Maye, Auke Jan Ijspeert

***Abstract*— We present a Bluetooth scatternet protocol (SNP) that provides the user with a serial link to all connected members in a transparent wireless Bluetooth network. By using only local decision making we can reduce the overhead of our scatternet protocol dramatically. We show how our SNP software layer simplifies a variety of tasks like the synchronization of central pattern generator controllers for actuators, collecting sensory data and building modular robot structures. The whole Bluetooth software stack including our new scatternet layer is implemented on a single Bluetooth and memory chip. To verify and characterize the SNP we provide data from experiments using real hardware instead of software simulation. This gives a realistic overview of the scatternet performance showing higher order effects that are difficult to be simulated correctly and guaranties the correct function of the SNP in real world applications.**

I. INTRODUCTION

During the past a couple of wireless standards were developed each including its own strengths and weaknesses like WLAN, Infrared and Zigbee. However we believe that Bluetooth [1] is the best choice when developing wireless sensor networks that need a good tradeoff between power consumption and data rate as well as for autonomous robots that need a flexible and powerful communication infrastructure while being powered from a limited source like a battery. We believe this for the following reasons: (1) Implementations of the latest Bluetooth standard 2.0 reported power consumptions of more than 10 times less than for WLAN. That is why Bluetooth is very interesting for embedded systems that are battery powered and there for have limited energy resources. (2) Latest Bluetooth implementations report communications speeds of up to 3Mbit/s. Already the former Bluetooth standard 1.2 supports data rates of up to 600kbit/s and could provide wireless serial links with a baud rate of about 115200. That is why Bluetooth is much more applicable to robotics than e.g. Zigbee which currently only supports data rates of 20-250 kbit/s. (3) Bluetooth is using a very robust communication protocol called Frequency Hopping. This allows Bluetooth networks to operate reliably in noisy environments and in parallel to other wireless communication networks. (4) In contrast with infrared, Bluetooth devices do not have to be in the visual range of each other to support a wireless link. (5) For any wireless standard there remains to be a tradeoff between communication range, data rate and power consumption. The Bluetooth standard tries to support users by providing three different classes of Bluetooth devices that provide communication ranges from 10m to 100m and consume a 1mw to 100mE of power, respectively. For wireless applications that need bigger communication ranges Bluetooth can be suitable if intermediate devices forward messages. (6) Bluetooth is a standard that makes sure that every certified Bluetooth device can communicate with every other certified Bluetooth. It gets continuously improved by a group of companies organizing themselves in the so-called Bluetooth interest group. (7) Because Bluetooth devices are manufactured in high numbers and due to the fact that Bluetooth is operating in the license-free frequency band the price for Bluetooth devices is very low. (8) Many Bluetooth software stacks are available; some of them even for free. That is why the time for the development of new Bluetooth software and protocols is dramatically reduced. However, although free Bluetooth software stacks are available the development of software based on Bluetooth can remain to be a pain. This has two main reasons: (1) For interfacing the Bluetooth stack the user still needs quite a lot of knowledge about the wireless standard. (2) Because Bluetooth was originally designed as a cable replacement with a central host device Bluetooth networks have some major restrictions: As shown in Figure 1(a) up to 8 Bluetooth devices can be connected to form a so-called piconet. In this piconet one Bluetooth device acts as the central controlling device called master while the other 7 devices act as slaves. The master is controlling the whole communication in the piconet and only the master device can directly send data to the other devices. The network structure becomes even more difficult if more than 8 devices shall be connected. In this case as shown in Figure 1(b) a slave device has to act as a master in another piconet and a so-called scatternet, a composition of

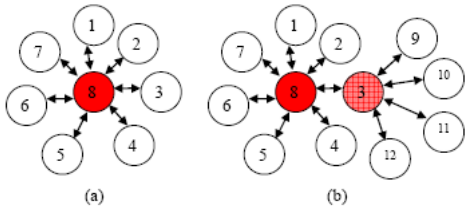

Fig. 1. (a) Piconet with master device number 8 and 7 slaves (number 1-7). (b) To form a Bluetooth network with more than 8 actively connected devices, a scatternet has to be created. There for device number 3 becomes a master/slave device. It acts like a slave for the original piconet with master number 8 and like a master device for the devices number 9-12.

Rico Möckel is with the Institute of Neuroinformatics at the Uni/ETH Zürich. moeckel@ini.phys.ethz.ch

Alexander Spröwitz, Jérôme Maye and Auke Jan Ijspeert are with the School of Computer and Communication Science at the Ecole Polytechnique Fédérale de Lausanne (alexander.sproewitz, jerome.maye, auke.ijspeert)@epfl.ch.

piconets, is formed. In this scatternet only directly connected devices know about each other and can directly send data to each other.

There are many approaches to the question on how to build up a scatternet in the literature. Please see [2], [3], [4], [5], [6] and [8] for examples. However many of them like in [8] are more of theoretical value for a user that wants to build up a real wireless sensor network or a robot hardware infrastructure. Since that protocols are mostly simulated in software a user has to face a couple of challenges: (1) First the designer has to find the right protocol for his purpose. This is already a challenge since some protocols propose changes to the Bluetooth standard and therefore do not work with the available hardware, produce too complex networks or use some random techniques that do not guarantee the connectivity of all Bluetooth devices like in [4]. Other protocols like in [3] use the possibility to park slaves in a network. These parked slaves cannot actively take part in the communication process but have to wait until another device is parked and they are recovered by the master of their piconet into active mode. The main problem of scatternet formation is that a general approach normally does not solve the specific problem of the system designer but creates a huge overhead [7] just for creating a wireless infrastructure to send some data. That is why our protocol focuses on providing the necessary functionality like transparent data submission and remote control of the Bluetooth network by reducing the overhead as much as possible. (2) The second challenge that a user has to face is to find the necessary hardware for providing the Bluetooth network. Given a considerable number of Bluetooth devices on the market this is not a simple problem. Especially when taking into account that many embedded devices do not support the whole Bluetooth standard like scatternet functionality but need some external microcontroller or embedded PC. Furthermore there are scatternet protocols like in [5] that inquire additional hardware like a GPS receiver. For our protocol no additional hardware than a Bluetooth chip is necessary. (3) Finally a system designer has to solve the problem of the implementation of the scatternet protocol. This includes the choice for an embedded Bluetooth stack that is supporting all necessary functionality. We chose the ZV4002 embedded Bluetooth device from Zeevo that is running the whole Bluetooth stack plus our protocol on-chip. When using the same Bluetooth chip and our firmware and connecting it via a serial interface to your favorite microcontroller your system is ready for Bluetooth including scatternet functionality.

The SNP is automatically learning the shortest path to the receiver and creates only a minimal overhead to the Bluetooth stack. To make sure that our scatternet protocol works together with avaible hardware and protocol stacks we tested the SNP in real applications and provide data not from simulation but from experiments with real hardware. This has the clear advantage that we can show second order effects like from parallel communication of different piconets or communication overhead from devices that act in two piconets at the same time that is difficult to be simulated correctly and often ignored. So we provide a simple and robust wireless interface that can be copied by any designer of a robot or a wireless sensor network system in terms of hardware and firmware and that is characterized and reliably working in real world applications.

Section II explains the mechanism of the scatternet protocol while section III describes possible applications and shows how we successfully applied our scatternet protocol to modular robots and sensor networks. Section IV concludes and gives some outlook over future work.

II. Scatternet Protocol

We developed a new scatternet protocol (SNP) layer that makes the Bluetooth communication transparent. A user who wants to send data to any other device in a Bluetooth network simply sends a packet with the address of the receiver into the network. The SNP is responsible for finding the shortest path through the network and to guarantee that the packet is received by the target device. When the network is changed the SNP is adapting and learning new paths. Only local information is used to find the shortest path. The SNP extracts the routing information by looking at the data packets that are passing the Bluetooth device it is running on. The SNP also supports broadcasts and gives full remote control for all connected Bluetooth devices. This allows a user to control all Bluetooth devices in a scatternet from a single host device. To support these functionalities we added three new mechanisms to the original Bluetooth stack: (1) SNP addresses are new user defined addresses. (2) SNP packets are responsible to carry the payload through the scatternet. (3) SNP friend tables contain local routing information that is used to forward the SNP packets towards the receiver.

Fig. 2. SNP packets. The header contains 5 Bytes specifying the command, the SNP addresses of the receiver and sender, the number of hops a packet was traveling as well as 1 Byte giving the amount of payload that the packet contains.

A. SNP Addresses

A Bluetooth-MAC address contains 6 Bytes. This is necessary to give a unique address to every Bluetooth device on the planet. However, only up to 256 Bluetooth devices can be actively connected in a scatternet at a time. To reduce the overhead of addressing a Bluetooth device when using the SNP layer we decided to support the user by the opportunity to give every Bluetooth device a new SNP address using 1 Byte. SNP addresses between 0 and 254 can be given to the Bluetooth devices by the user. Address number 255 is reserved for broadcasts. An internal SNP routing table maps the user-defined SNP addresses to the Bluetooth-MAC addresses.

B. SNP Packets

When data should be sent from one Bluetooth device to another through a scatternet special packets have to be formed. An overview of the packets used for the scatternet protocol is given in Figure 2. The overall overhead for a SNP packet header is 5 Bytes containing (1) a SNP command, (2) the SNP address of the receiving device, (3) the SNP address of the sending device, (4) a hopping counter and (5) 1 Byte specifying the amount of payload that belongs to the packet. Up to 255 Bytes of payload can follow after the header. If more data should be sent, more than one packet has to be created.

The SNP command field contains the command that should be executed by the SNP layer. For a user 3 different commands are relevant: (1) The SNP command s stands for packets that contain data. These packets are forwarded by each Bluetooth device on the path from the sender to the receiver which transmits the packet via UART to a host device. (2) The SNP command c is used to tell a Bluetooth device to connect another one. (3) The SNP command d is used to tell a Bluetooth device to disconnect another one. It is important to mention that the SNP layer forwards all packets to the receiver without looking at the command field. That is why a remote control of all Bluetooth devices in the scatternet becomes very simple. By specifying the SNP address of the receiver device this can be forced to execute any command in the same way as it would be directly connected to the host via UART. To also get feedback during remote control, a remote address can be specified to which all devices send their error and status messages.

The hopping counter is initialized to 0 once a packet is created for the first time. It is incremented every time the packet is received by a new Bluetooth device in the scatternet. That is why the hopping contains how far a device is away in the scatternet. This information is essential for updating the SNP routing tables and for finding the shortest path in the communication network. A maximum of hops can be specified. If the hopping counter reaches this value the packet is deleted and not forwarded any more. This mechanism can be very useful to avoid infinite loops in more complicated network graphs that also include cycles.

SNP address	BT address	Forward address	Num hops
5	42:49:52:47:00:01	255	255
8	42:49:52:47:00:02	255	255
11	42:49:52:47:00:03	255	255
2	42:49:52:47:00:04	8	1
7	42:49:52:47:00:05	255	255

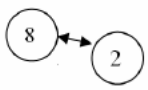

Fig. 3. The friend table contains the necessary information for the SNP protocol.

C. SNP Friend Table

To allow automatic packet forwarding as well as to determine the mapping of Bluetooth-MAC addresses and SNP addresses, we implemented internal routing tables that we call SNP friend table. An example of a friend table of a Bluetooth device with the SNP address 8 that is connected to another Bluetooth device with the SNP address 2 is shown in Figure 3. Column 1 of the SNP friend table stores all SNP addresses the Bluetooth device knows about while column 2 determines the corresponding Bluetooth-MAC addresses. Column 3 in the friend table stores the SNP address of the device to which a packet has to be forwarded. To reduce the overhead of the SNP layer and to use as much local information as possible the Bluetooth devices do not store the whole structure of the communication network. The only information that is necessary to be stored about the network is to which device a packet has to be forwarded to send the data to the receiver device. This strategy that is based on local information and decision making is one of the powerful strengths of our SNP layer. It minimizes the overhead for communication between devices as well as the information that has to be stored in each device and supports a fast and parallel decision making process. No additional exchange of information between Bluetooth devices is necessary to find out to which device the packet has to be forwarded to. Column 4 in the friend table in Figure 3 (a) determines the minimum number of hops that are necessary to send a packet to a receiver device. This information is useful when the shortest path to the receiver should be automatically learned as we will describe later. We set the forward address and the number of hops to 255 if a device is not connected. Since in Figure 3 device number 2 is directly connected the SNP address of device number 8 is copied into the column 3 that stores the forward address and a 1 is stored in column 4 since device number 2 is only one hop away. To make the scatternet

(a)

SNP address	BT address	Forward address	Num hops
5	42:49:52:47:00:01	8	1
8	42:49:52:47:00:02	255	255
11	42:49:52:47:00:03	255	255
2	42:49:52:47:00:04	8	1
7	42:49:52:47:00:05	255	255

(b)

SNP address	BT address	Forward address	Num hops
5	42:49:52:47:00:01	8	1
8	42:49:52:47:00:02	255	255
11	42:49:52:47:00:03	255	255
2	42:49:52:47:00:04	8	1
7	42:49:52:47:00:05	5	2

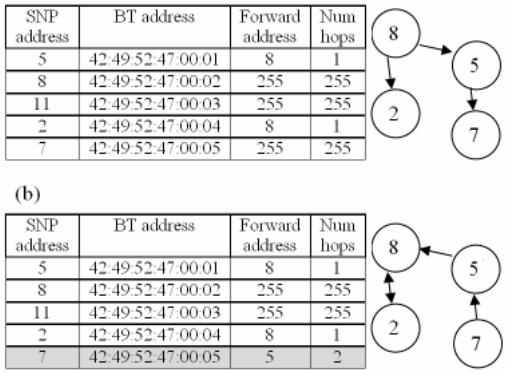

Fig. 4. (a) If a packet should be send to a device to which the path is unknown, the packet as well as a search packet gets broadcasted. (b) The answer packet from the receiver is used to update the friend table.

protocol as robust as possible a Bluetooth device broadcasts the message to all directly connected devices (except to the device it received the message from) every time it should send data to another device to which the path is unknown. Imagine device number 8 belongs to the same network like device number 2 and 7 as shown in Figure 4(a) but does not know about device number 7. In this case device 8 sends all packets that are addressed for device 7 to all directly connected neighbors to make sure that the device 7 gets the

packet if it belongs to the network. This broadcast makes the scatternet protocol very robust but can also comprise a huge overhead in a complicated network. To minimize the overhead device 8 also broadcasts a search packet with the receiver address 7. If device number 7 receives the search packet it directly sends back an answer packet to device number 8 which device number 8 uses to update its friend table. As shown in Figure 4(b) it stores device number 5 as the forward address because it received the answer message over this device and a 2 for the number of hops that device number 7 is away. The number of hops is determined from the hopping counter field in the answer packet.

The hopping counter is useful when the shortest path to the receiver should be automatically learned like in our scatternet protocol. To determine the shortest path also in a communication network that contains cycles every Bluetooth device checks the header of every packet it receives, even if this packet has only to be forwarded. If the packet comes from an unknown device the friend table gets updated. That is why device number 7 in Figure 4 does not have to send a search message if it wants to send some data to device number 8. The original message from device number 8 is already used to learn the path to this device. For updating the path to another device, different strategies can be used. (1) To make the protocol as robust as possible the last known path can be used. (2) To learn the shortest path the hopping counter of a packet can be compared with the hopping counter in the friend table. The friend table gets only updated if the number of hops in the friend table is bigger than the number of hops stored in the packet. (3) However, to find the optimal path more effort has to be made since the optimal path in a communication network with cycles does not necessarily need to be the shortest path if there is already a lot of traffic on that path while other paths are unused. To reduce the overhead of our scatternet protocol we decided to always learn the shortest path.

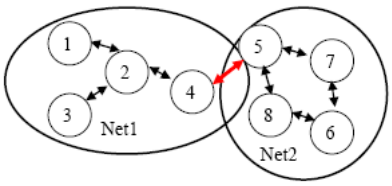

Fig. 5. To reduce the protocol overhead device 4 and 5 exchange a part of their friend table when they get connected. Device 4 forwards its friend table into the network Net2. Device 5 forwards its friend table into the network Net1. In case that Net1 and Net2 get divided because device 4 and 5 get disconnected device 4 and 5 inform the member of Net1 and Net 2 about all disconnected devices of Net2 and Net1 respectively.

D. GotConnected and GotDisconnect Packets

Especially when two big Bluetooth scatternets should be merged learning the routing tables by sending search and answer packages can mean a huge overhead. To avoid unnecessary messages we decided to introduce GotConnected and GotDisconnected packages. When merging two Bluetooth networks or dividing a network into two smaller ones the directly connected or disconnected devices send a part of their friend table to the newly connected device or inform the remaining members of their network about the disconnected devices, respectively.

If e.g. like shown in Figure 5 two Bluetooth networks Net1 and Net2 should be merged by directly connecting device 4 of Net1 and device 5 of Net2, device 4 sends a GotConnected package into Net2. This package contains the SNP addresses and hop values of the members of Net1 in the friend table of device 4. Device 5 reads this package and updates its friend table accordingly now knowing that e.g. a message with the receiver address 1 has to be forwarded towards device 4. Afterwards device 5 broadcasts the connect message to the members of Net2. Since Net2 contains a cycle the GotConnected message would be forwarded until the maximum allowed number of hops is reached. To avoid this each Bluetooth device looks into the GotConnected package and stops forwarding it if it does not contain new information. E.g. device 6 in Net2 will receive the GotConnected message both from device 7 and 8. If it receives the message from device 7 first, device 6 will update its friend table accordingly. When device 6 receives the message a second time from device 8 the GotConnected packet does not contain any new information and device 6 will not have to update its friend table again and delete that message.

In the case that a huge Bluetooth network like in Figure 5 should be separated into two sub-networks Net1 and Net2 by disconnecting the devices 4 and 5, these devices will send a GotDisconnected packet towards the remaining members of their network. E.g. in a network like shown in Figure 5 device 5 will send a packet towards device 7 and 8 to inform them that not only device 4 but also all the other members of Net1 got disconnected. Device 7 and 8 will update their friend tables accordingly resetting all entries of these devices by setting the entries for the forward address and the number of hops to 255. To avoid unnecessary packet forwarding in term of cycles in the network the GotDisconnected packet only gets forwarded by a device that has to update its friend table accordingly like in the case of the GotConnected packet.

III. Discussion and Results

We implemented and tested our scatternet protocol on the embedded Bluetooth device ZV4002 which was developed and distributed by Zeevo Inc. The ZV4002 contains both the analog Bluetooth components and an ARM microcontroller. Zeevo provides an embedded Bluetooth stack that is directly running on the ARM. The stack is certified for the Bluetooth standard 1.2 and supports data rates of up to 600kbit/s. We implemented our scatternet protocol layer on top of the serial port profile. With this strategy we are able to provide a full Bluetooth scatternet protocol encapsulated in a single chip. This simplifies the design of a robot or sensor that should contain a wireless Bluetooth interface dramatically. A designer does not need to create a completely new system but can choose his favorite sensor or microcontroller and connect it via a serial interface to the Bluetooth chip.

The SNP is well suited for sensor networks where the information of several sensors has to be collected and the maximum distance between devices is less than 100m. In combination with our SNP Bluetooth allows the formation of transparent networks of up to 255 devices. Since we do not use special Bluetooth modes like parking all sensors of the network can continuously stay active and e.g. send their data to a central monitoring device. For sensor networks that need direct communication over distances bigger than 100m other communication systems have to be used. However our SNP can help here to support transparent communication.

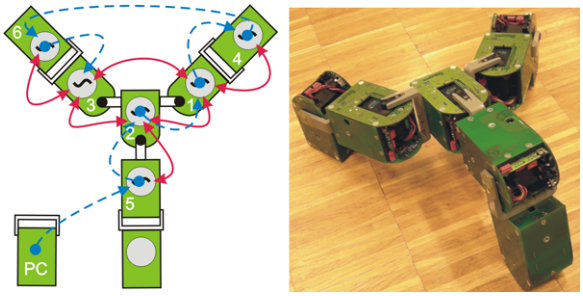

Fig. 6. Example of a tripod robot (right), the corresponding CPG configuration (red lines) and the SNP setup (blue, dashed lines). We implemented a CPG controller for each actuator and exchanged the CPG state variables via the Bluetooth scatternet.

We successfully applied our SNP to the field of modular robots. We implemented a central pattern generator in each module and coupled the controller via the Bluetooth network. Central pattern generators (CPGs) are biological neural networks capable of producing coordinated patterns of rhythmic activity while being initiated and modulated by simple input signals. Therefore they are ideally suited for controlling actuators.

We implemented the CPG model as a system of N coupled amplitude-controlled phase oscillators, one per actuator. An oscillator i is implemented as follows:

$$\dot{\phi}_i = \omega_i + \sum_j (w_{ij}\, r_j \sin(\phi_j - \phi_i - \varphi_{ij}) \quad (1)$$

$$\ddot{r}_i = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i) \quad (2)$$

$$\ddot{x}_i = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i) \quad (3)$$

$$\theta_i = x_i + r_i \cos(\phi_i) \quad (4)$$

where θ_i is the oscillating set-point (in radians) extracted from the oscillator, and ϕ_i, r_i and x_i are state variables that encode respectively the phase, the amplitude, and the offset of the oscillations (in radians). The parameters ω_i, R_i and X_i are control parameters for the desired frequency, amplitude and offset of the oscillations. The parameters w_{ij} and ϕ_{ij} are respectively coupling weights and phase biases which determine how oscillator j influences oscillator i. An oscillator i receives the value of state variables ϕ_j and r_j of neighbor modules j via the Bluetooth communication protocol. The parameters a_r and a_x are constant positive gains ($a_r = a_x = 20$ [rad/s]). The equations were designed such that the output of the oscillator θ_i in Equation (4) exhibits limit cycle behavior i.e. produces a stable periodic output. From any initial conditions, the state variables r_i and x_i will asymptotically and monotonically converge to R_i and X_i. This allows one to smoothly modulate the amplitude and offset of oscillations.

An example robot configuration of a modular robot that we successfully tested is given in Figure 6. We constructed a tripod from robot modules each containing a servo motor. Each actuator is controlled through a CPG that is implemented on an ARM microcontroller on each robot module. We believe that this experiment provides a good way of showing the power of our SNP since to allow coordination between the CPG controllers they have to be coupled. During the experiments we coupled 6 CPGs over our Bluetooth scatternet protocol by sending packets containing the internal state variables ϕ_j and r_j of a CPG to the coupled neighbors every 250ms. Furthermore, we used the Bluetooth network to send commands to the devices and monitor internal states.

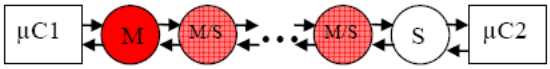

Fig. 7. Test setup for measuring the delays in the Bluetooth scatternet. M is a master, S a slave and M/S is a device that acts as a slave in one piconet and as a master in another piconet, respectively.

To evaluate the performance of the SNP further we arranged up to 10 Bluetooth modules in a chain as shown in Figure 7. We connected one microcontroller both to the beginning (C1) and end (C2) of the chain and measure the time that a packet takes from C1 to C2 and back to C1. Figure 8 shows the mean delay and standard deviation measured for packets containing 7, 25 and 48 Bytes in a chain of 2 to 10 Bluetooth devices. We repeated the experiments 20 times. The graphs in Figure 8 show no visible difference for the different packet sizes. We believe that the reason can be found in the Bluetooth stack. Since the lower level protocol layers add 72 bits of access code and a 54 bit header to the data before transmission the different packet sizes do not have a huge impact. Thus for short packets the transmission time mostly depends on the number of hops between sender and receiver but not on the amount of data. Figure 8 shows further that the transmission delays are broadly spread and the standard deviation increases with the number of hops. This can be explained by the non-deterministic time and frequency sharing in the scatternet. Within a piconet the master is responsible for selecting a specific sequence for the frequency hopping and for assigning time slots to its slaves to make sure that all slaves are synchronized. Depending on when a packet is received by a slave it might be able to transmit it directly to its master or it might have to wait until it got a time slot. Further delays occur because of the frequency sharing between different piconets. Since the frequency sequences of different piconets are not synchronized, overlaps in the frequency band and packet collisions occur. This explains why e.g. sending data within a scatternet with four hops does not just need double

the amount of time of the data transmission in a piconet with two hops but introduces additional delays. We measured e.g. a mean delay of 27ms fors a 5-Byte packet using two hops but 88ms in a four-hop scatternet. The serial communication and computation on the microcontroller takes approximately 2.2ms. So the transmission in the four-hop scatternet should just take 51.8ms. We believe that the additional delay of 36.2ms can be explained by the non-synchronized frequency hopping of the neighboring piconets and the fact that the devices that act both as master and slaves have to listen in two piconets.

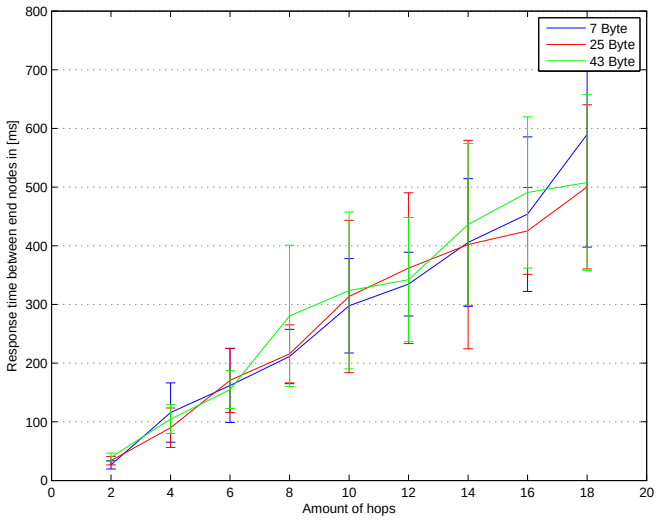

Fig. 8. Delays for transmission of data packets in the wireless Bluetooth scatternet for different numbers of hops in the network.

IV. Conclusion and Future Work

We presented a protocol layer that supports transparent wireless communication in Bluetooth scatternets and showed how our protocol can be used in real robot and sensor network applications. By using our protocol the original Bluetooth standard is getting more flexible and allows a user virtually to use Bluetooth like a communication network where all communication devices are directly connected with each other. A user does not have to take care about the physical structure of the Bluetooth network but simply sends a message into the network. Defining the address of the receiver the user can be sure that the message gets received. By using only local decision making we can reduce the overhead of our scatternet protocol dramatically.

The only phase remaining where a user has to think about the physical structure of the Bluetooth network is when the user is building up or changing the network structure. Although we are already supporting the user during this phase by allowing the user to build up complex network structures via remote control this can remain a challenge. That is why we are currently investigating in a new Bluetooth Optimization Protocol (BOP) that will be responsible for building up efficient scatternets based on the activity in the network.

V. Acknowledgment

We gratefully acknowledge the technical support of André Guignard, Andres Upegui, Elmar Dittrich, Adamo Maddalena, André Badertscher and Peter Brühlmeier in the design and construction of the robot modules.

References

[1] J. Haartsen, "BLUETOOTH - the universal radio interface for ad hoc wireless connectivity," *Ericsson Review*, vol.3, pp. 110–117, 1998.

[2] G. Záruba, S. Basagni and I. Chlamtac, "BlueTrees—Scatternet Formation to enable Bluetooth-based personal area networks", *Proceedings of the IEEE International Conference on Communications*, vol. 1, pp. 273–277, June 2001.

[3] C. Petrioli, S. Basagni and I. Chlamtac, "Configuring BlueStars: Multihop scatternet formation for Bluetooth networks", *IEEE Transactions on Computers*, no. 52, vol. 6, pp. 779–790, 2003.

[4] Z. Wang, R. J. Thomas and Z. Haas, "BlueNet—A new scatternet formation scheme", *Proceedings of the 35th Hawaii International Conference on System Science*, pp. 9, January 2002.

[5] X. Li and I. Stojmenovic, "Partial Delaunay triangulation and degree limited localized Bluetooth scatternet formation", *Proceedings of AD-HOC NetwOrks and Wireless*, September 2002.

[6] C. Petrioli and S. Basagni, "Degree-constrained multihop scatternet formation for Bluetooth networks", *Proceedings of the IEEE Globecom*, vol. 1, pp. 222–226, November 2002.

[7] S. Basagni, R. Bruno, G. Mambrini, C. Petroli, "Comparative Performance Evaluation of Scatternet Formation Protocols for Networks of Bluetooth Devices", *IEEE Transactions on Wireless Networks*, vol. 10, pp. 197–213, 2004.

[8] R. Kapoor, M. Y. Sanadidi, M. Gerla. "An Analysis of Bluetooth Scatternet Topologies", *IEEE International Conference on Communications*, vol. 1, pp. 266–270, May 2003.

[9] R. Möckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. J. Ijspeert. Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface. *Industrial Robot*, vol. 33(4), pp. 285–290, 2006.

[10] A. Spröwitz, R. Möckel, J. Maye, A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization", submitted to: *International Journal of Robotics Research* (under review).