A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternets

András Rácz, György Miklós, Ferenc Kubinszky, András Valkó
Traffic Analysis and Network Performance Lab., Ericsson Research
Laborc 1, 1037 Budapest, Hungary
Ph: +36-1-4377621, Fax: +36-1-4377767
Andras.Racz@eth.ericsson.se

ABSTRACT

The emergence of Bluetooth as a default radio interface allows handheld devices to be rapidly interconnected into ad hoc networks. Bluetooth allows large numbers of piconets to form a scatternet using designated nodes that participate in multiple piconets. A unit that participates in multiple piconets can serve as a bridge and forwards traffic between neighbouring piconets. Since a Bluetooth unit can transmit or receive in only one piconet at a time, a bridging unit has to share its time among the different piconets. To schedule communication with bridging nodes one must take into account their availability in the different piconets, which represents a difficult, scatternet wide coordination problem and can be an important performance bottleneck in building scatternets. In this paper we propose the Pseudo-Random Coordinated Scatternet Scheduling (PCSS) algorithm to perform the scheduling of both intra and inter-piconet communication. In this algorithm Bluetooth nodes assign meeting points with their peers such that the sequence of meeting points follows a pseudo random process that is different for each pair of nodes. The uniqueness of the pseudo random sequence guarantees that the meeting points with different peers of the node will collide only occasionally. This removes the need for explicit information exchange between peer devices, which is a major advantage of the algorithm. The lack of explicit signaling between Bluetooth nodes makes it easy to deploy the PCSS algorithm in Bluetooth devices, while conformance to the current Bluetooth specification is also maintained. To assess the performance of the algorithm we define two reference case schedulers and perform simulations in a number of scenarios where we compare the performance of PCSS to the performance of the reference schedulers.

Keywords

Bluetooth, scheduling, inter-piconet communication, scatternet

1. INTRODUCTION

Short range radio technologies enable users to rapidly interconnect handheld electronic devices such as cellular phones, palm devices or notebook computers. The emergence of Bluetooth [1] as default radio interface in these devices provides an opportunity to turn them from stand-alone tools into networked equipment. Building Bluetooth ad hoc networks also represents, however, a number of new challenges, partly stemming from the fact that Bluetooth was originally developed for single hop wireless connections. In this paper we study the scheduling problems of inter-piconet communication and propose a lightweight scheduling algorithm that Bluetooth nodes can employ to perform the scheduling of both intra and inter-piconet communication.

MobiHOC 2001, Long Beach, CA, USA

Bluetooth is a short range radio technology operating in the unlicensed ISM (Industrial-Scientific-Medical) band using a frequency hopping scheme. Bluetooth (BT) units are organized into *piconets*. There is one Bluetooth device in each piconet that acts as the master, which can have any number of slaves out of which up to seven can be active simultaneously. The communication within a piconet is organized by the master which polls each slave according to some polling scheme. A slave is only allowed to transmit in a slave-to-master slot if it has been polled by the master in the previous master-to-slave slot. In Section 3 we present a brief overview of the Bluetooth technology.

A Bluetooth unit can participate in more than one piconet at any time but it can be a master in only one piconet. A unit that participates in multiple piconets can serve as a bridge thus allowing the piconets to form a larger network. We define *bridging degree* as the number of piconets a bridging node is member of. A set of piconets that are all interconnected by such bridging units is referred to as a *scatternet* network (Figure 1). Since a Bluetooth unit can transmit or receive in only one piconet at a time, bridging units must switch between piconets on a time division basis. Due to the fact that different piconets are not synchronized in time a bridging unit necessarily loses some time while switching from one piconet to the other. Furthermore, the temporal unavailability of bridging nodes in the different piconets makes it difficult to coordinate the communication with them, which impacts throughput and can be an important performance constraint in building scatternets.

There are two important phenomena that can reduce the efficiency of the polling based communication in Bluetooth scatternets:

- slaves that have no data to transmit may be unnecessarily polled, while other slaves with data to transmit may have to wait to be polled; and
- at the time of an expected poll one of the nodes of a master-slave node pair may not be present in the piconet (the slave

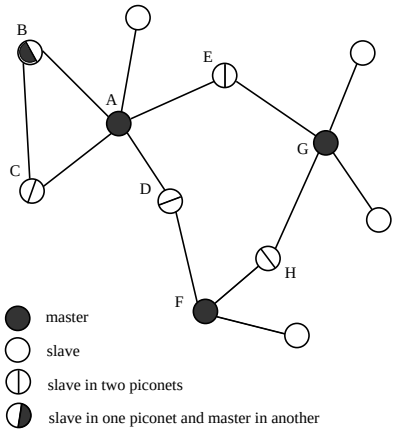

Figure 1: Example scatternet

that is being polled is not listening or the master that is expected to poll is not polling).

The first problem applies to polling based schemes in general, while the second one is specific to the Bluetooth environment. In order to improve the efficiency of inter-piconet communication the scheduling algorithm has to coordinate the presence of bridging nodes in the different piconets such that the effect of the second phenomenon be minimized.

However, the scheduling of inter-piconet communication expands to a scatternet wide coordination problem. Each node that has more than one Bluetooth links have to schedule the order in which it communicates with its respective neighbours. A node with multiple Bluetooth links can be either a piconet master or a bridging node or both. The scheduling order of two nodes will mutually depend on each other if they have a direct Bluetooth link in which case they have to schedule the communication on their common link for the same time slots. This necessitates some coordination between the respective schedulers. For instance in Figure 1 the scheduling order of node A and the scheduling order of its bridging neighbours, B, C, D and E mutually depend on each other, while nodes D and E further effects nodes F, G and H as well. Furthermore, the possible loops in a scatternet (e.g., A-E-G-H-F-D) makes it even more complicated to resolve scheduling conflicts.

In case of bursty traffic in the scatternet the scheduling problem is further augmented by the need to adjust scheduling order in response to dynamic variation of traffic intensity. In a bursty traffic environment it is desirable that a node spends most of its time on those links that have a backlogged burst of data.

One way to address the coordination problem of inter-piconet scheduling is to explicitly allocate, in advance, time slots for communication in each pair of nodes. Such a *hard coordination* approach eliminates ambiguity with regards to a node's presence in piconets, but it implies a complex, scatternet wide coordination problem and requires explicit signaling between nodes of a scatternet. In the case of bursty traffic, hard coordination schemes generate a significant computation and signaling overhead as the communication slots have to be reallocated in response to changes in traffic intensity and each time when a new connection is established or released.

In this paper we propose the Pseudo-Random Coordinated Scatternet Scheduling algorithm which falls in the category of *soft coordination* schemes. In soft coordination schemes nodes decide their presence in piconets based on local information. By nature, soft coordination schemes cannot guarantee conflict-free participation of bridging nodes in the different piconets, however, they have a significantly reduced complexity. In the PCSS algorithm coordination is achieved by implicit rules in the communication without the need of exchanging explicit control information. The low complexity of the algorithm and its conformance to the current Bluetooth specification allow easy implementation and deployment.

The first key component of the algorithm is the notion of *checkpoints* which are defined in relation to each pair of nodes that are connected by a Bluetooth link and which represent predictable points in time when packet transmission can be initiated on the particular link. In other words, checkpoints serve as regular meeting points for neighboring nodes when they can exchange packets. In order to avoid systematic collision of checkpoints on different links of a node the position of checkpoints follows a pseudo random sequence that is specific to the particular link the checkpoints belong to.

The second key component of the algorithm is the dynamic adjustment of checking intensity, which is necessary in order to effectively support bursty data traffic. Bandwidth can be allocated and deallocated to a particular link by increasing and decreasing checkpoint intensity, respectively.

To assess the performance of the algorithm we define two reference schedulers and relate the performance of the PCSS scheme to these reference algorithms in a number of simulation scenarios.

The remainder of the paper is structured as follows. In Section 2 we give an overview of related work focusing on Bluetooth scheduling related studies available in the literature. Section 3 gives a brief overview of the Bluetooth technology. In Section 4 and 5 we introduce the proposed algorithm. In Section 6 we define the reference schedulers. Finally, in Section 7 we present simulation results.

2. RELATED WORK

A number of researchers have addressed the issue of scheduling in Bluetooth. Most of these studies have been restricted, however, to the single piconet environment, where the fundamental question is the polling discipline used by the piconet master to poll its slaves. These algorithms are often referred to as intra-piconet scheduling schemes. In [7] the authors assume a simple round robin polling scheme and investigate queueing delays in master and slave units depending on the length of the Bluetooth packets used. In [5] Johansson *et al.* analyze and compare the behavior of three different polling algorithms. They conclude that the simple round robin scheme may perform poorly in Bluetooth systems and they propose a scheme called Fair Exhaustive Polling. The authors demonstrate the strength of this scheme and argue in favor of using multi-slot packets. Similar conclusions are drawn by Kalia *et al.* who argue that the traditional round robin scheme may result in waste and unfairness [8]. The authors propose two new scheduling disciplines that utilize information about the status of master and slave queues. In [9, 10] the authors concentrate on scheduling policies designed with the aim of low power consumption. A number of scheduling policies are proposed which exploit either the *park* or *sniff* low power modes of Bluetooth.

Although the above studies have revealed a number of important performance aspects of scheduling in Bluetooth piconets, the algorithms developed therein are not applicable for inter-piconet communication. In [6] the authors have shown that constructing an optimal link schedule that maximizes total throughput in a Bluetooth scatternet is an NP hard problem even if scheduling is performed by a central entity. The authors also propose a scheduling algorithm referred to as Distributed Scatternet Scheduling Algorithm (DSSA), which falls in the category of distributed, hard coordination schemes. Although the DSSA algorithm provides a solution for scheduling communication in a scatternet, some of its idealized properties (e.g., nodes are aware of the traffic requirements of their neighbours) and its relatively high complexity make it difficult to apply it in a real life environment.

There is an ongoing work in the Personal Area Networking (PAN) working group of the Bluetooth Special Interest Group (SIG) [2] to define an appropriate scheduling algorithm for Bluetooth scatternets.

3. BLUETOOTH BACKGROUND

Bluetooth is a short range radio technology that uses frequency hopping scheme, where hopping is performed on 79 RF channels spaced 1 MHz apart. Communication in Bluetooth is always between master and slave nodes. Being a master or a slave is only a logical state: any Bluetooth unit can be a master or a slave. The Bluetooth system provides full-duplex transmission based on slotted Time Division Duplex (TDD) scheme, where each slot is 0.625 ms long. Master-to-slave transmission always starts in an even-numbered time slot, while slave-to-master transmission always starts in an odd-numbered time slot. A pair of master-to-slave and slave-to-master slots are often referred to as a frame. The communication within a piconet is organized by the master which polls each slave according to some polling scheme. A slave is only allowed to transmit in a slave-to-master slot if it has been polled by the master in the previous master-to-slave slot. The master may or may not include data in the packet used to poll a slave. Bluetooth packets can carry synchronous data (e.g., real-time traffic) on Synchronous Connection Oriented (SCO) links or asynchronous data (e.g., elastic data traffic, which is the case in our study) on Asynchronous Connectionless (ACL) links. Bluetooth packets on an ACL link can be 1, 3 or 5 slot long and they can carry different amount of user data depending on whether the payload is FEC coded or not. Accordingly, the Bluetooth packet types DH1, DH3 and DH5 denote 1, 3 and 5 slot packets, respectively, where the payload is not FEC encoded, while in case of packet types DM1, DM3 and DM5 the payload is protected with FEC encoding. There are two other types of packets, the POLL and NULL packets that do not carry user data. The POLL packet is used by the master when it has no user data to the slave but it still wants to poll it. Similarly, the NULL packet is used by the slave to respond to the master if it has no user data. For further information regarding the Bluetooth technology the reader is referred to [1, 3].

4. OVERVIEW OF THE PCSS ALGORITHM

Coordination in the PCSS algorithm is achieved by the unique pseudo random sequence of checkpoints that is specific to each master-slave node pair and by implicit information exchange between peer devices. A checkpoint is a designated Bluetooth frame. The activity of being present at a checkpoint is referred to as *to check*. A master node actively checks its slave by sending a packet to the slave at the corresponding checkpoint and waiting for a response from the slave. The slave node passively checks its master by listening to the master at the checkpoint and sending a response packet in case of being addressed.

The expected behaviour of nodes is that they show up at each checkpoint on all of their links and check their peers for available user data. The exchange of user data packets started at a checkpoint can be continued in the slots following the checkpoint. A node remains active on the current link until there is user data in either the master-to-slave or slave-to-master directions or until it has to leave for a next checkpoint on one of its other links. In the PCSS scheme we exploit the concept of randomness in assigning the position of checkpoints, which excludes the possibility that checkpoints on different links of a node will collide systematically, thus giving the node an equal chance to visit all of its checkpoints. The pseudo random procedure is similar to the one used to derive the pseudo random frequency hopping sequence. In particular, the PCSS scheme assigns the positions of checkpoints on a given link following a pseudo random sequence that is generated based on the Bluetooth clock of the master and the MAC address of the slave. This scheme guarantees that the same pseudo random sequence will be generated by both nodes of a master-slave pair, while the sequences belonging to different node pairs will be different. Figure 2 shows an example for the pseudo random arrangement of checkpoints in case of a node pair A and B. The length of the current base checking interval is denoted by $T_{check}^{(i)}$ and the current checking intensity is defined accordingly as $\frac{1}{T_{check}^{(i)}}$. There is one checkpoint within each base checking interval and the position of the checkpoint within this window is changing from one time window to the other in a pseudo random manner.

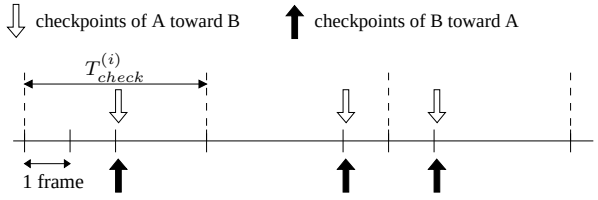

Figure 2: Pseudo-random positioning of checkpoints

Since the pseudo random sequence is different from one link to another, checkpoints on different links of a node will collide only occasionally. In case of collision the node can attend only one of the colliding checkpoints, which implies that the corresponding neighbours have to be prepared for a non-present peer. That is, the master might not poll and the slave might not listen at a checkpoint. We note that a collision occurs either if there are more than one checkpoints scheduled for the same time slot or if the checkpoints are so close to each other that a packet transmission started at the first checkpoint necessarily overlaps the second one. Furthermore, if the colliding checkpoints belong to links in different piconets, the necessary time to perform the switch must be also taken into account.

During the communication there is the possibility to increase or decrease the intensity of checkpoints depending on the amount of user data to be transmitted and on the available capacity of the node. According to the PCSS algorithm a node performs certain traffic measurements at the checkpoints and increases or decreases the current checking intensity based on these measurements. Since

nodes decide independently about the current checking intensity without explicit coordination, two nodes on a given link may select different base checking periods. In order to ensure that two nodes with different checking intensities on the same link can still communicate we require the pseudo random generation of checkpoints to be such that the set of checkpoint positions at a lower checking intensity is a subset of checkpoint positions at any higher checking intensities. In the Appendix we are going to present a pseudo random scheme for generating the position of checkpoints, which has the desired properties.

5. OPERATION OF PCSS

In what follows, we describe the procedures of the PCSS algorithm. We start by the initialization process which ensures that two nodes can start communication as soon as a new link has been established or the connection has been reset. Next, we describe the rules that define how nodes calculate their checkpoints, decide upon their presence at checkpoints and exchange packets. Finally, we present the way neighboring nodes can dynamically increase and decrease of checkpoint intensity.

5.1 Initialization

In the PCSS algorithm there is no need for a separate initialization procedure to start communication, since the pseudo random generation of checkpoints is defined such that once a master slave node pair share the same master's clock and slave's MAC address information, it is guaranteed that the same pseudo random sequence will be produced at each node. That is, it is guaranteed that two nodes starting checkpoint generation at different time instants with different checking intensities will be able to communicate. It is the own decision of the nodes to select an appropriate initial checking intensity, which may depend for example on the free capacities of the node or on the amount of data to transmit. Once the communication is established the increase and decrease procedures will adjust the possibly different initial checking intensities to a common value.

5.2 Communication

A pair of nodes can start exchanging user data packets at a checkpoint, which can expand through the slots following the checkpoint. The nodes remain active on the current link following a checkpoint until there is user data to be transmitted or one of them has to leave in order to attend a checkpoint on one of its other links. After a POLL/NULL packet pair has been exchanged indicating that there is no more user data left the nodes switch off their transmitters/receivers and remain idle until a next checkpoint comes on one of their links. However, during the communication any of the nodes can leave in order to attend a coming checkpoint on one of its other links. After one of the nodes has left the remaining peer will realize the absence of the node and will go idle until the time of its next checkpoint. If the master has left earlier the slave will realize the absence of the master at the next master-to-slave slot by not receiving the expected poll. In the worst case the master has left before receiving the last packet response from the slave, which can be a 5 slot packet in which case the slave wastes 5+1 slots before realizing the absence of the master. Similarly, if the master does not get a response from the slave it assumes that the slave has already left the checkpoint and goes idle until its next checkpoint. Note that the master may also waste 5+1 slots in the worst case before realizing the absence of the slave.

A node stores the current length of the base checking interval and the time of the next checkpoint for each of its Bluetooth links separately. For its i^{th} link a node maintains the variable $T_{check}^{(i)}$ to store the length of the current base checking period in number of frames and the variable $t_{check}^{(i)}$, which stores the Bluetooth clock of the master at the next checkpoint. After passing a checkpoint the variable $t_{check}^{(i)}$ is updated to the next checkpoint by running the pseudo random generator ($PseudoChkGen$) with the current value of the master's clock $t^{(i)}$ and the length of the base checking period $T_{check}^{(i)}$ and with the MAC address of the slave $A_{slave}^{(i)}$ as input parameters; $t_{check}^{(i)} = PseudoChkGen(T_{check}^{(i)}, A_{slave}^{(i)}, t^{(i)})$. The procedure $PseudoChkGen$ is described in the Appendix.

There is a maximum and minimum checking interval $T_{max} = 2^{f_{max}}$ and $T_{min} = 2^{f_{min}}$, respectively. The length of the checking period must be a power of 2 number of frames and it must take a value from the interval $[2^{f_{min}}, 2^{f_{max}}]$.

5.3 Increasing and Decreasing Checking Intensity

The increase and decrease procedures are used to adjust the checking intensity of a node according to the traffic intensity and to the availability of the peer device. Each node decides independently about the current checking intensity based on traffic measurements at checkpoints.

Since the time spent by a node on a link is proportional to the ratio of the number of checkpoints on that link and the number of checkpoints on all links of the node, the bandwidth allocated to a link can be controlled by the intensity of checkpoints on that link. This can be shown by the following simple calculation.

Let us assume that the node has L number of links and assume further that for the base checking periods on all links of the node it holds that $T_{min} \leq T_{check}^{(i)} \leq T_{max}$, $\forall i = 1, \ldots, L$. Then the average number of checkpoints within an interval of length T_{max} is $N = \sum_{i=1}^{L} \frac{T_{max}}{T_{check}^{(i)}}$, and the average time between two consecutive checkpoints is

$$\overline{t} = \frac{T_{max}}{N} = \frac{1}{\sum\limits_{i=1}^{L} \frac{1}{T_{check}^{(i)}}},$$

provided that the pseudo random generator produces a uniformly distributed sequence of checkpoints. Then, the share of link j from the total capacity of the node is

$$r_j = \frac{1/T_{check}^{(j)}}{\sum\limits_{i=1}^{L} \frac{1}{T_{check}^{(i)}}}.$$

A node has to measure the utilization of checkpoints on each of its links separately in order to provide input to the checking intensity increase and decrease procedures. According to the algorithm a given checkpoint is considered to be utilized if both nodes have shown up at the checkpoint and at least one Bluetooth packet carrying user data has been transmitted or received. If there has not been a successful poll at the checkpoint due to the unavailability of any of the nodes or if there has been only a POLL/NULL packet pair exchange but no user data has been transmitted, the checkpoint is considered to be unutilized. We note that due to packet losses the utilization of a given checkpoint might be interpreted differently by the nodes. However, this does not impact correct operation of the algorithm.

To measure the utilization of checkpoints $\rho^{(i)}$ on the i^{th} link of the node we employ the moving average method as follows. The utilization of a checkpoint equals to 1 if it has been utilized, otherwise it equals to 0. If the checkpoint has been utilized the variable $\rho^{(i)}$ is updated as,

$$\rho^{(i)} = q_{uti} \cdot \rho^{(i)} + (1 - q_{uti}) \cdot 1;$$

if the checkpoint has not been utilized it is updated as,

$$\rho^{(i)} = q_{uti} \cdot \rho^{(i)} + (1 - q_{uti}) \cdot 0,$$

where $0 \leq q_{uti} < 1$ is the time scale parameter of the moving average method. A further parameter of the utilization measurement is the minimum number of samples that have to be observed before the measured utilization value is considered to be confident and can be used as input to decide about increase and decrease of checking intensity. This minimum number of samples is a denoted by $N_{sample,min}$.

Finally, a node also has to measure its total utilization, which is defined as the fraction of time slots where the node has been active (transmitted or received) over the total number of time slots. To measure the total utilization of a node we employ the moving average method again. Each node measures its own utilization $\rho^{(node)}$ and updates the $\rho^{(node)}$ variable after each $N_{uti,win}$ number of slots as follows:

$$\rho^{(node)} = q_{uti}^{(node)} \cdot \rho^{(node)} + (1 - q_{uti}^{(node)}) \cdot \rho^{(win)},$$

where $\rho^{(win)}$ is the fraction of time slots in the past time window of length $N_{uti,win}$ where the node has been active over the total number of time slots $N_{uti,win}$.

If the utilization of checkpoints on link i falls below the lower threshold ρ_{lower}, the current base checking period $T_{check}^{(i)}$ will be doubled. Having a low checkpoint utilization can be either because one or both of the nodes have not shown up at all of the checkpoints or because there is not enough user data to be transmitted. In either cases the intensity of checkpoints has to be decreased. Whenever a decrease or increase is performed on link i the measured utilization $\rho^{(i)}$ must be reset.

Since the parameter $T_{check}^{(i)}$ is one of the inputs to the pseudo random checkpoint generation process, $PseudoChkGen$ the checkpoints after the decrease will be generated according to the new period. Furthermore, due to the special characteristic of the checkpoint generation scheme the remaining checkpoints after the decrease will be a subset of the original checkpoints, which guarantees that the two nodes can sustain communication independent of local changes in checking intensities.

An example for the checking intensity decrease in case of a node pair A and B is shown in Figure 3. First, node A decreases checking intensity by doubling its current base checking period in response to the measured low utilization. As a consequence node B will find node A on average only at every second checkpoint and its measured utilization will decrease rapidly. When the measured utilization at node B falls below the threshold ρ_{lower}, B realizes that its peer has a lower checking intensity and follows the decrease by doubling its current base checking period. Although we have not explicitly indicated in the Figure, it is assumed that there has been user data exchanged at each checkpoint where both nodes were present.

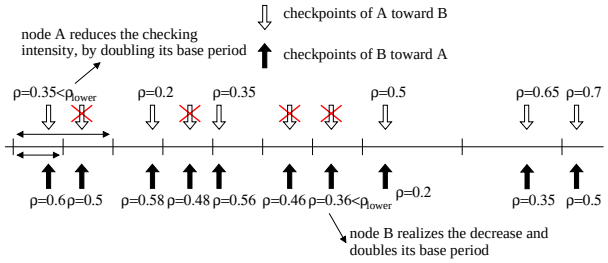

Figure 3: Checking intensity decrease

Recall from the utilization measurement procedure that there is a minimum number of checkpoints $N_{sample,min}$ that has to be sampled before the measured utilization is considered to be confident and can be used to decide about checking intensity decrease. The parameter $N_{sample,min}$ together with the parameter of the moving average method q_{uti} determine the time scale over which the utilization of checkpoints has to be above the threshold ρ_{lower}, otherwise the node decreases checking intensity. It might be also reasonable to allow that the parameter $N_{sample,min}$ and the moving average parameter q_{uti} can be changed after each decrease or increase taking into account for example the current checking intensity, the available resources of the node or the amount of user data to be transmitted, etc. However, in the current implementation we apply fixed parameter values.

After a checkpoint where user data has been exchanged (not only a POLL/NULL packet pair) checking intensity can be increased provided that the measured utilization of checkpoints exceeds the upper threshold ρ_{upper} and the node has available capacity. Formally a checking intensity increase is performed on link i if the following two conditions are satisfied: $\rho^{(i)} > \rho_{upper}$ and $\rho^{(node)} < \rho_{upper}^{(node)}$, where $\rho_{upper}^{(node)}$ is the upper threshold of the total utilization of the node. This last condition ensures that the intensity of checkpoints will not increase unbounded. The intensity of checkpoints is doubled at each increase by dividing the current length of the base checking period $T_{check}^{(i)}$ by 2. For typical values of ρ_{upper} we recommend $0.8 \leq \rho_{upper} \leq 0.9$ in which case the respective ρ_{lower} value should be $\rho_{lower} \leq 0.4$ in order to avoid oscillation of increases and decreases.

Figure 4 shows an example where node A and B communicate and after exchanging user data at the second checkpoint both nodes double the checking intensity. In the Figure we have explicitly indicated whether there has been user data exchanged at a checkpoint or not.

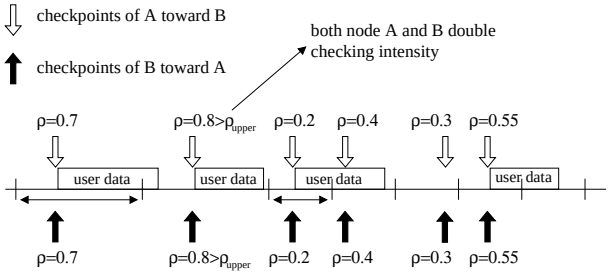

Figure 4: Checking intensity increase

6. REFERENCE ALGORITHMS

In this section we define the Ideal Coordinated Scatternet Scheduler (ICSS) and the Uncoordinated Greedy Scatternet Scheduler (UGSS) reference algorithms. The ICSS algorithm represents the "ideal" case where nodes exploit all extra information when scheduling packet transmissions, which would not be available in a realistic scenario. The UGSS algorithm represents the greedy case where nodes continuously switch among their Bluetooth links in a random order.

6.1 The ICSS Algorithm

The ICSS algorithm is a hypothetical, ideal scheduling algorithm that we use as a reference case in the evaluation of the PCSS scheme. In the ICSS algorithm a node has the following extra information about its neighbours, which represents the idealized property of the algorithm:

- a node is aware of the already pre-scheduled transmissions of its neighbours; and
- a node is aware of the content of the transmission buffers of its neighbours.

According to the ICSS algorithm each node maintains a *scheduling list*, which contains the already pre-scheduled tasks of the node. A task always corresponds to one packet pair exchange with a given peer of the node. Knowing the scheduling list of the neighbours allows the node to schedule communication with its neighbours without overlapping their other communication, such that the capacity of the nodes is utilized as much as possible. Furthermore being aware of the content of the transmission buffers of neighbours eliminates the inefficiencies of the polling based scheme, since there will be no unnecessary polls and the system will be work-conserving.

In the scheduling list of a node there is at most one packet pair exchange scheduled in relation to each of its peers, provided that there is a Bluetooth packet carrying user data either in the transmission buffer of the node or in the transmission buffer of the peer or in both. After completing a packet exchange on a given link the two nodes schedule the next packet exchange, provided that there is user data to be transmitted in at least one of the directions. If there is user data in only one of the directions, a POLL or NULL packet is assumed for the reverse direction depending on whether it is the master-to-slave or slave-to-master direction, respectively. The new task is fitted into the scheduling lists of the nodes using a first fit strategy. According to this strategy the task is fitted into the first time interval that is available in both of the scheduling lists and that is long enough to accommodate the new task. Note that the algorithm strives for maximal utilization of node capacity by trying to fill in the unused gaps in the scheduling lists.

If there is no more user data to be transmitted on a previously busy link, the link goes to idle in which case no tasks corresponding to the given link will be scheduled until there is user data again in at least one of the directions.

An example for the scheduling lists of a node pair A and B is shown in Figure 5. The tasks are labeled with the name of the corresponding peers the different tasks belong to. Each node has as many pre-scheduled tasks in its scheduling list as the number of its active Bluetooth links. A link is considered to be active if there is

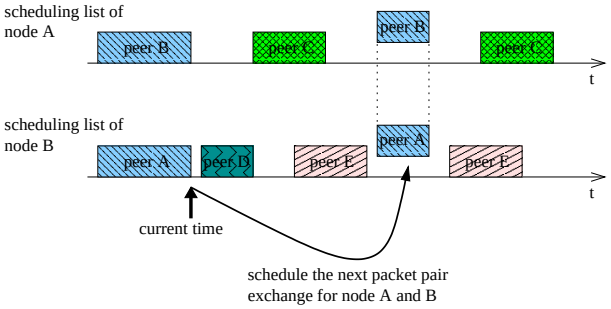

Figure 5: Example for the scheduling lists of a node pair in case of the ICSS algorithm

user data packet in at least one of the directions. Node A has active peers B and C, while node B has active peers A, D and E. After node A and B have finished the transmission of a packet pair they schedule the next task for the nearest time slots that are available in both of their scheduling lists and the number of consecutive free time slots is greater than or equal to the length of the task.

6.2 The UGSS Algorithm

In the UGSS algorithm Bluetooth nodes do not attempt to coordinate their meeting points, instead each node visits its neighbours in a random order. Nodes switch continuously among their Bluetooth links in a greedy manner. If the node has n number of links it chooses each of them with a probability of $1/n$. The greedy nature of the algorithm results in high power consumption of Bluetooth devices.

If the node is the master on the visited link it polls the slave by sending a packet on the given link. The type of Bluetooth packet sent can be a 1, 3 or 5 slot packet carrying useful data or an empty POLL packet depending on whether there is user data to be transmitted or not. After the packet has been sent the master remains active on the link in order to receive any response from the slave. If the slave has not been active on the given link at the time when the master has sent the packet it could not have received the packet and consequently it will not send a response to the master. After the master has received the response of the slave or if it has sensed the link to be idle indicating that no response from the salve can be expected, it selects the next link to visit randomly.

Similar procedure is followed when the node is the slave on the visited link. The slave tunes its receiver to the master and listens for a packet transmission from the master in the current master-to-slave slot. If the slave has not been addressed by the master in the actual master-to-slave slot it immediately goes to the next link. However, if the slave has been addressed it remains active on the current link and receives the packet. After having received the packet of the master the slave responds with its own packet in the following slave-to-master slot. After the slave has sent its response it selects the next link to visit randomly.

7. SIMULATION RESULTS

First, we evaluate the algorithm in a realistic usage scenario, which is the Network Access Point (NAP) scenario. Next we investigate theoretical configurations and obtain asymptotical results that reveals the scaling properties of the algorithm. For instance we investigate the carried traffic in function of the number of forwarding

hops along the path and in function of bridging degree. Both in the realistic and theoretical configurations we relate the performance of the PCSS scheme to the performance of the ICSS and UGSS reference algorithms. Before presenting the scenarios and simulation results we shortly describe the simulation environment and define the performance metrics that are going to be measured during the simulations.

7.1 Simulation Environment

We have developed a Bluetooth packet level simulator, which is based on the Plasma simulation environment [4]. The simulator has a detailed model of all the packet transmission, reception procedures in the Bluetooth Baseband including packet buffering, upper layer packet segmentation/reassemble, the ARQ mechanism, etc. The simulator supports all Bluetooth packet types and follows the same master-slave slot structure as in Bluetooth. For the physical layer we employ a simplified analytical model that captures the frequency collision effect of interfering piconets.

In the current simulations the connection establishment procedures, e.g., the inquiry and page procedures are not simulated in detail and we do not consider dynamic scatternet formation either. Instead we perform simulations in static scatternet configurations where the scatternet topology is kept constant during one particular run of simulation.

In the current simulations we run IP directly on top of the Bluetooth link layer and we apply AODV as the routing protocol in the IP layer. The simulator also includes various implementations of the TCP protocol (we employed RenoPlus) and supports different TCP/IP applications, from which we used TCP bulk data transfer in the current simulations.

One of the most important user perceived performance measures is the achieved throughput. We are going to investigate the throughput in case of bulk TCP data transfer and in case of Constant Bit Rate (CBR) sources.

In order to take into account the power consumption of nodes we define *activity ratio* of a node, r_{act} as the fraction of time when the node has been active over the total elapsed time; and *power efficiency*, p_{eff} as the fraction of the number of user bytes successfully communicated (transmitted and received) over the total time the node has been active. The power efficiency shows the number of user bytes that can be communicated by the node during an active period of length 1 sec. Power efficiency can be measured in [kbit/sec], or assuming that being active for 1 sec consumes 1 unit of energy we can get a more straightforward dimension of [kbit/energy unit], which is interpreted as the number of bits that can be transmitted while consuming one unit of energy.

7.2 Network Access Point Scenario

In this scenario we have a NAP that is assumed to be connected to a wired network infrastructure and it provides network access via its Bluetooth radio interface. The NAP acts as a master and up to 7 laptops, all acting as slaves, can connect to the NAP. Furthermore we assume that each laptop has a Bluetooth enabled mouse and each laptop connects to its mouse by forming a new piconet as it is shown in Figure 6.

We simulate a bulk TCP data transfer from the NAP towards each laptop separately. Regarding the traffic generated by the mouse we assume that the mouse produces a 16 byte long packet each 50 ms,

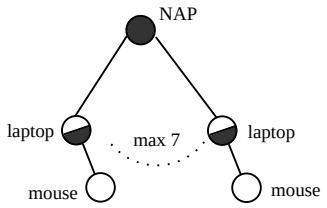

Figure 6: Network Access Point Scenario

periodically. In the NAP-laptop communication we are interested in the achieved throughput while in the laptop-mouse communication we are concerned with the delay perceived by the mouse. In the current scenario we switched off the dynamic checkperiod adjustment capability of the PCSS algorithm and we set the base checking period to 32 frames (40 ms), which is in accordance with the delay requirement of a mouse. Note that this same base checking period is applied also on the NAP-laptop links, although, there is no delay requirement for the TCP traffic. However, the current implementation in the simulator does not yet support the setting of the base checking periods for each link separately. The dynamic checking period adjustment would definitely improve the throughput of NAP-laptop communication as we are going to see later in case of other configurations.

The simulation results are shown in Figure 7. In plot (a) the averaged throughput of NAP-laptop communications are shown in the function of number of laptops for the different algorithms, respectively. Graph (b) plots the sum of the throughputs between the NAP and all laptops. As we expect, the individual laptop throughput decreases as the number of laptops increases. However, it is important to notice that the sum of laptop throughputs do not decrease with increasing number of laptops in case of the PCSS and ICSS algorithms. As the number of laptops increases the efficient coordination becomes more important and the total carried traffic will decrease with the uncoordinated UGSS scheme. The increase of the total throughput in case of the PCSS algorithm is the consequence of the fixed checking intensities, which allocates one half of a laptop capacity to the mouse and the other half to the NAP. In case of small number of laptops this prevents the laptops to fully utilize the NAP capacity, which improves as the number of laptops increases.

The 99% percentile of the delay seen by mouse packets is shown in plot (c). The delay that can be provided with the PCSS algorithm is determined by the base checking period that we use. Recall, that in the current setup the base checking period of the PCSS scheme was set to 32 frames, which implies that the delay has to be in the order of 32 frames, as shown in the figure. The low delay with the UGSS algorithm is due to the continuous switching among the links of a node, which ensures high polling intensity within a piconet and frequent switching between piconets. The UGSS algorithm provides an unnecessarily low delay, which is less than the delay requirement at the expense of higher power consumption.

Plots (d) and (e) show the averaged activity ratio over all laptops and mice, respectively. The considerably higher throughput achieved for small number of laptops by the ICSS scheme explains its higher activity ratio. On graph (f) the averaged power efficiency of laptops is shown, which relates the number of bytes transmitted to the total time of activity. The power efficiency of the PCSS

scheme decreases with increasing number of laptops, which is a consequence of the fixed checking intensities. Since the NAP has to share its capacity among the laptops, with an increasing number of laptops there will be an increasing number of checkpoints where the NAP cannot show up. In such cases the dynamic checking intensity adjustment procedure could help by decreasing checking intensity on the NAP-laptop links. Recall that in the current scenario we employed fixed checking intensities in order to satisfy the mouse delay requirement. It is also important to notice that with the uncoordinated UGSS scheme the activity ratio of a mouse is relatively high, which is an important drawback considering the low power capabilities of such devices.

7.3 Impact of Number of Forwarding Hops

In what follows, we investigate the performance impact of the number of forwarding hops along the communication path in the scatternet configuration shown in Figure 8. The configuration consists of a chain of S/M forwarding nodes (F_i) and a certain number of additional node pairs connected to each forwarding node in order to generate background traffic. The number of S/M forwarding nodes is denoted by N_F. There are N_B number of background node pairs connected to each forwarding node as masters. The background traffic flows from each source node $B_{ij}^{(S)}$ to its destination pair $B_{ij}^{(D)}$ through the corresponding forwarding node F_i. The traffic that we are interested in is a bulk TCP data transfer between node S and D. The background traffic is a CBR source, which generates 512 byte long IP packets with a period of length 0.05 sec.

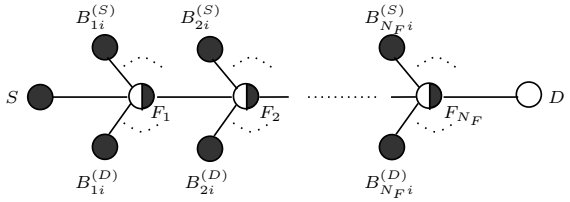

Figure 8: Impact of number of forwarding nodes

During the simulations we vary the number of forwarding hops N_F and the number of background node pairs N_B connected to each forwarding node. As one would expect, with increasing number of forwarding hops and background node pairs the coordinated algorithms will perform significantly better than the one without any coordination (UGSS).

The throughput of the S-D traffic as a function of the number of forwarding nodes (N_F) without background traffic ($N_B = 0$) and with two pairs of background nodes ($N_B = 2$) are shown in Figure 9 (a) and (b), respectively. The throughput in case of no cross traffic drops roughly by half when we introduce the first forwarding node. Adding additional forwarding hops continuously reduces the throughput, however, the decrease at each step is less drastical. We note that in case of the ICSS scheme one would expect that for $N_F > 1$ the throughput should not decrease by adding additional forwarding hops. However, there are a number of other effects besides the number of forwarding hops that decrease the throughput. For instance, with an increasing number of forwarding hops the number of piconets in the same area increases, which, in turn, causes an increasing number of lost packets over the radio interface due to frequency collisions. Furthermore with increasing number of hops the end-to-end delay suffered by the TCP flow increases, which makes the TCP connection less reactive to recover from packet losses.

In the no background traffic case the PCSS scheme performs close to the UGSS algorithm in terms of throughput. However, as we introduce two pairs of background nodes the UGSS algorithm fails completely, while the PCSS scheme still achieves approximately 20 kbit/sec throughput. Furthermore, the power efficiency of the PCSS scheme is an order of magnitude higher than that of the UGSS algorithm in both cases, which indicates that the PCSS algorithm consumes significantly less power to transmit the same amount of data than the UGSS scheme.

7.4 Impact of Bridging Degree

Next we investigate the performance of scheduling algorithms as the number of piconets that a bridging node participates in is increased. The scatternet setup that we consider is shown in Figure 10, where we are interested in the performance of the bridging node C. Node C is an all slave bridging node and it is connected to master nodes P_i, where the number of these master nodes is denoted by N_P. To each master node P_i we connect N_L number of leaf nodes as slaves in order to generate additional background load in the piconets. We introduce bulk TCP data transfer from node C towards each of its master node P_i and CBR background traffic on each $L_{ij} - P_i$ link. The packet generation interval for background sources was set to 0.25 sec, which corresponds to a 16 kbit/sec stream. During the simulation we vary the number of piconets N_P participated by node C and investigate the performance of the PCSS algorithm with and without dynamic checkpoint intensity changes. The number of background nodes N_L connected to each master node P_i was set to $N_L = 3$ and it was kept constant in the simulations.

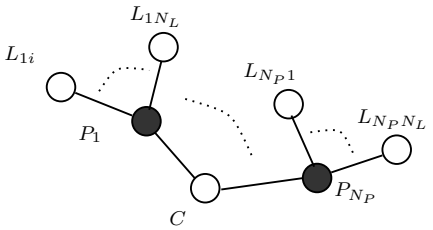

Figure 10: Impact of number of participated piconets

The throughputs of TCP flows between node C and each P_i are averaged and it is shown in Figure 10 (a). The sum of TCP throughputs are plotted in graph (b) and the power efficiency of the central node is shown in graph (c). The PCSS algorithm has been tested both with fixed base checking periods equal to 32 frames ("PCSS-32") and with dynamic checking intensity changes as well ("PCSS-dyn"). The parameter settings of the dynamic case is shown in Table 1.

$q_{uti} = 0.7$	$N_{sample,min} = 4$	
$\rho_{lower} = 0.3$	$\rho_{upper} = 0.7$	
$q_{uti}^{(node)} = 0.7$	$N_{uti,win} = 10$	$\rho_{max}^{(node)} = 0.8$
$T_{min} = 8$	$T_{max} = 256$	

Table 1: Parameter setting of the dynamic PCSS scheme

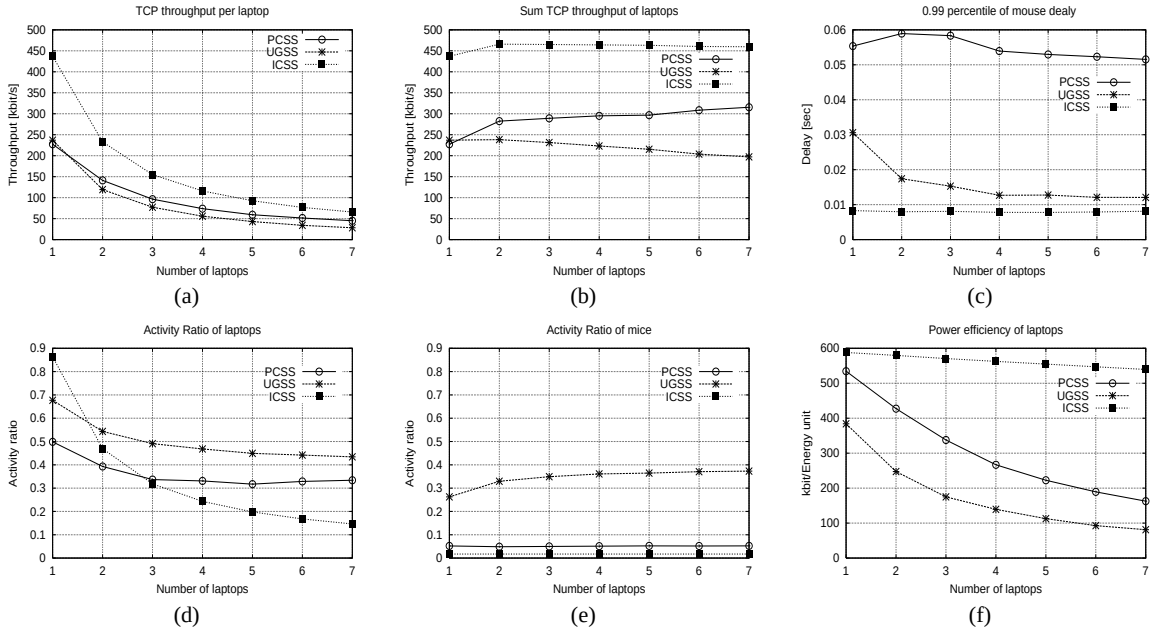

Figure 7: Throughput, delay and power measures in the function of number of laptops connected to the NAP

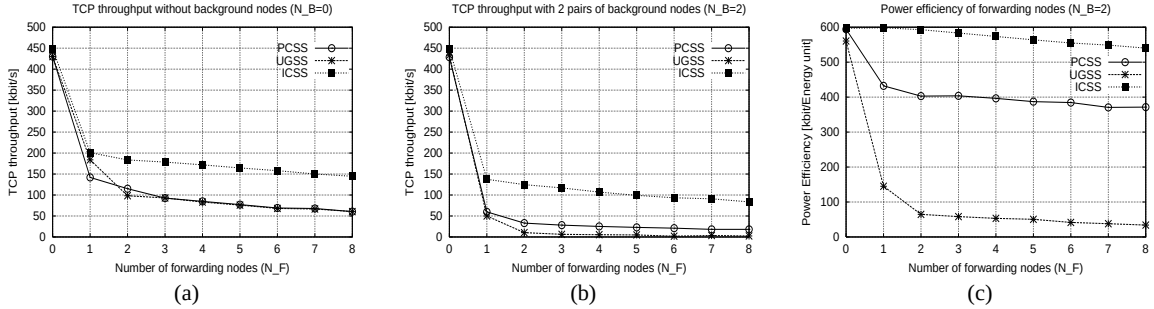

Figure 9: Throughput and power efficiency in function of number of forwarding hops

It is important to notice that the per flow TCP throughputs in case of the dynamic PCSS scheme matches quite closely the throughput achieved by the ICSS algorithm and it significantly exceeds the throughput that has been achieved by the fixed PCSS. This large difference is due to the relatively low background traffic in the neighbouring piconets of node C, in which case the dynamic PCSS automatically reduces checkpoint intensity on the lightly loaded links and allocates more bandwidth to the highly loaded ones by increasing checking intensity.

8. CONCLUSIONS

We have presented Pseudo Random Coordinated Scatternet Scheduling, an algorithm that can efficiently control communication in Bluetooth scatternets without exchange of control information between Bluetooth devices. The algorithm relies on two key components, namely the use of pseudo random sequences of meeting points, that eliminate systematic collisions, and a set of rules that govern the increase and decrease of meeting point intensity without explicit coordination.

We have evaluated the performance of PCSS in a number of simulation scenarios, where we have compared throughput and power measures achieved by PCSS to those achieved by two reference schedulers. The first reference scheduler is an uncoordinated greedy algorithm, while the other is a hypothetical "ideal" scheduler.

In all the scenarios investigated we have found that PCSS achieves higher throughput than the uncoordinated reference algorithm. Moreover, with the traffic dependent meeting point intensity adjustments the throughput and power measures of PCSS quite closely match the results of the "ideal" reference algorithm. At the same time PCSS consumes approximately the same amount of power as the ideal scheduler to achieve the same throughput, which is significantly less than the power consumption of the uncoordinated reference scheduler.

9. REFERENCES

[1] Bluetooth Special Interest Group. *Bluetooth Baseband Specification Version 1.0 B*. http://www.bluetooth.com/.

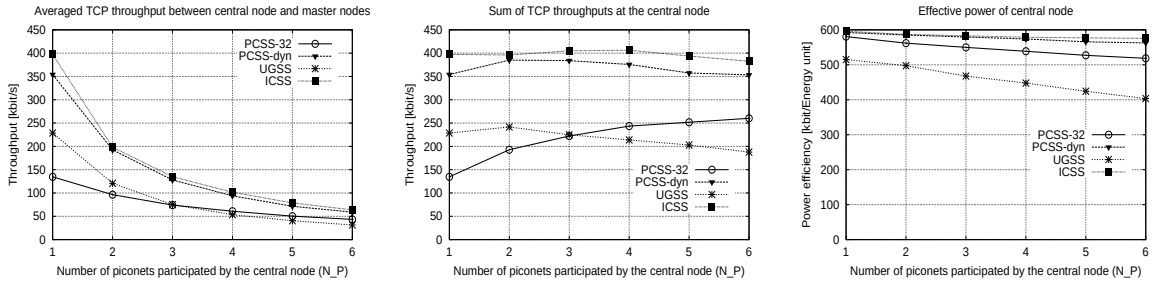

Figure 11: Throughput and power efficiency in function of the bridging degree of node C

[2] Bluetooth Special Interest Group. http://www.bluetooth.com/.

[3] J. Haartsen. BLUETOOTH- the universal radio interface for ad-hoc, wireless connectivity. *Ericsson Review*, (3), 1998.

[4] Z. Haraszti, I. Dahlquist, A. Faragó, and T. Henk. Plasma - an integrated tool for ATM network operation. In *Proc. International Switching Symposium*, 1995.

[5] N. Johansson, U. Körner, and P. Johansson. Performance evaluation of scheduling algorithms for Bluetooth. In *IFIP TC6 WG6.2 Fifth International Conference on Broadband Communications (BC'99)*, Hong Kong, November 1999.

[6] N. Johansson, U. Körner, and L. Tassiulas. A distributed scheduling algorithm for a Bluetooth scatternet. In *Proc. of The Seventeenth International Teletraffic Congress, ITC'17*, Salvador da Bahia, Brazil, September 2001.

[7] P. Johansson, N. Johansson, U. Körner, J. Elgg, and G. Svennarp. Short range radio based ad hoc networking: Performance and properties. In *Proc. of ICC'99*, Vancouver, 1999.

[8] M. Kalia, D. Bansal, and R. Shorey. MAC scheduling and SAR policies for Bluetooth: A master driven TDD pico-cellular wireless system. In *IEEE Mobile Multimedia Communications Conference MOMUC'99*, San Diego, November 1999.

[9] M. Kalia, D. Bansal, and R. Shorey. MAC scheduling policies for power optimization in Bluetooth: A master driven TDD wireless system. In *IEEE Vehicular Technology Conference 2000*, Tokyo, 2000.

[10] M. Kalia, S. Garg, and R. Shorey. Efficient policies for increasing capacity in Bluetooth: An indoor pico-cellular wireless system. In *IEEE Vehicular Technology Conference 2000*, Tokyo, 2000.

APPENDIX

Here, we present the procedure for generating the pseudo random sequence of checkpoints, where we reuse the elements of the pseudo random frequency hop generation procedure available in Bluetooth. The inputs to the checkpoint generation procedure $PseudoChkGen$ are the current checking period $T_{check}^{(i)}$, the Bluetooth MAC address of the slave A_{slave} and the current value of the master's clock $t^{(i)}$. A node can perform checkpoint generation using the $PseudoChkGen$ procedure at any point in time, it is always guaranteed that the position of checkpoint generated by the two nodes will be the same, as it has been pointed out in Section 5.1. Nevertheless the typical case will be that whenever a node arrives to a checkpoint it generates the position of the next checkpoint on the given link. The variable $t_{check}^{(i)}$ always stores the master's clock at the next checkpoint, thus it needs to be updated every time a checkpoint is passed. Here we note that the Bluetooth clock of a device is a 28 bit counter, where the LSB changes at every half slot.

Let us assume that the base period of checkpoints on the i^{th} link of the node is $T_{check}^{(i)} = 2^{j-2}$, $j > 2$ number of frames, which means that there is one pseudo randomly positioned checkpoint in each consecutive time interval of length $T_{check}^{(i)}$ and the j^{th} bit of the Bluetooth clock changes at every $T_{check}^{(i)}$. Upon arrival to a checkpoint the variable $t_{check}^{(i)}$ equals to the current value of the master's clock on that link. After the checkpoint generation procedure has been executed the variable $t_{check}^{(i)}$ will store the master's clock at the time of the next checkpoint on that link.

Before starting the procedure the variable $t_{check}^{(i)}$ is set to the current value of the master's clock $t^{(i)}$ in order to cover the general case when at the time of generating the next checkpoint the value of $t_{check}^{(i)}$ does not necessarily equals to the current value of the master's clock $t^{(i)}$. The position of the next checkpoint is obtained such that the node first adds the current value of $T_{check}^{(i)}$ to the variable $t_{check}^{(i)}$, clears the bits $[j-1, \ldots, 0]$ of $t_{check}^{(i)}$ and then generates the bits $[j-1, \ldots, 2]$ one by one using the procedure $PseudoBitGen(X, W_{ctrl})$. When generating the k^{th} bit $(j-1 \leq k \leq 2)$ the clock bits $X = t_{check}^{(i)}[k+1, \ldots, k+5]$ are fed as inputs to the $PseudoBitGen$ procedure, while the control word W_{ctrl} is derived from $t_{check}^{(i)}$ including the bits already generated and from the MAC address of the slave A_{slave}. The schematic view of generating the clock bits of the next checkpoint is illustrated in Figure 12.

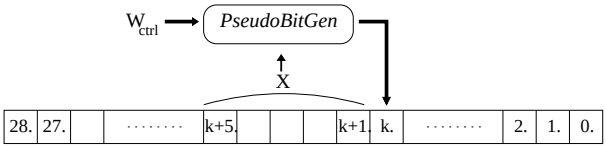

Figure 12: Generating the clock bits of the next checkpoint

The $PseudoBitGen$ procedure is based on the pseudo random scheme used for frequency hop selection in Bluetooth. However, before presenting the $PseudoBitGen$ procedure we give the pseudo-code of the $PseudoChkGen$ procedure.

PseudoChkGen procedure:
$t^{(i)}$: the current value of the master's clock;
$T^{(i)}_{check} = 2^{j-2},\ j > 2$: current length of the base checkperiod in terms of number of frames.

$t^{(i)}_{check} = t^{(i)}$;
$t^{(i)}_{check}[j-1, \ldots, 0] = 0$;
$t^{(i)}_{check} = t^{(i)}_{check} + T^{(i)}_{check}$;
$k = j - 1$;
while ($k \geq 2$)
    $X[0, \ldots, 4] = t^{(i)}_{check}[k+1, \ldots, k+5]$;
    $t^{(i)}_{check}[k] = PseudoBitGen(X, W_{ctrl})$;
    k=k-1;
end

Finally, we discuss the $PseudoBitGen$ procedure, which is illustrated in Figure 13.

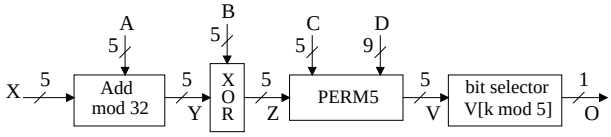

Figure 13: The *PseudoBitGen* procedure

The control words of the $PseudoBitGen$ procedure $W_{ctrl} = \{A, B, C, D\}$ are the same as the control words of the frequency hop selection scheme in Bluetooth and they are shown in Table 2. However, the input X and the additional bit selection operator at the end are different. As it has been discussed above the input X is changing depending on which bit of the checkpoint is going to be generated. When generating the k^{th} clock bit of the next checkpoint the clock bits $X = t^{(i)}_{check}[k+1, \ldots, k+5]$ are fed as inputs and the bit selection operator at the end selects the $(k\ mod\ 5)^{th}$ bit of the 5 bits long output V.

A	$A_{slave}[27-23] \oplus t^{(i)}_{check}[25-21]$
B	$B[0-3] = A_{slave}[22-19]$, $B[4] = 0$
C	$A_{slave}[8,6,4,2,0] \oplus t^{(i)}_{check}[20-16]$
D	$A_{slave}[18-10] \oplus t^{(i)}_{check}[15-7]$

Table 2: Control words

The operation PERM5 is a butterfly permutation, which is the same as in the frequency hop selection scheme of Bluetooth and it is described in Figure 14. Each bit of the control word P is associated with a given bit exchange in the input word. If the given bit of the control word equals to 1 the corresponding bit exchange is performed otherwise skipped. The control word P is obtained from C and D, such that $P[i] = D[i],\ i = 0 \ldots 8$ and $P[j+9] = C[j],\ j = 0 \ldots 4$.

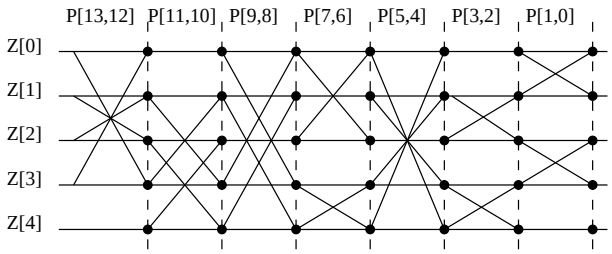

Figure 14: Butterfly permutation