The Big Night Out: Experiences from Tracking Flying Foxes with Delay-Tolerant Wireless Networking

Philipp Sommer, Branislav Kusy, Adam McKeown and Raja Jurdak

Abstract Long-term tracking of small-size animals with wireless sensor networks remains a challenge as only limited energy harvesting and storage is possible due to stringent size and weight constraints for animal collars. We present first experiences towards a perpetual monitoring system for free-living flying foxes. The high mobility of flying foxes requires a delay tolerant wireless network for data gathering: GPS positions and sensor data have to be stored locally until a wireless gateway deployed in bat congregation areas, so called roosting camps, comes within radio range. In this chapter, we present the system architecture and discuss our design decisions towards sustainable and reliable monitoring of flying foxes with a limited energy budget for sensing, storage and communication. Using empirical data from three free-living flying foxes, we characterize the overall system performance in terms of energy consumption and latency.

1 Introduction

The ongoing technological innovation, driven by the explosion of interest in mobile phone technology, has led to ever smaller, less energy demanding, and more accurate sensors and computation devices available on the market. Modern smart phones can

P. Sommer (✉) · B. Kusy · R. Jurdak
Autonomous Systems Lab, CSIRO Computational Informatics,
Brisbane, QLD, Australia
e-mail: philipp.sommer@csiro.au

B. Kusy
e-mail: brano.kusy@csiro.au

A. McKeown
CSIRO Ecosystem Sciences, Cairns, QLD, Australia
e-mail: adam.mckeown@csiro.au

R. Jurdak
e-mail: raja.jurdak@csiro.au

K. Langendoen et al. (eds.), *Real-World Wireless Sensor Networks*,
Lecture Notes in Electrical Engineering 281, DOI: 10.1007/978-3-319-03071-5_2,

localize users with an accuracy of a few meters, thus enabling novel applications that would not have been possible a few years ago. Decreasing form factor, weight, cost and energy consumption have made GPS receivers a versatile research tool enabling novel applications across several domains.

Wireless sensor networks (WSNs) are well suited for wildlife habitat monitoring applications. Sensor nodes, also called *motes*, combine sensing, processing, storage, communication, and energy harvesting capabilities in a small and light-weight package powered by batteries. Researchers typically place motes at specific locations to deliver non-intrusive long-term observation of natural habitats. However, many research questions require tracking the movements of individual animals within a population at high spatial and temporal resolution [3, 5, 9]. Recently, GPS-enabled collars weighting just above 100 g have been used for long-term tracking of Whooping Crane migration in North America [1]. As maximum weight of a collar is usually limited to 5 % of the body weight of the animal, tracking small-size animals remains a challenge.

Application context Flying foxes are mammals that belong to the family of fruit bats (*Pteropididae*). They are common in the tropic and subtropic areas of Asia, Australia and Pacific Islands. They congregate in large numbers of up to several thousands individuals to roost during daytime in so called *camps*. Flying foxes are nocturnally active and leave the camps for foraging from fruit trees. During the nightly foraging, they are able to cover distances up to 100 km over more than 10 h. Monitoring mobility and behavior of individual flying foxes is motivated by the need to better understand this threatened species. Furthermore, flying foxes are attributed to carry diseases, such as the Hendra or Lyssa viruses, which can be transmitted to other animals or humans.

Challenges Developing a hardware and software system to track flying foxes is challenging due to several constraints. First of all, animal ethics requires that the collar accounts for less than 5 % of the animal's body weight. Depending on the type of animal and differences in body weight between male and female individuals, this results in a total weight limit of 30 to 50 g for all electronic components, small solar panels and batteries. Consequently, we are severely limited in terms of available energy resources required to perform sensing, processing, storage and communication tasks. Second, our goal is to track free-living animals, so we will have no physical contact with the collar after the deployment time. Finally, flying foxes leave the camp for a period of more than 10 h every night and can be away for multiple days. Collars need to collect data in a delay-tolerant way and operate for long periods of time during which they cannot communicate with the base station.

Contributions In this work, we present the system architecture and discuss our design decisions to achieve perpetual low-power operation of a tracking device (Sect. 2). First, we have developed *BatMac*, a simple scheduling algorithm for low-power wireless communication tailored to our particular application domain. Specifically, a large number of animals congregate within the relatively small area of a camp during daytime, which allows for offloading previously gathered sensor data to a base station. We use a slotted schedule based on GPS time to desynchronize transmissions

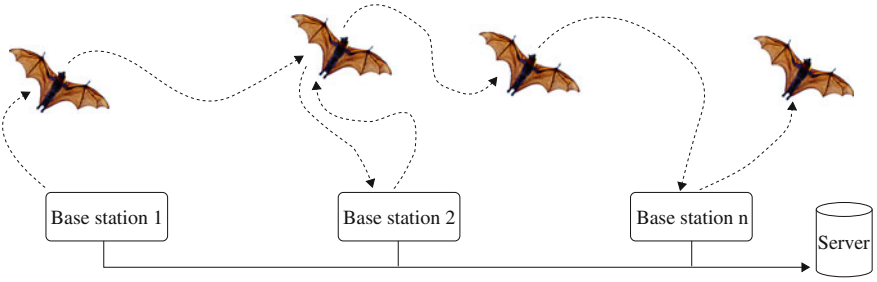

Fig. 1 The network architecture of the Bat Monitoring Project

of individual collars and use a simple radio duty-cycling approach that assumes an always-on base station (Sect. 3). Second, profiling and debugging of wireless sensor nodes on free-living animals paired with the intermittent radio connectivity is a challenging task. We have implemented a scheduler that guards execution of individual sensing tasks and use several mechanisms to improve software and hardware reliability. We also implemented an over-the-air reconfiguration protocol that helps to mitigate the lack of physical access to the device after the deployment time (Sect. 4). Finally, we use empirical data from three flying foxes to demonstrate performance of individual system components in real-world deployments.

2 System Architecture

In this section, we give a brief overview of the system architecture, shown in Fig. 1. Our wireless sensor network consists of three layers: (1) the mobile sensing nodes integrated into an animal collar deployed on flying foxes, (2) the base station layer which consists of several spatially distributed units, and (3) the central database server.

2.1 Mobile Sensing Layer

The purpose of the mobile sensing device is to gather sensor data from an individual collared animal using a variety of sensors (e.g., GPS, accelerometer, pressure sensor). The mobile sensor device is housed inside a collar, which can be attached around the animal's neck by experts trained in handling flying foxes.

In order to meet the stringent constraints in terms of weight, size and power consumption, we decided to build our own printed circuit board (PCB). A detailed description of this board is available in [8]. The software on the mobile sensor node is running a modified version of the Contiki operating system that adds custom extensions for logging and remote procedure calls (RPC) (see Sects. 3 and 4).

2.2 Base Station Layer

Bat roosting camps provide an ideal opportunity for the placement of static infrastructure, so called base stations, as thousands of animals congregate within a relatively small area during daytime. The base station is responsible for downloading sensor data from nearby mobile nodes by using short-range wireless connectivity. We use a gateway node with a TI CC1101 radio connected to an embedded Linux system for control and monitoring of the download operations. In addition, a 2G/3G wireless modem connects the base station to our central server for data uploads. We employ solar panels and batteries to allow autonomous operation in bat camps. Solar energy harvesting is usually a reliable source of power in tropical or subtropical locations with plenty of sunshine, but consecutive days with cloud cover or dense vegetation can limit the amount of solar energy harvested. Consequently, we might only be able to operate the base station during a limited time and have to batch downloading data from animals and uploading to the database.

2.3 Backend Storage and Control Layer

Sensor readings from different mobile nodes are downloaded by spatially separated base stations and transferred to a central database for permanent storage and offline analysis. The database is further responsible to keep a synchronized view of which pages have been already downloaded from nodes. This information is required to avoid duplicate downloads of the same page when the animal is roaming between different camps. Network health data such as battery voltage and number of packets received from different base stations are periodically reported to the database to assist in continuous monitoring and network management.

3 Delay Tolerant Networking for Animal Tracking

Flying foxes are known to cover large distances during nightly foraging and seasonal migrations between different camps. Satellite-based communication systems allow data upload at global scale but pose a significant burden in terms of their cost, size and power consumption. The large spatial coverage of cellular communication networks (e.g., 2G and 3G systems) offers a flexible and cost effective alternative to satellite based systems. However, size and power hinders the integration into collars for small mammals and birds with more stringent constraints. Therefore, we have opted for a low-power, short-range wireless transceiver (TI CC1101) that allows energy efficient operation within unlicensed bands of the frequency spectrum. The disadvantage of our approach is the need to maintain the infrastructure of base stations in flying-fox congregation areas.

Protocols for data collection During the last decade, several data collection protocols for wireless sensor networks have evolved for different application scenarios. Data collection protocols such as CTP [7] and Dozer [2], maintain a routing tree along which packets are forwarded towards the sink node. While both protocols are able to duty-cycle the radio transceivers to save energy, maintaining the network state requires periodic radio beacons. Recently, several communication protocols have been proposed that not need to keep topology-dependent state, such as the Low-Power Wireless Bus [6], which are resilient to high node mobility, but require the nodes to maintain accurate synchronization.

3.1 The BatMac Protocol

Given the uncontrolled mobility patterns of free-living animals and limited energy resources for wireless communication, we decided to implement a novel protocol called *BatMac*. BatMac is a time-synchronized medium access protocol, which is tailored to the intermittent connectivity between mobile nodes and the base station. BatMac is a sender-initiated single-hop protocol implemented on top of Contiki's RIME network stack. It is based on the observation that, unlike homogeneous sensor networks, the distribution of power budgets is highly asymmetric in our network. Mobile nodes can aggressively duty-cycle their radios while the base station operates its radio continuously. Therefore, we do not to use multi-hop communication for packet forwarding, which allows mobile receivers to put their radio in sleep mode for long periods of time.

Slotted communication We use a combination of time-based communication slots and a request-response protocol to avoid interference when multiple mobile nodes are present within communication range of a single base station. Medium access is scheduled using a concept of communication rounds that each consist of several sub-slots. Nodes can access the radio channel exactly once during each communication round, in a sub-slot that is determined by their node ID. For example, for a system with 10 nodes, we might define a communication round to take 5 min and consist of 10 sub-slots. Each of the 10 nodes will then transmit once every 5 min for up to 30 s.

Timings of rounds and sub-slots are determined based on UTC time tracked by the real-time clock of mobile nodes. The real-time clock is periodically re-synchronized on every GPS lock. The mapping between nodes and their corresponding sub-slot is based on their unique identifier. Selection of the round length and the number of sub-slots is a tradeoff between data transmission latency and the maximum number of nodes that we support. As our deployment needs to scale up to 1000 nodes or more, we can assign multiple nodes to the same slot through a modulo operation on their node IDs. We synchronize the real-time clock to within a second to the GPS time, which has the effect of randomizing transmission of beacons within the same sub-slot.

Announcement beacon Each mobile node periodically broadcasts a radio packet containing an announcement beacon at the beginning of its designated communication slot. The announcement contains the node's identifier, application version, and the current flash page number. After sending the radio packet, the node keeps its radio on until a predefined timeout (e.g., 1 s) expires. If the timeout expires, the radio is switched off until the next announcement.

Node selection The base station is continuously listening for incoming announcement beacons from mobile nodes. Upon reception of an announcement, the base station determines if further communication with the node is required, e.g., if new data needs to be downloaded from the flash or the node's configuration should be updated. If further communication is needed, the base station keeps the node's radio awake, by sending a `radio_on(timeout)` command, which will set a new timeout to switch off the radio at the node.

3.2 Data Storage

We are interested in collecting sensor readings from different sensors while the collar is on the animals for several weeks, months or years. Mobility patterns of free-living animals make it very difficult to predict when the animal will be nearby a camp where a base station is deployed. Thus, our software is required to provide persistent storage of sensor data for several hours, days or even weeks. We implemented a first-in first-out data store using the external flash as a circular buffer. Our AT25DF641 flash chip is divided in 32768 pages of 256 bytes each.

The variety of sensors on the mobile device requires a data storage system that is able to handle readings of different payload sizes and at different data rates. For example, a single GPS reading includes values for the timestamp, latitude, longitude, height, speed and an estimation of position accuracy, which results in a total of 15 bytes every second. On the other hand, the combined 3-axis accelerometer and magnetometer sensor generates 12 bytes per reading and can operate at sampling rates up to 100 Hz.

Tagged data format We adapt the Tagged Data Format (TDF) from [4] to pack sensor readings into a byte stream. TDF adds metadata such as the sensor type and timestamp in front of each reading (see Fig. 2). The ID of the sensor type and flags indicating the type of timestamp (relative or absolute) are encoded into a 2-byte header field. Sensor readings associated with an absolute timestamp need 6 bytes to encode the seconds (4 bytes) and millisecond fraction (2 bytes) of the timestamp. If the timestamp of the current reading can be encoded using an offset to the previous timestamp, it is only necessary to store a 2-byte offset. Each sensor type has a specific length for sensor readings, which is fixed and needs to be known to both the encoder and decoder of a TDF stream. To enable decoding of each flash page individually, the first sensor reading always uses an absolute timestamp.

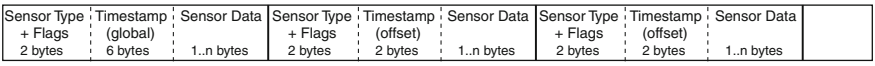

Fig. 2 Storage of sensor readings in flash using the tagged data format (TDF)

Evaluation TDF is a flexible format that provides a compact representation of heterogeneous sensor readings in flash storage. However, the TDF encoder has to jump to the next page if not enough bytes are available in the remainder of the flash page. This fragmentation leads to empty bytes at the end of a page. We characterize the overhead of encoding sensor readings using TDF on sensor data from a collar attached to a free-living flying fox. We downloaded 1147 pages from the flash storage of the node and analyzed 17730 sensor readings encoded in the TDF stream. Our results indicate that the actual sensor data accounts for 65 % of the flash page size of 256 bytes, while headers account for 12 % (sensor type) and 19 % (timestamp). Finally, the overhead due to empty bytes at the end of a flash page accounts for only 4 % of the flash size, which is acceptable given the flexibility that TDF offers for storing heterogeneous sensor data into a continuous flash buffer.

3.3 Data Retrieval

Data downloads are initiated by the base station as a response to an announcement beacon, based on the node's current flash page number contained in the beacon. The download handler runs as a Python script on the embedded Linux machine. We implemented a greedy approach to scheduling downloads from nodes within radio range of a base station. If the current page number in the beacon is higher than the last downloaded page, the base station requests missing pages from the node in sequential order.

Since a full page of 256 bytes would not fit into a single radio packet, we use RPC calls to request chunks of bytes within a specific page from the base. Each RPC command is retransmitted up to 5 times if not acknowledged by the node. If a complete page has been transferred, the base reassembles it and decodes its content using the TDF parser. If the page contains no errors, the base will upload the sensor data to the central database server and mark the corresponding page as complete. If no Internet connection is available at the base station, data will be buffered locally at the base until it can be uploaded to the database.

Data consistency The request/response approach requires the base station to know the latest information from each node that is stored in the database. Since our system architecture includes multiple spatially distributed base nodes, we use a centralized approach to coordinate data downloads across the base stations. Base stations keep track of the most recently downloaded flash page for every mobile node, which allows mobile nodes not to keep track of the download process. Mobile nodes need to simply announce their most recent flash page number and then respond to a base

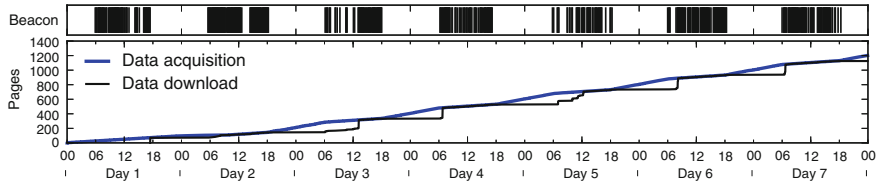

Fig. 3 Timeline of received radio announcement packets at the base station from a single bat (*top*) and the number of generated versus downloaded flash pages (*bottom*)

station's RPC to transmit a specific flash page. Each page will only be downloaded exactly once as long as the state information is synchronized across all base stations. However, this needs to be done only every couple of hours as animals are unlikely to move between the spatially separated base nodes within that time window.

Evaluation We present experimental results using data downloaded from a single wild bat during a 7-day period. Figure 3 shows received announcement packets and number of pages written to flash. In general, we are able to receive announcement beacons from the mobile node just before 6 am in the morning until just after 6 pm in the evening. In the morning, the earliest received beacon was at 5.34 am on the second day while no beacons were received until 6.14 am on Day 4. The last beacon from the animal was received between 5.09 (Day 1) and 6:19 pm (Day 5, 6 and 7). We calculate the packet reception rate (PRR) for announcement beacons as the fraction between received beacons and the number of expected beacons between the first and last received beacon for every day. The range of the observed PRR is between 0.35 (Day 5) and 0.78 (Day 2).

By analyzing the timestamps embedded in the downloaded pages, we are able to track the flash storage consumption over time. Flash pages are written at a lower rate during daytime, as we are mainly logging node health information and a few GPS positions. The data rate increases between 6 pm and 6 am when high-frequency GPS sampling is activated. The latency between data acquisition and download is low during the day since the base station is able to download new pages continuously. Clearly, the latency is higher for data gathered during the night as we have up to 12 h without contact to the base station. Depending on the quality of the radio link in the camp, it might take several hours until the nightly backlog is downloaded (e.g., Day 5).

4 Configuration and Debugging

Development of hardware and software for animal tracking is challenging due to the mobility of the animals. While it is relatively easy to follow the path of collared livestock, catching and collaring of free-living animals such as flying foxes is labor intensive and notoriously difficult. Large nets mounted between high poles are required to catch animals while they are flying back in or out of the camp during

the night. Catching a collared animal a second time is almost impossible given the large number of individuals populating a camp. Therefore, our development approach assumes that it is not possible to gain physical access to the mobile node ever again after the initial deployment.

4.1 Remote Task Configuration

Several mechanisms for over-the-air code distribution in wireless sensor networks have been proposed in the literature. However, supporting wireless reprogramming increases the complexity of the code running on the node and requires dedicated storage for new and fallback images. Furthermore, any failure during the reprogramming process might leave the node in a defective state. Therefore, we decided not to implement over-the-air reprogramming for our mobile nodes. Instead, we integrated methods to support wireless reconfiguration for a set of well-tested tasks within our application. Each task is associated with a Contiki process that implements a specific sensing task (e.g., getting several GPS fixes, or measuring the battery voltage). Tasks can be limited to a specific time interval (start, stop), periodicity within that time interval, and minimum battery voltage. Tasks can also have additional arguments which are specific to a sensor (e.g., number of samples). The task scheduler is executed once every second to start or stop tasks according to the current configuration.

Reconfiguration Task configurations use a dedicated part of the flash for persistent storage. We provide remote procedure calls (RPC) sent over the radio to view, update and delete a task on the sensor node.

4.2 Remote Debugging

We implemented two methods for debugging mobile nodes deployed on animals: Node inspection by remote procedure calls sent over the radio, and logging of debug output to the flash storage as part of TDF data. In addition, we use a combination of hardware and software based mechanisms to recover from error conditions. In the remainder of this section, we describe the debugging capabilities integrated with our application and highlight how their usefulness for debugging in practice.

Node inspection We implemented several helper methods for debugging and inspection of the current node state. Thereby, we do not want to halt code execution on the node, but rather acquire a snapshot of the node's status, e.g., the current value of a variable in RAM. Furthermore, custom RPC methods allow us to reboot the node, sample the battery voltage, read data from the external flash, and read/modify/delete tasks.

Debug instrumentation While RPC methods are useful to inspect the node status when radio connectivity is available, little information is available during periods

with no radio connectivity. Therefore, we implemented two additional sensing tasks useful to aid the process of debugging. The *POWER* task will periodically sample the battery voltage, solar panel voltage, and charge current and write the ADC readings to the flash storage using the TDF logging abstraction. In addition, the *DEBUG* task periodically logs the reason for the last microcontroller reset and the current uptime in seconds, allowing us to track node reboots and determine their cause retrospectively.

Watchdog and Grenade timers We combine a hardware watchdog and a software timer to recover from potential problems that could cause the software running on the microcontroller to get stuck. The hardware watchdog of the MSP430 will trigger a reboot if the watchdog timer is not being reset within 2 s. Therefore, we instrument Contiki's main scheduler loop to reset the watchdog periodically. This mechanism will protect us from possible software errors within the implementation of our Contiki processes such as infinite loops. A so called *grenade timer* is used as an additional layer of protection against hardware and software problems. Thereby, we force a graceful node reboot when the uptime counter reaches a predefined value (e.g., once a day). Grenade reboots can easily be distinguished from uncontrolled reboots by looking both at the reset reason and the value of the uptime counter logged to flash.

5 Experimental Data: A Day in the Life of a Flying Fox

We present preliminary experimental results from two mobile nodes attached to free-living flying foxes. Both animals left the camp during the night and returned at dawn. We configured two different tasks for GPS sampling to gather the position of the mobile node depending on the current time of day. The low-frequency GPS task will log the current location every 3 h during the day. During the night, the high-frequency GPS task logs locations every 10 min.

5.1 GPS Tracking

The interval between startup of the GPS receiver until the first GPS position is available is commonly denoted as the *Time to First Fix (TTFF)*. During a *coldstart*, the GPS receiver has to search for a signal from several GPS satellites to determine its current position. After the first valid position has been calculated, the receiver keeps tracking the existing satellites in order to periodically update its estimation of the current location. We use the sleep mode of the u-blox MAX6 module, which retains real-time clock and satellite orbit information while the main part of the receiver is put into a low-power mode. A GPS *hotstart* is possible if the receiver wakes up from sleep mode and has up-to-date satellite information to calculate the current position, otherwise, it has to go through a coldstart phase again. We collected TTFF values of 1103 successful GPS location requests from two nodes. Reported values for TTFF

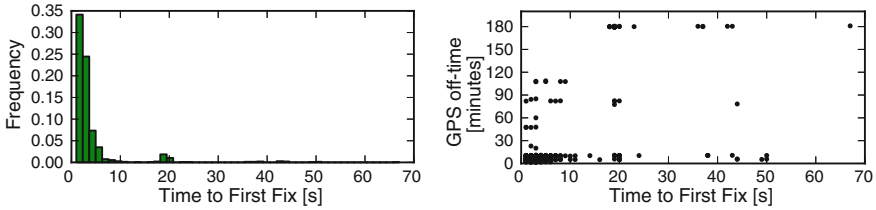

Fig. 4 Distribution of the time to first GPS fix (*left*) and the relation between the preceding GPS off interval and the time to first fix (*right*)

are between 1 and 67 s while it takes 4.17 s to get the first valid position on average (see Fig. 4). We further observe that it takes longer to get the first fix when the GPS was inactive for a longer period of time.

5.2 Power Management

The amount of remaining energy stored in the rechargeable battery depends on several factors, such as the amount of solar energy harvested and the current draw during recent days. We periodically measure the battery voltage, the solar panel voltage and the charge current into the battery using the on-board ADC on the sensor node. While we aim to derive an accurate estimation for the remaining energy based on measurements in future versions, the task scheduler currently only uses the battery voltage when deciding whether to execute a task. Power measurements and corresponding GPS task execution times are reported in Fig. 5. We measure similar values for the charge current for both nodes. We see considerable differences between the two mobile devices, although the animals are roosting in the same camp. Node A maintains a relatively stable battery voltage across the whole measurement period while the battery voltage of Node B decreases rapidly during the night time. As a result of the low voltage, Node B stops gathering GPS samples as its battery drops below the threshold voltage. We can also see a significant difference between rainy weather (Day 1) and sunny weather (Days 2 to 5). Furthermore, we continually increase the duty-cycle of the GPS task during the night. Starting from fixes every 10 min between 6 pm and 6 am (Day 1/2), we extended the window from 5.30 pm to 6 am to capture the departure of the animals from the camp (Day 3/4). Finally, we reconfigured the GPS task to gather continuous GPS fixes at a frequency of 1 Hz between 5.35 and 6 pm on the evening of Day 5. Figure 6 shows the positions of the animals on a map.

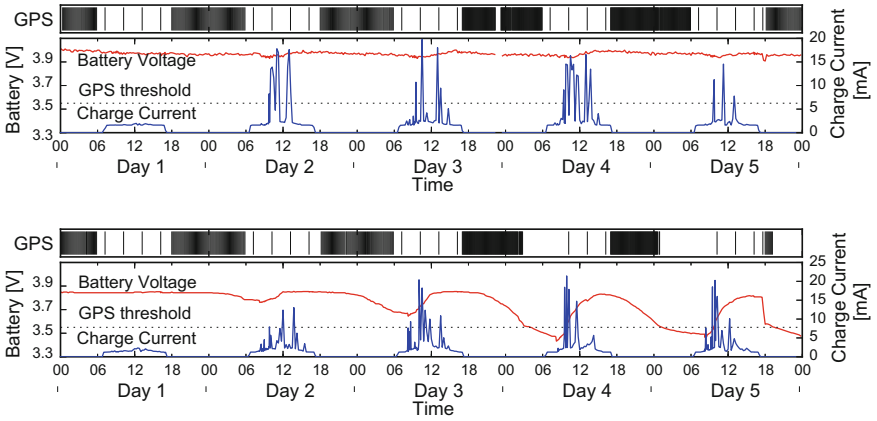

Fig. 5 Battery voltage, charge current and activation of the GPS task for Node A (*top*) and B (*bottom*)

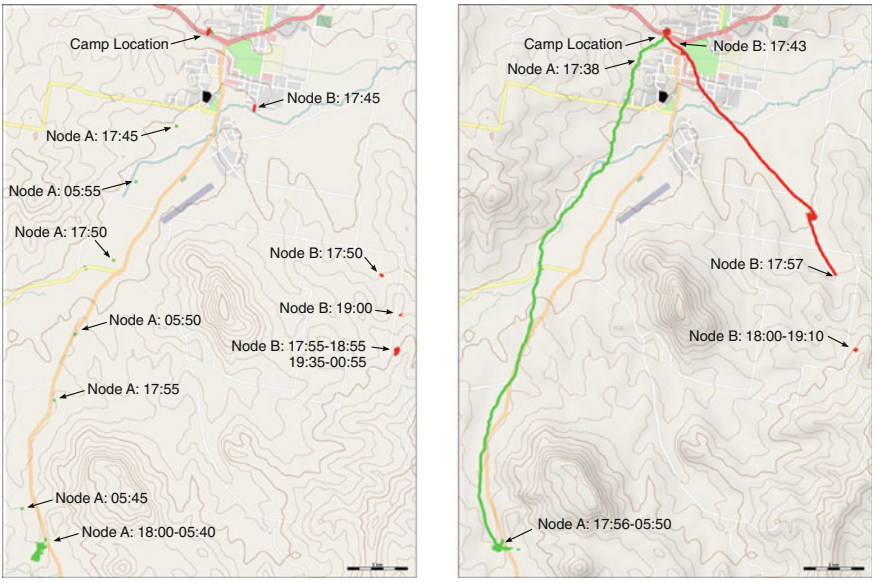

Fig. 6 GPS waypoints of two mobile nodes: The GPS was configured to get fixes every 10 min during night time on Day 4 (*left*). Continuous sampling allows to track the accurate flight path when animals leave the camp on Day 5 (*right*)

6 Lessons Learned

In this chapter, we have presented our first experience with a novel hardware and software architecture for monitoring flying foxes using mobile sensor nodes. Although this project is only at an early stage and deployment of several hundred animal

collars is planned over the next months, we have gathered a large set of empirical data, which allows us to verify the correctness of system operations and provides invaluable information for fine-tuning our system in future deployments.

We believe that staggered releases of new software features to a small number of nodes can mitigate the impact of possible software problems instead of rolling out several hundred nodes simultaneously. However, this requires co-existence of different software versions, as collars with older software versions still remain active. Thus, we assign a software version to each node which is also included as part of the announcement beacon, enabling to identify the capabilities of different nodes. Furthermore, diagnostic tools built on top of remote procedure calls enable flexible inspection of nodes within range.

Logging state information to persistent storage is vital for debugging a system with intermittent network connectivity. The over-the-air reconfiguration of tasks provides control over the amount of debug output during different stages of the deployment, thus providing code instrumentation only when needed.

Acknowledgments We would like to thank Ben Mackey, Philip Valencia, Chris Crossman, Luke Hovington, Les Overs and Stephen Brosnan for their contributions to this project.

References

1. Anthony, D., Bennett, W., Vuran, M., Dwyer, M., Elbaum, S., Lacy, A., Engels, M., Wehtje, W.: Sensing through the continent. In: ACM/IEEE IPSN (2012)
2. Burri, N., von Rickenbach, P., Wattenhofer, R.: Dozer: Ultra-low power data gathering in sensor networks. In: ACM/IEEE IPSN (2007)
3. Butler, Z.: From robots to animals: virtual fences for controlling cattle. J. Robot. Res. **25**, 5–6 (2006)
4. Corke, P., Wark, T., Jurdak, R., Moore, D., Valencia, P.: Environmental wireless sensor networks. Proc. IEEE, **98**(11), (2010)
5. Dyo, V., et al.: WILDSENSING: design and deployment of a sustainable sensor network for wildlife monitoring. ACM Trans. Sens. Netw.**8**(4), (2012)
6. Ferrari, F., Zimmerling, M., Mottola, L., Thiele, L.: Low-power wireless bus. In: ACM SenSys, (2012)
7. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection Tree Protocol. In: ACM SenSys, (2009)
8. Jurdak, R., Sommer, P., Kusy, B., Kottege, N., Crossman, C., Mckeown, A., Westcott, D.: Camazotz: multimodal activity-based GPS sampling. In: ACM/IEEE IPSN, (2013)
9. Zhang, P., Sadler, C.M., Lyon, S.A., Martonosi, M.: Hardware design experiences in ZebraNet. In: ACM SenSys (2004)

http://www.springer.com/978-3-319-03070-8

Real-World Wireless Sensor Networks
Proceedings of the 5th International Workshop,
REALWSN 2013, Como (Italy), September 19-20, 2013
Langendoen, K.; Hu, W.; Ferrari, F.; Zimmerling, M.;
Mottola, L. (Eds.)
2014, XII, 261 p. 126 illus., Hardcover
ISBN: 978-3-319-03070-8