Title

TempMesh – A Flexible Wireless Sensor Network for Monitoring River Temperatures

Journal

ACM Transactions on Sensor Networks, 19(1)

ISSN

1550-4859

Authors

Burman, Scott G
Gao, Jingya
Pasternack, Gregory B
et al.

Publication Date

2023-02-28

DOI

10.1145/3542697

TempMesh – A Flexible Wireless Sensor Network for Monitoring River Temperatures

SCOTT G. BURMAN, JINGYA GAO, GREGORY B. PASTERNACK, and
NANN A. FANGUE, University of California, Davis
PAUL CADRETT, U.S. Fish and Wildlife Service
ELIZABETH CAMPBELL, U.S. Army Corps of Engineers
DIPAK GHOSAL, University of California, Davis

For a Chinook salmon restoration project in the lower Yuba River in California, we designed and deployed a wireless sensor network to monitor river temperatures at micro-habitat scales. The study required that temperatures be measured along a 3 km study reach, across the channel, and into off-channel areas. To capture diel and seasonal fluctuations, sensors were sampled quarter-hourly for the full duration of the six-month juvenile salmon winter residency. This sampling duration required that nodes minimize power-use. We adopted event-based software on MSP430 micro-controllers with 433 MHz radio and minimized the networking duty-cycle. To address link failures, we included network storage. As the network lacked real-time clocks, data were timestamped at the destination. This, coupled with the storage, yielded timestamp inaccuracies, which we re-aligned using a novel algorithm. We collected over six months of temperature data from 35 sensors across seven nodes. Of the packets collected, we identified 21% as being incorrectly timestamped and were able to re-align 41% of these incorrectly timestamped packets. We collected temperature data through major floods, and the network uploaded data until the flood destroyed the sensors. The network met an important need in ecological sampling with ultra-low power (multi-year battery life) and low-throughput.

CCS Concepts: • **General and reference → Empirical studies**; • **Hardware → Sensor applications and deployments**; • **Computer systems organization → Sensor networks**; • **Networks → Network protocol design**; **Sensor networks;**

Additional Key Words and Phrases: River temperature monitoring, wireless sensor network, fluvial temperatures, network storage, power efficiency, timestamp alignment, sensor deployment, fish, salmon

Scott G. Burman and Jingya Gao contributed equally to this research.

This work was supported by a cooperative agreement (F16AC00735) with the Anadromous Fisheries Restoration Program within the US Fish and Wildlife Service. The USDA National Institute of Food and Agriculture support Greg Pasternack (Hatch: CA-D-LAW-7034-H) and Nann Fangue (Hatch: CA-D-WFB-2098-H).

Authors' addresses: S. G. Burman (corresponding author), J. Gao, G. B. Pasternack, N. A. Fangue, and D. Ghosal, University of California, Davis, 1 Shields Avenue, Davis, CA, 95616; emails: {sgburman, aligao, gpast, nafangue, dghosal}@ucdavis.edu; P. Cadrett, U.S. Fish and Wildlife Service, 850 S Guild Ave Suite 105, Lodi, CA, 95240; email: paul_cadrett@fws.gov; E. Campbell, U.S. Army Corps of Engineers, San Francisco District, 450 Golden Gate Avenue, 4th Floor, San Francisco, CA, 94102; email: elizabeth.a.campbell@usace.army.mil.

1550-4859/2022/12-ART15
https://doi.org/10.1145/3542697

ACM Reference format:
Scott G. Burman, Jingya Gao, Gregory B. Pasternack, Nann A. Fangue, Paul Cadrett, Elizabeth Campbell, and Dipak Ghosal. 2022. TempMesh – A Flexible Wireless Sensor Network for Monitoring River Temperatures. *ACM Trans. Sensor Netw.* 19, 1, Article 15 (December 2022), 28 pages.
https://doi.org/10.1145/3542697

1 INTRODUCTION

Anadromous fish like salmon swim from oceans into rivers where they spawn. While their time in rivers is much shorter than their oceanic life-stages, the in-river stages are extremely energetically costly: Adults swim for miles against currents (without feeding), generate gametes, and then distribute them by spawning. Then, the next generation's juveniles hatch, grow in size by orders of magnitude, avoid predators, find and consume food, and finally smoltify—which is a physiological change from juvenile parr to smolt, to prepare for downstream migration. These anadromous fish must be adapted to the oceans where they live, which are fairly homogeneous in temperature. At the same time, they must retain (and pass on to their progeny) the ability to expend considerable energy and thrive in rivers with unique temperature profiles. To reach our study system, the **lower Yuba River (LYR)**, the fish must first enter the San Francisco Bay, move up the Sacramento River and then into the Feather River, migrate up the Feather River, then move into the Yuba River. This migration is a particular challenge, as the Feather River is considerably warmer than the very cold Yuba River. This study system is therefore ideal for addressing questions about thermal tolerances of salmonids. The diversity of temperatures these fish traverse is extreme, and we sought insights into their adaptation to the different rivers that they traverse to reach spawning sites, and where juveniles grow and smoltify. Beyond these adaptive or evolutionary energetic impacts of temperature, we endeavored to place river temperature into an ecological context by assessing temperatures in micro-habitats. Micro-habitats in this context are at scales below the river-reach, which required sensors dispersed both across the channel and longitudinally along the length of the river.

Fluvial river temperatures are dynamic and complex. Two trends tend to dominate river temperatures: They generally increase moving downstream from headwaters to mouth [1] and are related-to (but not forced by) air-temperatures [2]. Yet, these trends vary both seasonally and daily [3]. Solar warming is a strong predictor of river temperature [4], and as a result, vegetative shading can have intense local effects on micro-habitat temperatures [5]. Finally, flows through sediments, which are shaded from the sun and have no convective warming from air can maintain relatively consistent temperatures over vast distances [6]. Integrating these various inputs makes temperature predictions in rivers challenging. Modeling river temperatures was found to be feasible, but computationally difficult, particularly at fine-scales. Physics-based models were less effective than interpolative models [7, 8].

We found that existing sensing methodologies (FLIR [9] and fiber-optics [10]) could not meet the (spatial and temporal) sampling density and extent that we required to characterize juvenile micro-habitats and instead opted to build a wireless sensor network to collect real-time temperature data.

Wireless sensor networks (WSNs) [11, 12] have been used in monitoring applications in similarly harsh applications, including in the early detection of forest fires [13], in actuating applications such as precision agriculture [14], to measure and control energy usage in smart homes [15, 16], for high-resolution spatio-temporal monitoring of underwater environments [17], and in tracking applications such as animal telemetry [18]. In a typical scenario, sensors are distributed over a geographical area and the data from these sensors are—using a network of relay nodes—aggregated at a gateway node, which in turn uploads the data to a server for processing and analysis.

Multiple environmental factors influenced the design of the sensors and the wireless mesh network that we created. First, the temperature sensors needed to be able to survive changing flows and river conditions. Second, the river on which sensors were installed was difficult to access and there was no access to reliable power. Consequently, sensors and wireless networking nodes were battery-powered. Once deployed, changing batteries at remote node locations was impractical and costly. It was therefore imperative that we optimize the software to minimize power use. There were also challenges to consider when placing nodes: topography and geography limited the places where sensors and nodes could be placed. Finally, hikers, poachers, and animals (beavers and bears) damaged equipment. These constraints limited the density and positioning of relay nodes, which meant that they were spread over larger distances, which rendered link quality more susceptible to failure during inclement weather. To solve these issues, we introduced a network layer function to store data in intermediate nodes until network links were re-established.

This is a **practical implementation paper**, in which we describe the design, implementation, and deployment of a wireless mesh network to gather spatial and temporal river-temperature data from the lower Yuba River at scales appropriate to assess juvenile salmonid micro-habitat. The key contributions of this article include:

(1) Design considerations of the data transfer protocol and network architecture, which include a network storage function robust to intermittent link failure;
(2) Node software architecture based upon event driven proto-threads, designed explicitly to minimize power use;
(3) Experimental results that quantify the node power use and an evaluation of the efficacy of the architecture and implemented low power modes;
(4) A methodology for correcting timestamps of data with an incremental identifier, collected via a network with unknown transmission delays.

The remainder of this article is organized as follows: In Section 3, we discuss some of the technological choices we made and the constraints imposed by these choices. In Section 4, we describe the network architecture and the data transfer protocol. We present how the network storage function was implemented to account for intermittent link failure. In Section 5, we describe the software architecture of the node based on proto-threads and the methods that were implemented to minimize power. In Section 6, we present the field results and discuss the important issue of timestamp alignment. In Section 7, we characterize what we have learned from this work and how it contributes to *in situ* environmental sensing more broadly. In Section 8, we discuss the related work, and in Section 9, we give the conclusions and discuss the future work.

2 BACKGROUND AND MOTIVATION

In 2014, river managers and experts working on the Yuba Accord Management Team for the lower Yuba River in California's Central Valley sought to enhance Chinook salmon populations by targeting the juvenile life-stage for in-river restoration efforts. There was hope that restoration work could modify the river to better meet the energetic needs of juvenile Chinook salmon, which were historically abundant in the Yuba River [19]. Energy balance is dictated by exertion and consumption. To improve the available habitat for the energetics of an organism, the system must either have more abundant or more nutritious food (higher consumption), or require that the organism use less energy in the system (less exertion). Energy use of salmon is largely dependent upon water temperature. To find the optimal temperatures for salmonids, laboratory experiments have considered the relationship between water temperature and aerobic scope (the difference in the fish's energy-use at maximum exertion and at rest). These studies found that salmonids were adapted to the temperatures they experienced in their home rivers [20, 21].

We determined that to meaningfully sample the temperature experienced by juvenile salmonids in the lower Yuba River, we would need hourly temperature data for a 3 km reach of this multi-threaded river. To capture the full juvenile life-stage, this sampling needed to survive in-river for at least six months. We also knew that sensors needed to be able to record and report data even when we could not physically access the river during flood events. Monitoring needed to be highly resolved, both in space and time. Specifically, it was important that the temperature be monitored both longitudinally (along the river, with sensors spaced hundreds of meters apart) and across the channel (with sensors spaced a few meters apart). We opted to sample every 15 minutes to provide multiple samples per hour.

3 TECHNOLOGY

3.1 Temperature Sensing

There exist a few sensing technologies capable of the type of data collection required for the project. **Forward-looking infrared (FLIR)** cameras can be used to image temperatures at the necessary spatial density, either aerially or from channel margins. Unfortunately, achieving the required spatial extent was infeasible, as it would require that we install dozens of FLIR cameras or move them constantly. They also can only image the water's surface [9]. FLIR deployments also fail to meet the temporal extent needed for this study, as they would require considerable maintenance and power over the course of the six-month juvenile residency period.

Fiber-optic networks can sample large distances (~13 km) at a time, with about 1 meter spatial resolution [10]. As a result, fiber optic temperature sensing met the necessary spatial density and extent for our work, with the fiber optics repeatedly crossing the river in a zig-zag pattern. Unfortunately, fiber-optic temperature sensing was not feasible for this study due to the required lasers, which use considerable amounts of power and the need for continuous, on-going upkeep and maintenance. This was impractical given the access issues of the study sites and the sampling duration.

It became clear that the best option for temperature monitoring was the use of electronic temperature sensors. We initially considered data-logging sensors, but moved away from these because the lower Yuba River experiences extreme flood events that can transport trees, large boulders, and other detritus down-river. These events can damage, bury, or dislodge temperature sensors. Since there is no way to remotely recover data from low-cost *in situ* logging sensors, destructive events eliminate all data since the previous download, and data from flood events is lost.

Ultimately, we opted to construct a wireless sensor network that sensed data using off-the-shelf temperature sensors. We selected the Campbell Scientific CS225-L SDI-12 temperature sensor cables, which were environmentally sealed and protected all sensor hardware by housing it in a thick, sealed cable assembly. These assemblies included individually addressable, digital temperature sensors, allowing for long cable runs without concern about voltage loss or other drift that could alter sensor readings. These sensors were sufficiently accurate for the study ($\pm 0.2^oC$). Sensor assemblies were designed to contain five sensors, evenly spaced either 5 m or 10 m apart along the length of the cable, depending upon the river width at a particular location. Sensor cables also had between 10 m and 100 m of additional cabling on the end to allow for transmitters to be concealed in riparian foliage. These sensors were tethered to anchors buried in the riverbed. Sensors were initially designed to withstand large changes in the flow of the river, with sacrificial anchors, which would detach under destructive flows and allow the sensor cables to survive. Despite these precautions, it proved nearly impossible to design the sensor systems to survive impacts from massive detritus such as logs carried down by the river during the highest of flows.

3.2 Wireless Mesh Network

The Campbell Scientific CS225-L temperature sensors were integrated with a wireless mesh network built using inexpensive, low-power devices. Each device was equipped with a radio with reasonable range to aggregate the temperature data from the sensor strings to one aggregation point from which the data could be back-hauled to a server on the University of California, Davis, campus using a cellular connection.

There are four main technologies for implementing wireless mesh networks [22], based on (1) IEEE 802.11s, (2) **Bluetooth Low Energy (BLE)**, (3) IEEE 802.15.4, and (4) LoRa (and the corresponding MAC layer protocol, LoRaWAN). The key advantage of devices built using these technologies is that they come with protocols for node discovery, topology formation, and data forwarding and routing. These technologies also have high bandwidth. The range for BLE-based devices is about 30 m. While BLE's relatively high bandwidth at this range is optimal for its intended application, it was infeasible for the present study, as 30 m node ranging was inadequate and would have required far more mesh nodes than could be installed in the given terrain. IEEE 802.11s [23], IEEE 802.15.4 [24], and LoRa [25] are designed for **Internet of Things (IOT)** applications in industrial setting or Smart City applications where either power is readily available or batteries can be easily recharged or replaced. These technologies were not designed to minimize power use. We wanted to use 433 MHz band because of its superior propagation characteristics outdoors compared to 2.4 GHz or 900 MHz, despite the negative effects of having a larger Fresnel zone. As a result, the 433 MHz band has gained momentum for machine-to-machine communications using low-power wireless technologies [26]. DASH7 and IEEE 802.15.4f are two standards that have been developed for the 433 MHz band. At the time of the deployment, LoRa had not yet added support for the 433 MHz band. Our application required reliable data transmission in an energy-deficient environment. We prioritized power savings and data fidelity over bandwidth, latency or other considerations, because the raw temperature data occupied only a few bytes, which meant each sensor node could generate small packets every 15 minutes.

We opted to use wireless communication operating at 433 MHz, because the frequency struck a reasonable balance between range and penetration. It also was an unregulated frequency band—requiring only a **Federal Communications Commission (FCC)** experimental license. Using low-power transmitters like ours, operating at 433 MHz, 500 m range was achievable with simple, low-cost whip antennae. At this range, 433 MHz transmissions can penetrate vegetation while reflecting off of solid surfaces. In a river, the steep embankments leading up to the floodplains and levees can create something of an echo chamber, which we found to enhance transmission ranges.

3.3 Hardware

The nodes of the wireless mesh network were implemented using WizziMotes [27], an IoT device built on top of the **Texas Instruments (TI)** CC4305137, an MSP430-based **micro-controller unit (MCU)** with integrated 433 MHz wireless networking. The MSP430 family of MCUs are ideal for this application, as they are low-power and can be switched to ultra-low power modes, which significantly extend battery life [28, 29]. We used WizziMote because they were among the only devices available at the time that operated in the 433 MHz band and implemented some aspects of the DASH7 protocol [27, 30].

As a **System-on-Chip (SOC)**, MSP430 provides integrated peripherals for a variety of battery-operated wireless applications. The operating modes take into account three different needs: ultra-low power, speed and data throughput, and minimization of individual peripheral power consumption [28, 29]. Built with **Low Power Modes** (**LPMs**; LPM0: 80μA, LPM2: 6.5μA, LPM3: 2μA, LPM4: 1μA), MSP430 can preserve energy by shutting down respective clocks on the processing chip and

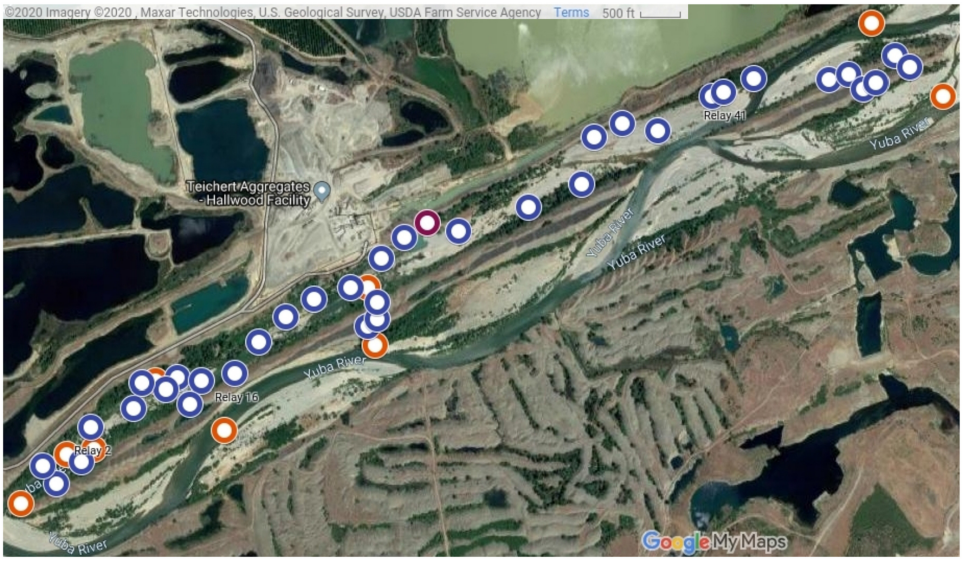

Fig. 1. Map of the network deployed in November 2018, which remained functional through May 2019 on the 3 km study reach of the lower Yuba River, immediately downstream from Daguerre Point Dam (not featured: just upstream of top-right of map) near Marysville, in California's Central Valley. The orange nodes are the sensor nodes, the blue nodes are the relay nodes, and the magenta node is the gateway node. Map generated using Google My Maps, and is 3.7 km by 2.1 km.

later returning to **Active Mode** (**AM**: 160μA, radio in RX: 15mA, radio in TX: ~30mA depending on transmit power) through enabled interrupts in less than 6μs [28, 29]. The state of execution is saved on the stack and is restored unless altered during the interrupt service routine. State and memory are maintained and, as a result, the network is able to store data while nodes are put to sleep. This was an important requirement given that we had intermittent link failures.

The network protocol stack of WizziMote is built around the DASH7 standard [30]. The data communication is simplex, i.e., the network interface can only operate as a transmitter or as a receiver at one time. The radio channel bandwidth is 1.74 MHz, spanning from 433.05 to 434.79 MHz. While DASH7 offers different classes of communication channels, in this project, we used the normal category [30]. It provides eight channels, each with a data-rate of 55 Kbps. In our experimental analysis with the WizziMote, the neighboring channel interference was high. As a result, we used only four non-overlapping channels.

4 NETWORK ARCHITECTURE AND PROTOCOLS

The wireless mesh network contained three types of nodes; **sensor nodes** that measured water temperatures, assembled the data into packets, and transmitted the packets to a backbone of **relay nodes**, which in turn moved the data packets to a **gateway node**, which uploaded the data to a server.

(1) **Sensor Nodes** *(Figure 1: orange, Figure 2(a))* interfaced with a string of five Campbell CS225-L temperature sensors, spaced either 5 or 10 m apart on the length of the sensor cable and anchored into the river bed. These nodes contained WizziMotes that connected to the wireless network and were the sources of the temperature data. At the heart of these nodes was an Arduino Mini Pro, which periodically polled sensors for the temperature data, formatted

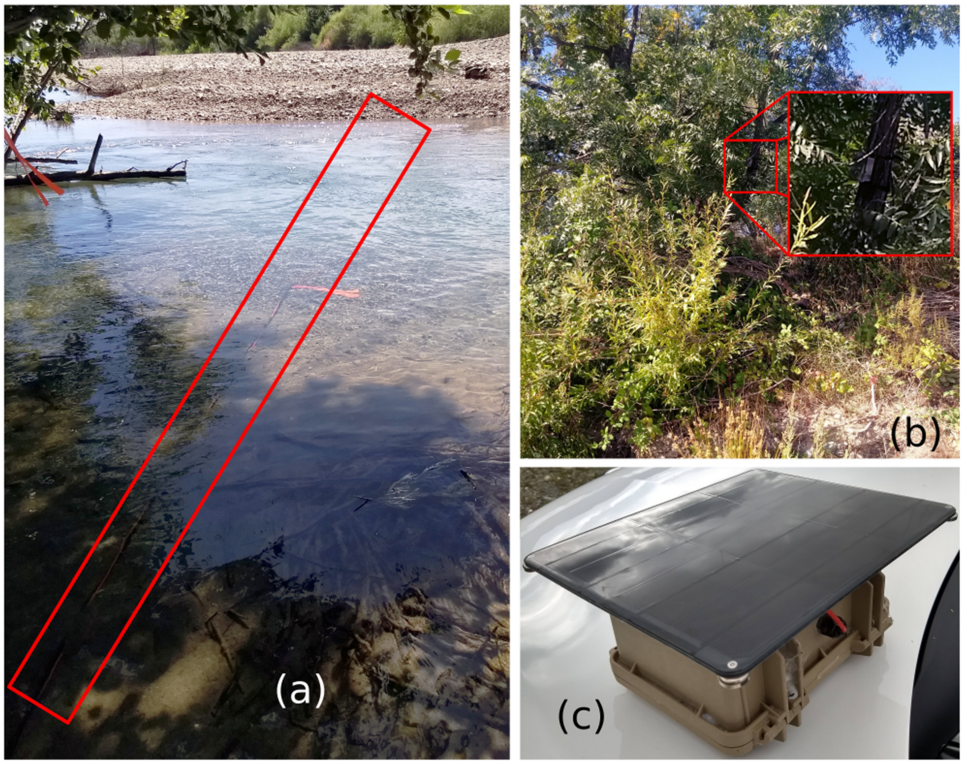

Fig. 2. Pictures illustrating the disparate nodes installed in the lower Yuba River, including a sensor node (a - red box indicates the path of the sensor), a relay node (b - inset of the relay in red), and the gateway (c).

the data into a packet, and passed the packet over a serial interface to the WizziMote that then transmitted the packet to a relay node in the wireless network. While our deployment had up to nine sensor nodes (this number fluctuated with damage and fouling by wildlife, but seven remained active through most of the sampling period), the network could handle many more sensor nodes. Each of our sensor nodes were equipped with five sensors anchored to the riverbed. The placement of sensors was based upon two criteria: (a) interest or importance of micro-habitat for juvenile salmonids and (b) feasibility of installation. Installed sensors both bisected major channels, and zig-zagged through off-channel micro-habitats.

(2) **Relay Nodes** *(Figure 1: blue, Figure 2(b))* passed data packets from relay and/or sensor nodes to other relay nodes, until the packets reached the the gateway node. These nodes contained a battery and a WizziMote. Together, these nodes formed the network of relay nodes transporting the sensor data to the gateway node. The main function of these nodes was to receive packets and transmit them. These nodes were able to buffer packets during periods when the network links failed and were unable to transmit packets towards the gateway node.

(3) **Gateway Node(s)** *(Figure 1: magenta, Figure 2(c))* was the aggregation point. On the network side, the gateway received packets via a WizziMote from one or more relay nodes. It then transmitted these data to a Raspberry P0 over a serial interface. The Raspberry Pi timestamped the data packets, stored them on a memory card, and periodically used a cellular modem to upload the data to an off-site server. In our deployment, there was a single gateway node that was powered by a solar-charged battery.

The relay and the sensor nodes were powered by non-rechargeable batteries, so energy savings was critically important for these nodes. Only the gateway node's batteries were charged by a solar panel. The most important reason for not using solar rechargeable batteries for the sensor and relay nodes was the need to hide the electronics from hikers, hunters, and animals. We needed to create a package that was small and could be hidden in trees and other hard-to-reach places where solar rechargeable batteries would not be very effective. This also meant that we could only fit a small number of batteries (sensors contained four, and relays contained only a single 19 Ah lithium thionyl-chloride batteries). This limited number of batteries further necessitated power savings. The gateway node with the solar panel was deployed in a tree and appropriately camouflaged.

4.1 The Deployed System

The network was deployed in November 2018 on the 3 km study reach of the lower Yuba River. The sensor network remained functional through May 2019. The study reach was immediately downstream from Daguerre Point Dam (Figure 1: not featured, just upstream of the top-right of the map) near Marysville, in California's Central Valley. The river is a multi-threaded river. The primary channel of the river is approximately 100 m wide in the study reach, with a bankful flow of roughly 141.6 m^3/s. The river has a course riverbed of large cobble, allowing for considerable hyporheic flows through the gravel. The river is heavily modified due to extensive aggregate gold mining in the region. This reach of river was in fact re-located a century ago, moved around the gold-fields and to the dam. As a consequence, the river is quite treacherous—surrounded by large gravel mounds. Further, the movement of the river was relatively recent, so there has been little ecological succession that would yield sedimentary organization. Indeed, much of the floodplain contains large, exposed cobble. It is a very difficult area to navigate, and low sediment organization, substantial hyporheic flows, high velocities, large cobble size, and forced channeling all contribute to a particularly dynamic study reach, prone to re-organization and movement during flood events (Figure 3). As such, sensors were placed within the river in places that were accessible by foot, without the need of a boat. Sensors were placed, orthogonal to the direction of flow, with effort made to run them through as many different geomorphic features as possible. Because of the limitations of this dynamic system, sensor placement was not random (though random number generators were used to select approximate location), field crew safety required adaptation to place sensors in areas that could be reached without workers being harmed or carried away by substantial flows.

4.2 Packet Types and Structure

There were three types of packets: **data (D; presently temperature data), health (H)**, and **test (T)**. Temperature packets consisted of a node ID (the unique identifier of the sensor node from which the data originated), sequence number (a counter that incremented at the sensor node for each transmitted packet so duplicate packets could be identified), sensor type (an ASCII character to characterize the sensor—presently "T" for temperature), message type "D" for data, first sensor ID, last sensor ID (sensor nodes were designed with multiple sensors per node—presently five—using the first and last sensor ID, we could infer whether sensor readings were ascending or descending in their ordering and allowed for the pinpointing of malfunctioning sensors), up to 20 bytes of data (in this application, up to 10 temperature readings), and two bytes of 0xFF to signify end of message.

A health packet (H) contained only metadata: node ID, sequence number, sensor type of "R" for recovery (sent when a node restarted or regained power after a power loss), and message type of "H" for health. Sent at one-hour intervals, the reception of health packets ensured that the network was functional between the originating nodes and the gateway. These packets were particularly useful during periods when network links failed. All nodes (including relay nodes)

Fig. 3. Image of a flood event in the lower Yuba River side channel, within the study reach, just downstream of the Gateway node. Marked in the image is the approximate location of a sensor (red), and a relay node (orange).

generated health packets, which allowed us to localize failures to a specific node or region of the network. A **testing packet (T)** was used for debugging purposes. Testing packets had a structure of node ID, sequence number, sensor type "T," and data type "T"; they were generated by handheld nodes, carried by researchers to test wireless communications between nodes as they were being deployed in the field, ensuring optimal node positioning and placement.

The maximum packet size was 28 bytes, set statically by the sensor node when creating packets. The first four of these 28 bytes were reserved for the packet metadata. The actual packet could be smaller, depending upon the amount of data collected by the sensors connected to the sensor node.

4.3 Medium Access Control (MAC) Protocol

Due to the limited access to the lower Yuba River, the lack of access to power, and the threat of vandalism posed to visible structures like solar panels, it was imperative to minimize power use. The sensor nodes aggregated temperature data from the sensors and transmitted to the next upstream relay node every 15 min. This implied that all sensor nodes could be powered down to low-power states with a duty cycle of 15 min. A related issue was that nodes that transferred data to the same upstream node use the same frequency. This led to collisions, which needed to be prevented to minimize power use.

Energy-efficient MAC protocols have been extensively studied in the context of wireless sensor networks with a sleep-awake duty cycle. Broadly, they can be categorized into four classes: synchronous, asynchronous, hybrid, and random [31]. Due to the lack of real-time synchronized clocks in the nodes, synchronous approaches such as T-MAC [32] and hybrid approaches that combine synchronous and asynchronous approaches could not be used. It was further impractical to implement any wireless time correction like GPS, as GPS reception can be impacted by vegetation, and this interference can yield errors in both positional and timing signals [33]. Additionally, GPS-based timers used considerably more power than was available in our installation. GPS receivers were also more expensive than our budget allowed. With low duty cycle sensors (such as those

implemented in this project), it has been found that idle listening can be reduced by having the transmitting node be the active entity when synchronizing with the receiving node [31]. A common way for this to be achieved is for the sender to—when it has data—transmit a preaamble that lasts for a length of time that is long enough to overlap with the short duration of time for which the receiving node wakes to listen for the preamble. This overlap is guaranteed even if the clocks in the transmitting and receiving nodes are not synchronized. If the receiving node detects the preamble, then it stays awake to receive the data. This is the approach adopted in B-MAC with additional optimization in X-MAC [31]. Recently, a randomized algorithm based on the Birthday protocol [34] has also been implemented [35].

The MAC protocol adopted by WizziMote was similar to B-MAC [36]. In the WizziMote, time was measured in "ticks," with each tick equal to $1/1,024$ seconds. For our deployment, we set the sleep duration to 900 ticks, while any advertisements were transmitted for 1,000 ticks. This ensured that receiving nodes woke up before the end of the advertisement period of the transmitting node. During the 1,000 ticks, the transmitting node continuously sent packets containing the remaining time of advertisement (in ticks). The receiving node used this information to synchronize with the start of the data transmission. At the transmitting node, the packet was transmitted immediately after the completion of the advertisement. Once the data were transmitted, the transmitting node switched the radio interface to the receive mode to receive the acknowledgment. At the receiving node, if the data passed the checksum, then the node immediately acknowledged the transmission on the same (backward) channel. Notably, the acknowledgments did not use advertisements. The data transmission and the acknowledgment were performed in one single operation [37]. Furthermore, nodes transmitted data regardless of whether or not any node received the advertisement due to a link failure. If the transmitting node did not receive an acknowledgment, then it reattempted the transmission in between sleeping until the receiving node acknowledged the data packet. The reattempts were guided by an exponential back-off algorithm (Section 4.5).

The energy savings of this approach was derived from the fact that the sender advertised only when it had data to send and the receiver was only awake for only a small fraction of the time. The use of advertisement to synchronize nodes significantly increased the chance of collision at locations where multiple branches of the network converged (i.e., where multiple nodes transmit to a single node). At these locations, multiple nodes could be advertising, transmitting, or acknowledging at the same time. This yielded considerable interference in laboratory experiments. These transmission collisions substantially increased transmission times and diminished network fidelity, filling the distributed network storage (discussed in Section 4.5). To avoid these collisions, while still preserving the advertisement-based synchronization, we implemented cross-listening: The transmitting node listened on the channel before it started to advertise on the channel. This was similar to **Carrier Sense Multiple Access (CSMA)**. The main difference was that traditional CSMA with **collision detect (CD)** and CSMA with **collision avoidance (CA)** required the radio interface to operate in full-duplex mode, i.e., able to transmit and receive at the same time; whereas the WizziMote radio interface operated in simplex mode. Thus, in cross-listen, the transmitting node only listened for the advertisement and not a general carrier signal before any data transmission. By minimizing channel conflicts, reducing contention, and decreasing transmission attempts, cross-listening also reduced duplicate packet transmissions, decreased the time for a packet to reach the gateway node, and reduced network-wide power use.

4.4 Routing Protocol

Routing protocols in the context of wireless sensor networks have been extensively studied [38] and there is a standard routing protocol defined for low-power, lossy networks like wireless sensor networks [39]. There were multiple constraints that limited the design and implementation of the

routing protocol. First, WizziMotes did not have any in-built node discovery and routing algorithm. As a result, any routing algorithm had to be implemented from scratch. Besides the complexity of many of the standard algorithms, there was limited memory in each WizziMote (see Section 3.3). Additionally, there were both topographical constraints (dense foliage and sharp embankments at many locations) and constraints on the number of potential locations where relay nodes could be deployed. Consequently, the network of the relay nodes was a sparse network and there were only a few routes between the sensor nodes and the gateway node. Near many of the sensor nodes there was only a single reachable relay node.

There were a number of conflicting constraints imposed upon our network; in balancing them, we opted for simple static routing and implemented network storage to account for temporary link failures. The locations of the temperature sensors were fixed by the scientific requirements. Specifically, we needed to measure temperature along a 3 km river reach at regularly spaced locations. At each location, we needed multiple, evenly spaced (5–10 m) sensors. The sensor nodes that aggregated the cross-channel temperature at different locations along the river were constrained by the river's inherent geometry. The harsh terrain prevented a very dense deployments of relay nodes and was further complicated by the incidence of fouling and vandalism. The equipment and deployment budget limited the number of relay nodes. We therefore deployed a relatively sparse network of relay nodes. Furthermore, the 433 MHz radio that was implemented in the WizziMotes could reliably discriminate only four out of the seven available channels. For all of these reasons, we implemented a simple static routing strategy and used network storage to mitigate link failures.

To simplify the routing of packets and reduce energy use by eliminating the need for node discovery and distributed topology creation, all routes in the network were set statically. To minimize interference, nodes that were in the hearing range of each other but on different routes were assigned different channels. The channels were reused in links that were far apart. Developing the network with static routing simplified the implementation and provided greater control over the exact traversal path of packets. This came at the cost that there were no alternate paths for packets when field conditions changed. But the limited deployment meant that in most places on the river, no alternative path existed.

A potential approach to deal with link failures would be to set up static back-up paths. If the primary path broke, then the back-up path could take effect to maintain the connectivity of the network. However, this option would only be viable with a dense deployment of relay nodes. In fact, for a dense deployment of relay nodes and with more power, processing, and memory in the relay nodes, a dynamic routing protocol would be beneficial. The overall topology of the sensor and the relay nodes had an unbalanced tree structure with the gateway at the root node (Figure 1). It was only at merge points that nodes in tree branches were close enough to hear each other. We used cross-listen at these points to reduce collisions.

4.5 Network Storage

By doing a pilot study with a small, experimental field deployment, we found that weather, distance, local topography, vegetation, antenna position and direction, and other factors reduced link fidelity and intermittently hindered transmissions. Further, transmission were lost to collisions at points where network branches merged. For times of diminished network stability, we designed modules that minimized repeated re-transmissions while maximizing the throughput of unique packets. This was achieved by exponentially backing off of the transmitter when transmissions failed. For example, if the base transmission rate was every 4 s, then after subsequent unsuccessful transmissions, the base was binary exponentially increased to 8 s, 16 s, 32 s, and so on. When the link quality was impacted by weather, it often remained so for some time. Thus, the exponential back-off (as opposed to a linear back-off) reduced the number of unsuccessful transmission

attempts, and saved power. While the node slowed down its transmissions, it still received packets until its queue was filled. This, in turn stored as many packets as possible within the relay nodes in the network. Once the link became reliable—or in other words, once the node successfully transmitted a packet again—it quickly recovered from exponential back-off by immediately resetting its base to four seconds.

We have separated the network storage algorithm into two parts (**Algorithm 1**): The first was done by the transmit thread, wherein it retrieved the number of unsuccessful packet transmissions, and then the receive thread updated the back-off timer. This back-off timer reduced the frequency with which transmissions were attempted. In our current implementation, the delay began at four seconds between transmission attempts and doubled for each transmission failure, up to a maximum of 30 minutes. This maximum ensured that each node attempted a transmission at least every half-hour. Any successful transmissions restored the delay to four seconds.

ALGORITHM 1: Network Storage Algorithm

Input: Back-off timer T_B; Counter C_f for failed transmission in the last round; Transmission base value B_B; Max transmission back-off B_{MAX}; Minimum transmission back-off B_{MIN}.

procedure TX(C_f, B_B, T_B) ▷ For any TX round i
  if Successful transmission **then**
    $C_f \leftarrow 0$
    $B_B \leftarrow T_{MIN}$ ▷ reset transmission base
    $T_B \leftarrow 1$
  else if Unsuccessful transmission **then**
    $C_f \leftarrow C_f + 1$
  end if
end procedure

procedure RX(C_f, B_B, T_B) ▷ For RX round i+1
  if $C_f > 0$ and $B_B < B_{MAX}$ **then**
    $C_f \leftarrow 0$
    $B_B \leftarrow B_B \times 2$
  end if
  if $T_B \bmod B_B == 0$ **then**
    $T_B \leftarrow 1$
    call TX procedure
  end if
end procedure

These transmission delays coupled with the 10-slot queues in each relay node coalesced to yield up to a few hours of distributed network storage (depending upon the design and topology of the network). This data retention was crucial for scientific sensing, which could generate a steady stream of sensor-derived data packets. Their timestamps were reverse-calculated once they successfully reached the gateway (Section 6).

5 NODE SOFTWARE ARCHITECTURE

In discussing the software architecture of the nodes, we focus on the relay nodes, as they were the only nodes that contained both transmit and receive functions.

5.1 Event-driven Proto-threads

Nodes in the network utilized the WizziMote's **Kernal Abstraction Layer (KAL)**, which maintained the illusion of multi-threading at a high level by encapsulating the Contiki proto-thread library [27], an open-source operating system that offers dynamic loading of application code onto embedded systems. While similar to a threaded architecture, pseudo-threads cannot run in parallel. Proto-threads are lightweight and stack-less, providing a blocking context on top of an event-driven kernel. Sequential flow of control is implemented with macros that save the processing states of functions without using complex state machines or full multi-threading. Proto-threads were particularly well suited for memory constrained devices (WizziMotes had only 4 KB of memory). Another useful feature is that preemptive multi-threading is implemented in the proto-threads as an application library. Consequently, multi-threading can be optionally linked with programs that explicitly require it [40].

Two types of events—timer and radio—controlled a pseudo-thread's execution flow. A pseudo-thread was always in one of the following states: inactive, processing an event, or waiting for an event. The WizziMote radio library was written with only one radio buffer. While both the transmit and receive threads could be active at the same time, they could not process radio events simultaneously because of the shared buffer. To prevent conflicts over radio resources, these threads were scheduled sequentially (transmit after receive).

5.2 Packet Queue

A finite packet buffer, implemented as a queue, connected the threads and acted as shared storage. Packets received from the receive thread were inserted into the queue, which were then removed from front of the queue by the transmit thread. To reduce network load from re-transmissions that resulted from unheard acknowledgments, the receive thread compared each received packet with the packet received before it. Identical, sequential packets were not inserted into the queue.

5.3 Thread Architecture

In each relay node, there were three threads: the management thread, the **transmit (TX)** thread, and the **receive (RX)** thread (Figure 4). At points of intersection in the topology, we also introduced the cross-listen thread for collision control.

5.3.1 Management Thread. (**Finite State Machine (FSM)**: Figure 4) as the name suggests, managed the transmit and receive threads by initializing them with the necessary parameters. There was also a main() function that initialized the **Hardware Abstraction Layer (HAL)**, KAL, and the management thread. The main() function and management thread coexisted because all KAL processes had to exit before the node could transition into sleep mode. Since the management pseudo-thread controlled the other threads, killing the management thread terminated the other threads, which yielded a clean state.

In addition, the main() function controlled two timers: the main timer, which put the node to sleep (see Section 5.4), and the watchdog timer, which detected inactivity. The MSP430's built-in watchdog timer performed a hardware restart if the timer was not reset within 16 seconds, which allowed for recovery from any hangs or bugs. This was particularly useful in our implementation, as elements of the WizziMote's software were provided pre-compiled and we did not have access to their source [28].

5.3.2 Receive (RX) Thread. (FSM: Figure 5) listened on the backwards channel for any incoming messages and inserted them into the queue. If nothing was on the channel, then the node switched to the transmit thread to transmit any packets that were in the queue. If the RX thread successfully

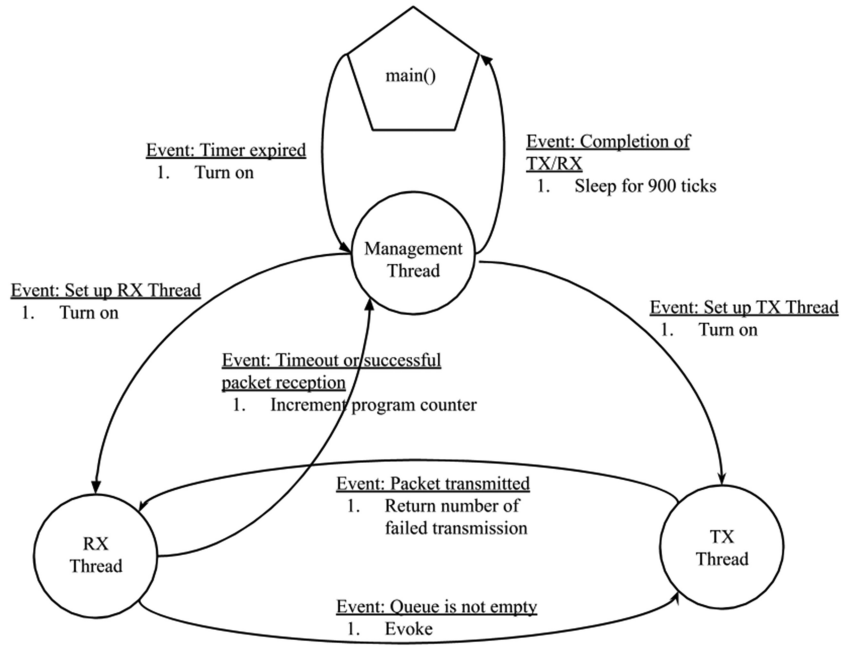

Fig. 4. Finite State Machine (FSM) showing the overall organization of the pseudo-threads and the events that triggered transitions to the different threads.

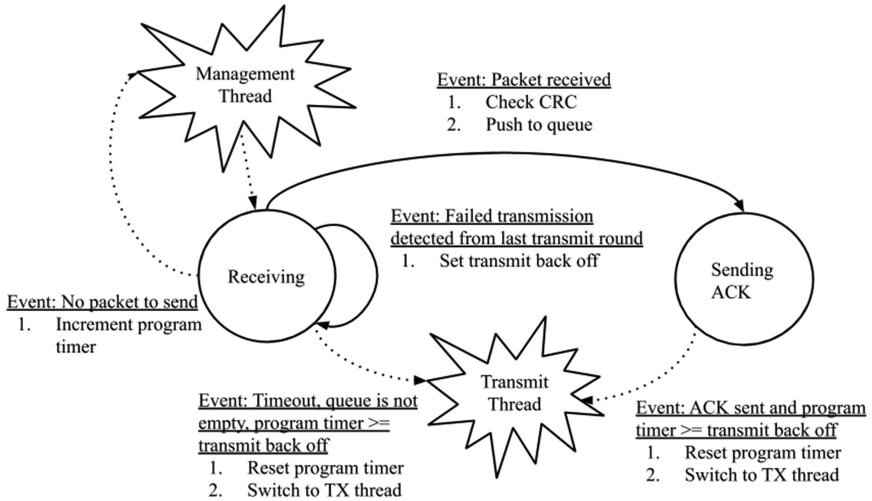

Fig. 5. The Finite State Machine of the RX thread referenced in Figure 4.

received a packet, then the thread acknowledged the packet to the transmitting node on the backwards channel, which signaled a successful transfer.

In the sequence of events upon waking up from sleep, the receive thread was instantiated before the transmit thread. This was to fill the queue as much as possible to maximize the storage utilized in the network. Even when the forward channel was broken, the node continually tried to receive until the queue was full.

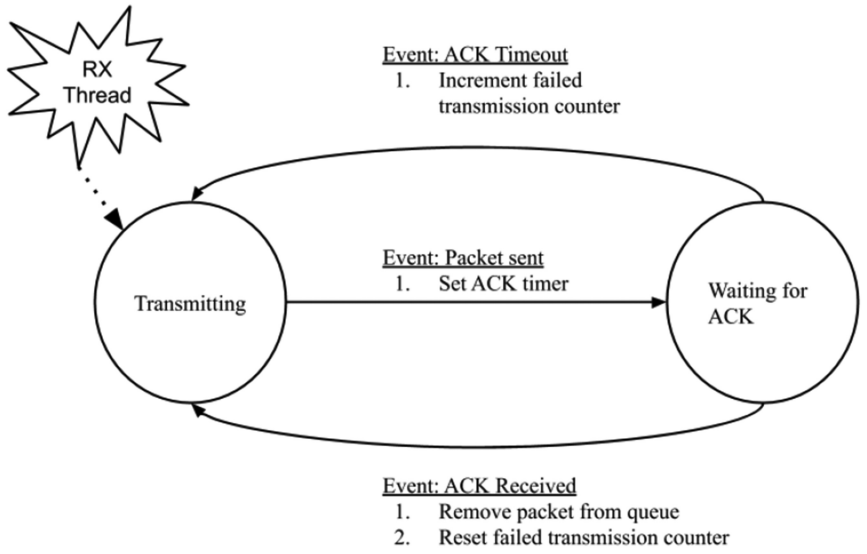

Fig. 6. The Finite State Machine of the TX thread referenced in Figure 4.

5.3.3 Transmit (TX) Thread. (FSM: Figure 6) transmitted packets (if there were any in the packet queue) on the forward channel. First, an advertisement was transmitted continuously on the forward channel to signal that a transmission was to be sent. The advertisement period had to be longer than the sleep period of the receiving node to ensure that the receiving node's listening period would overlap with parts of the advertisement. At the end of the advertisement, the node transmitted the data packet, regardless of whether a receiving node was active to listen for it or not. Finally, the transmitting node listened for an acknowledgment, which if received, prompted the removal of the transmitted packet from the sending node's queue. To minimize power use, the transmit process was not invoked every cycle of the node. If the forwarding channel was down, then the node only transmitted after the delay timer had expired (Section 4).

5.3.4 Cross-listen Thread. There were nodes at confluences within the network, where two or more nodes transmitted on the same channel to the same receiving node. In these nodes, an additional thread was added to minimize transmission collisions. Acting much like the receive thread, the cross-listen thread listened on the forward channel to see if another node was already transmitting. If it heard no transmissions, then it notified the TX thread to transmit the packet. If the channel was in use, then it backed off for a period of time. This delay was randomized to prevent additional collisions that could propagate from non-random or hard-coded delays.

5.4 Energy Efficiency

For nodes to survive in the field with limited battery power, they were put to sleep when not transmitting or receiving. During this sleep, each relay node was pulled to Low Power Mode 3 (LPM3), which disabled the **CPU (MCLK)** and the **high frequency clock (SMCLK)**, leaving only the 32 kHz low-frequency crystal clock ACLK powered [29]. In the main() of each relay node, a timer of 900 ticks was initialized and started, which created an interrupt and returned the node to Active Mode.

```
main_timer = kal_timer_start(KAL_ETIMER, NULL, TIMER_ID, 900);
```

A few lines later, after closing the radio layer and the serial, the node was put into LPM3 and global interrupts were enabled. The node woke up when the main timer timed out.

```
_BIS_SR(LPM3_bits + GIE);
```

This process was repeated after every cycle of receive and transmit.

6 DEPLOYMENT AND EVALUATION

The wireless sensor network was installed and active from November 2018 through May 2019. Many of the actual sensors were secured into the river over the summer of 2018, as low flows exposed more of the riverbed. The networking hardware was added to the sensors in October and November 2018 and brought online as it was completed. The full network was completed and functional in January 2019. During the initial deployment, some relays needed to be re-positioned or relocated to address changes in flow and vegetation (Figure 1). In May 2019, a restoration project began, which substantially modified the river channel in the study reach. This work rendered network maintenance impossible. In January and February 2019, much of the river became inaccessible due to flooding, and repairs were more challenging. The low-power, small footprint of all hardware yielded a relatively low-effort field deployment. Field trips were made as day trips from UC Davis, often with just a single worker in the field. Additionally, because all hardware was programmed in advance and sealed prior to the field deployment, field workers needed no specialized skills.

6.1 Power Use

To determine the power used by our relay nodes, we conducted an experiment that logged the current drawn by relay nodes. We did this by setting up three nodes: a dummy-sensor node that generated numerous packets at regular intervals, a relay that transmitted the data, and a gateway node that received the data from the relay. We set up the dummy-sensor node to transmit at regular intervals of 4, 8, 16, or 32 seconds. The relay node was connected to a 3.600 VDC source, through a high-speed, logging bench-top multimeter to measure draw. We measured voltages both above and below this logging meter to ensure that burden voltage was not too large and that supply voltage to the relay node stayed above 3.500 VDC. These voltage monitoring multimeters were both put into their high-speed max/min modes, which measured at least once every 1 ms. These multimeters were recently calibrated; one was an Amprobe AM-270 and the other a Fluke 87. The power source was a recently calibrated Power Designs 4010. The current was logged by a recently calibrated Rigol DM3051. This meter was capable of 500 samples per second, with storage for 200,000 samples—yielding 6 minutes and 40 seconds of continuous sampling.

Each of the relay node's states were reflected consistently in the data (Figures 7 and 8). Low-power sleep states used 2–3 μA, active mode used about 5 mA, and receive and transmit modes used 20–30 mA (Figures 7 and 8). These values were consistent with the anticipated power use from the CC430F5137 datasheet (Section 3.3) [28, 29].

6.2 Network Storage

To test the effectiveness of the network storage, we created a simple experimental network in the laboratory that simulated link instability and observed the network recovery and loss. We considered a linear network of a sensor node connected to a gateway node through two relay nodes. The sensor node transmitted data every 30 seconds, and the relay nodes had a packet queue of 10 packets. The link between the second relay and the gateway was shut down with increasing

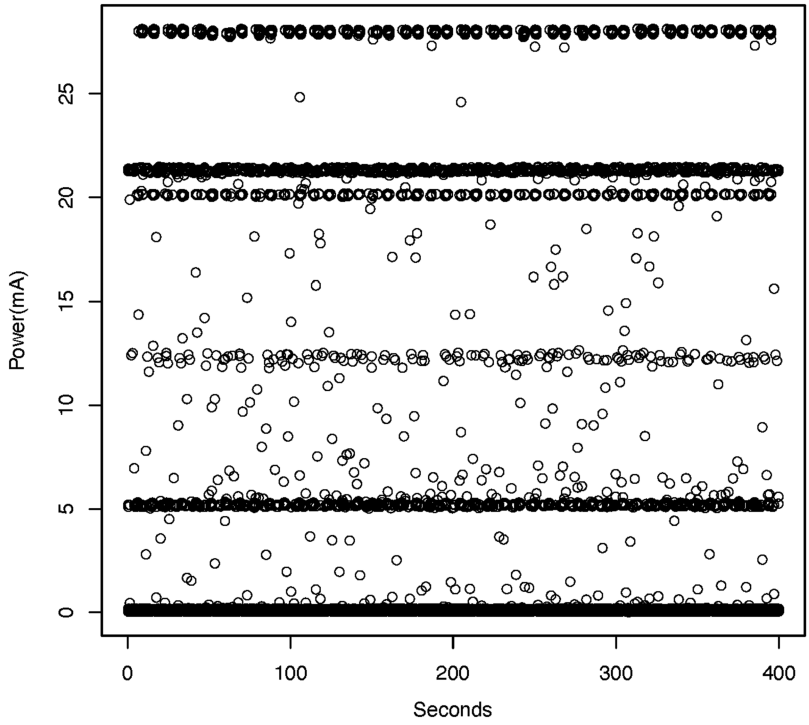

Fig. 7. Raw power-use data taken from the Rigol DM3051 in mA when relay node received a data packet every eight seconds. It is notable that the different power modes were very consistent. The readings near 0 mA were low-power sleep states (2–3 μA), the 5 mA readings corresponded to active mode, and those in the 20–30 mA range corresponded to receiving and transmitting. All others are transitional between states.

duration of 10, 100, 600, and 1,000 seconds. Each link failure was followed by 60 minutes of potential recovery time. Recall that the definition of network recovery in our network storage algorithm is equivalent to a successful transmission through the broken link (**Algorithm 1**). Hence, we defined the time for network recovery time as the time difference between the link failure and the receiving of the first packet after the link became available again. Note that there existed an idle time period during which the network has physically recovered, but was undiscovered until a packet traversed the previously broken link. For the purpose of this application, we considered the link to be dysfunctional during this idle time period, because the network health was dependent on the overall traversal of packets in the network rather than the individual link health.

Figure 9 shows the box plot of the recovery time for different durations of link failure. As the link breakage time increased, the time to recovery also increased. As the link was broken for longer periods of time, the delays in between transmissions increased exponentially and so did the recovery time of each node. The queue size also influenced the back-off rate of the network. When the queues of connected relays filled, the overall recovery time of the network increased as multiple relay nodes' back-off durations compounded.

The rate of packet creation was significantly greater than in a real-world deployment, which ensured that the packet queues would fill while the link was broken (particularly during the longer duration link failures). The purpose was to test the back-propagation of packet storage. This effect was observed by looking at the losses. For any given queue size, breaking any link in the

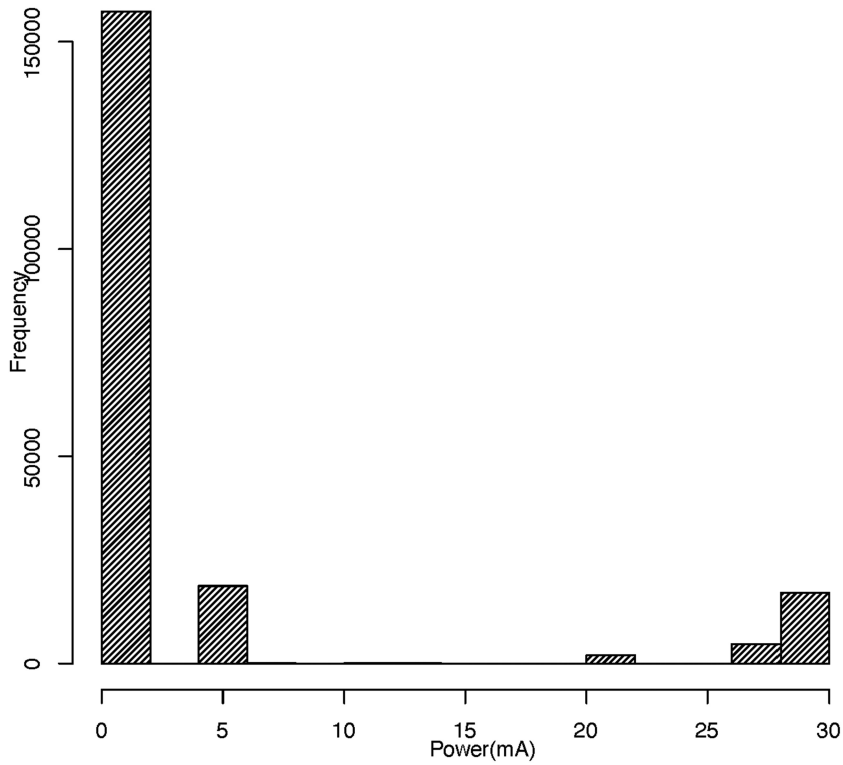

Fig. 8. Histogram of the power used in mA when relay node received a data packet every eight seconds. Each of the peaks in this histogram corresponded to a state for the Wizzimote. The peak around 0 mA corresponds to the low-power sleep state, 5 mA corresponds to active mode, and the peaks between 20 and 30 mA correspond to transmit and receive modes.

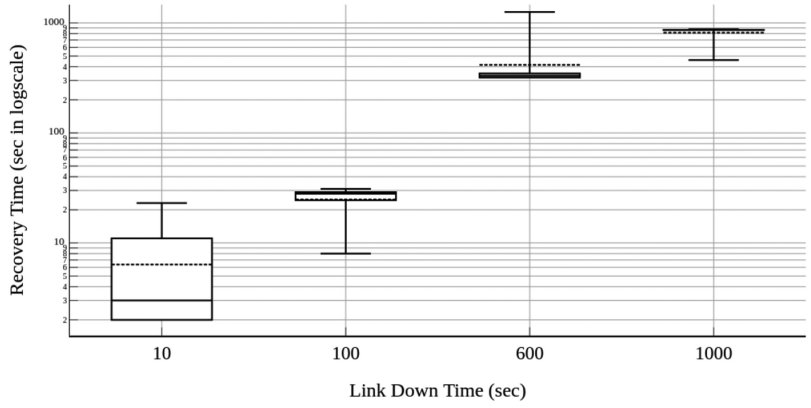

Fig. 9. Network recovery time in seconds (log scale) as a function of link failure duration. The mean recovery times are shown as the dashed line and were, respectively, 6.36 s, 24.83 s, 415.36 s, and 815.75 s. The median recovery times were, respectively, 3 s, 28 s, 331 s, 865.5 s.

network had no effect on the packet loss if the down time was relatively low. Once the queues of all available relays were filled, the packet loss increased in proportion to the link down time. In a real-world implementation, the packet frequency would be much smaller, and the number of relays much larger, granting the network more storage over a longer duration of time, minimizing the probability of packet loss.

6.3 Timestamp Alignment

The distributed network storage implemented in this network resulted in the accumulation of unknown delays during the transit of sensed temperature data. The network contained no real-time clocks, and packets were timestamped upon receipt at the gateway, after any applicable network delays. It was necessary to re-align these packets with real-time. Re-alignment was feasible, as each packet contained an 8-bit packet ID, which was sequential (rolling over at 255). We performed the re-alignment in a two-step process, first identifying improperly timestamped packets (Figure 10(a)), then re-aligning them (Figure 10(b)). The central idea underlying both the identification and re-alignment of packets was the use of neighboring packets to estimate the timestamp for each packet.

While packets from other sensors may be useful for aligning packets in situations where transmission delays propagate from sensor or network drop-outs, most of our network failures were the result of power-failures at the gateway. Since timestamps were generated at the gateway, and most link failures propagated from the gateway (as it was the only solar-powered unit in the network and was most prone to power loss during inclement weather), we did not use packet timestamps from other sensors to align packets, opting to instead use packets from each sensor to exclusively align timestamps from that same sensor.

Each sensor generated a data packet about every 15 minutes, and each packet contained a sequential packet ID. It was therefore possible to estimate the timestamp of any individual **packet (P)** using the timestamps of neighboring packets (X of length N). By taking the difference in packet ID and multiplying by time between sensor readings (i.e., 15 min), we were able to estimate which packets were properly aligned. The method was robust to many different modes of packet delay, because it treated most aspects of packet timings as variable. To this end, we knew that the time between sensor readings (T) was determined by a crystal, and never synced to any clock, and therefore fluctuated with temperature. To improve timestamp estimates, we began by setting T=14 min, then increased it in five-second increments until the model heuristics (Figure 10) indicated that we had reached an optimal timing for each packet. When evaluating whether timestamps were aligned (Figure 10(a)), we evaluated the range of estimates, the distance of the estimate from the observed packet timestamp, and the number of estimates used to generate the estimated timestamp. In each step, we also removed the minimum and maximum estimate in an effort to improve the mean estimate and decrease the spread of estimates. If the estimates derived from a set of neighboring packets failed to meet the appropriate model heuristics, or if after filtering out minimum and maximum, the number of estimates fell below L (L=6, presently), then we increased the number of packets considered (N) to include additional packets before and after the packet in question. In this way, we prioritized using the packets closest to the packet in question, expanding the window only when estimates were not sufficiently robust using a smaller numbers of packets. If N reached 72 (36 packets before and after P), then we marked packet P as bad, meaning that we could not create a sufficiently robust estimate of the correct timestamp for P. Once each packet P had been evaluated, any marked bad were removed from the data and the algorithm started over, evaluating all remaining packets not identified as bad in previous rounds and using only the remaining packets not already marked bad to evaluate each packet within the reduced dataset. This repeated until the list of bad timestamps grew by less than 5% in a given round. At this point, we considered

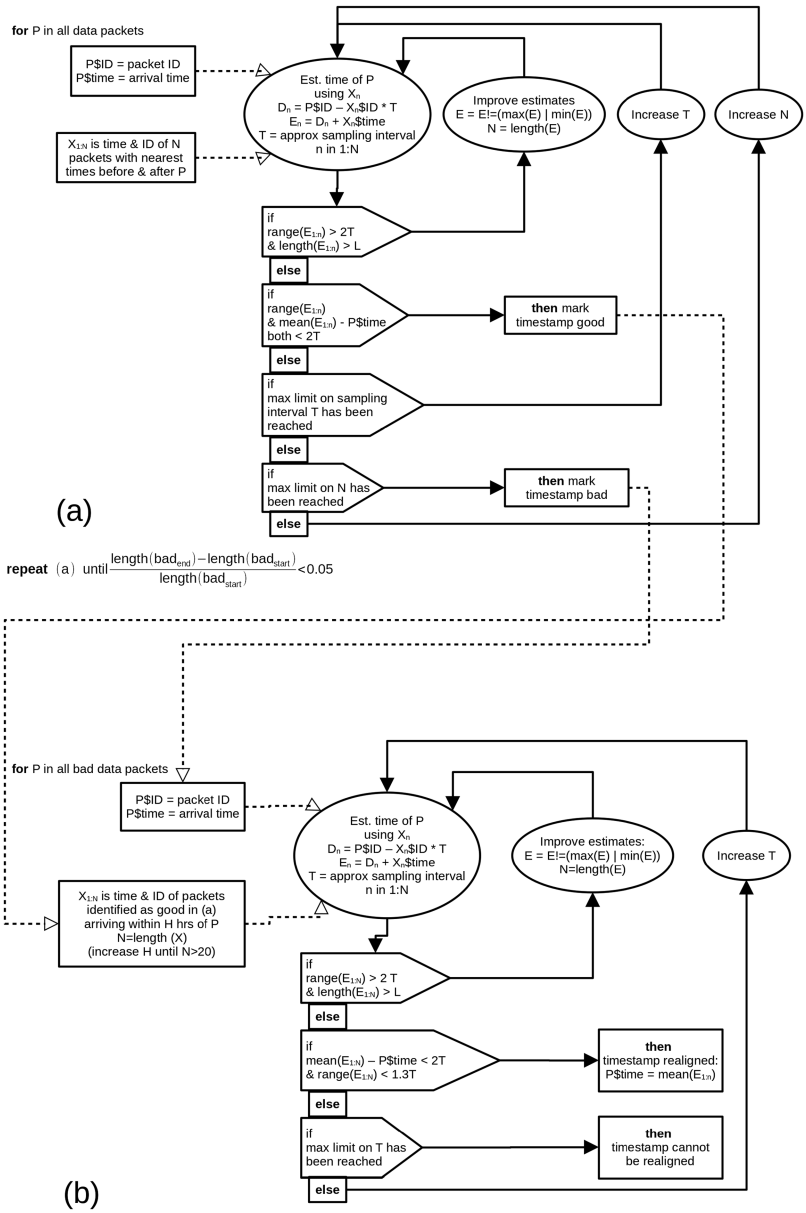

Fig. 10. Flow of control (filled arrows) and data (empty arrows) for (a) bad timestamp identification and (b) bad timestamp re-alignment. In both diagrams, T is the approximate time between data packets sent by the sensor (about 15 min). P is the packet that is being evaluated or re-aligned, X are the N-packets that are being used to evaluate the timestamp of P (notably, the selection of these packets differs between (a) and (b)). L is a means of ensuring that despite the removal of outlying time estimates, sufficient estimates are retained to ensure robust estimates.

the list of bad packets to have been identified for that sensor. These bad timestamps were then fed into a timestamp re-alignment algorithm (Figure 10(b)).

The process for timestamp re-alignment was similar to that of bad timestamp identification. We used neighboring packets that were not identified as bad (Figure 10(b)). Instead of using the

Table 1. Number of Timestamps, Broken Out by Sensor, Including Those with Bad Timestamps as Well as Those that Were Able to Be Realigned

Sensor	Date of First Packet	Date of Last Packet	Total Packets	Bad Timestamps	Corrected Timestamps
A	2018-12-05	2019-02-13	3,578	710	288
C	2018-11-06	2019-05-21	7,247	1,850	487
D	2018-11-12	2019-05-21	8,596	2,241	1,032
E	2018-11-01	2019-05-25	12,205	1,185	764
F	2018-11-07	2019-03-02	6,446	866	312
J	2018-12-14	2019-02-23	4,073	1,592	630
K	2018-12-14	2019-03-06	3,571	1,150	377
Total	2018-11-01	2019-05-21	45,716	9,594	3,890

Total number of packets is the sum of good and bad timestamped packets. Re-aligned timestamps are a subset of bad timestamps—all others could not be re-aligned.

N-nearest packets as above, we instead used good packets arriving within H hours of P. H was initialized at 7 hours before and after the documented timestamp of P, increasing iteratively until the **number (N)** of **packets (X)** used to estimate the timestamp of packet P was at least 20, with H having a maximum of 48 hours. Once H was set and the **N-packets (X)** within H hours of P, the timestamp of P was estimated and the minimum and maximum estimates were eliminated until the range of estimates was less than 30 min. If this could not be reached, then as above, the inter-packet **timing (T)** was incremented by 5 seconds and the process was repeated. This was repeated until T reached 16 minutes.

The parameterization of variables and coefficients in our implementation were largely based upon the scientific questions at play. We were interested in hourly river temperature data. As such, we sought at least two measurements per hour. This is why we chose to set our rejection thresholds at 30 min for both for evaluating whether or not timestamps were correct, relative to timestamp estimates (i.e., mean (timestamp estimates) - observed timestamp) as well as ranges of timestamp estimates. Other applications of these methods should use parameters informed by relevant research questions.

We identified 9,594 incorrectly timestamped packets. Of these, 3,890, or about 40%, were able to be re-aligned using this procedure (Table 1). As a result of this identification and realignment, 40,012 packets were able to be used in further analyses of fluvial water temperatures. Notably, each packet contained five sensor readings, yielding a total of 200,060 properly aligned or re-aligned temperature readings from the sampling period (Table 1). For most sensors, timestamp re-alignment was most common for network delays of under 10 hours (Figure 11). Indeed, for every sensor except sensor E, 75% of re-aligned timestamps were shifted less than 10 hours (Figure 11). However, for sensor E, which was closest to the gateway, large delays were more readily fixed during timestamp re-alignment. As a result, the median re-alignment was much larger (over 10 hrs: Figure 11). Notably, timestamp re-alignment was feasible for up to 20 hours for at least some packets from each sensor (Figure 11). Timestamp alignment utilized the sequential packet ID, which rolled over at 255. Any timestamp alignment was infeasible after 63.75 hr, when the counter rolled over. Re-alignment also successfully corrected hourly mean temperatures, bringing them in line with what would be expected, with lowest temperatures in the early morning and warmest temperatures in the late afternoon (Figure 12). Prior to timestamp processing, the mean temperature was elevated in the early morning. Timestamp alignment rectified these high mean temperatures, as these incorrectly timestamped packets were shifted to the correct time.

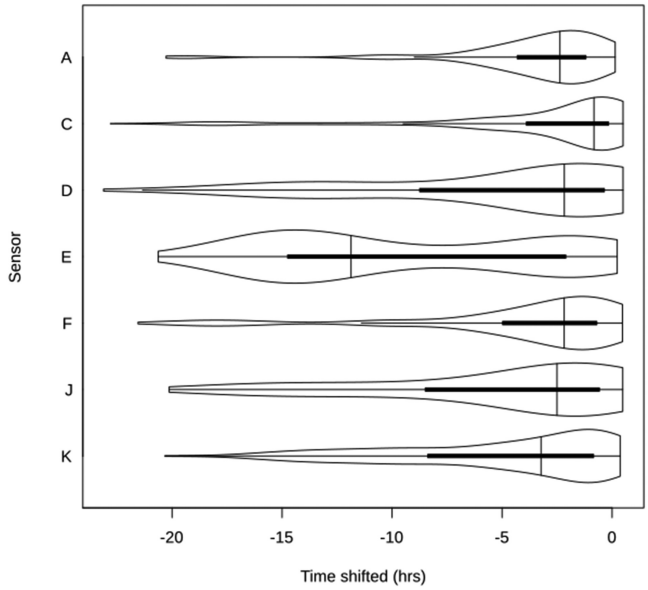

Fig. 11. Violin plots of the amount of time shifted during timestamp re-alignment, broken out by sensor. Violin plots also contain box plot, with whiskers at 0th and 100th quantile, and box extents to 25th and 75th quantiles. Perpendicular lines are at medians.

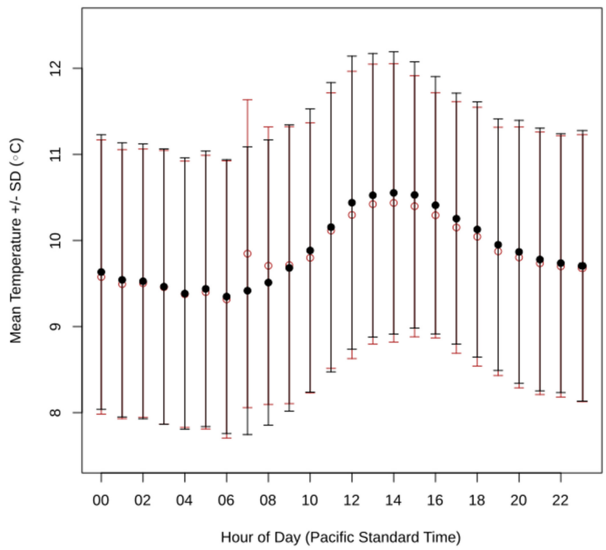

Fig. 12. Temperature data, broken out by time of day (local, Pacific Standard Time), with mean and standard deviations in temperature across all sensors for the whole of the sampling period. Red, unfilled circles demonstrate the data as collected. The red plot includes the data that were rejected during timestamp alignment and also uses the raw, unprocessed timestamps. The black, filled circles are the data after processing: All incorrectly timestamped packets have either been re-aligned (if feasible) or removed.

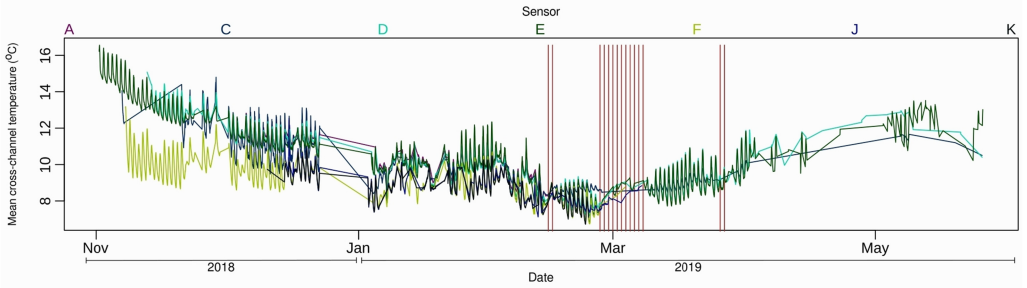

Fig. 13. Temperature data collected from the wireless sensor network. These temperatures represent cross-channel means: They are the average of the five individual sensors attached to each sensor-node. The vertical red lines are at midnight on days in which there was a flood of at least twice ($283m^3/s$) the bankful flow ($142m^3/s$).

6.4 Temperature Data

We collected temperature data from November 2018 through May of 2019 (Figure 13). In this time, we collected 200,060 temperature measurements. Different sensors intermittently failed at different times, as a result of varying network failures (Figure 13). Many were damaged during large flow events (Figure 13: red lines). Notably, sensor E persisted longest, as it was nearest to the gateway and subjected to the fewest relay failures (Table 1). Upon re-aligning the data, we found a predictable temperature curve, with hourly mean temperatures peaking in the late afternoon and reaching a minimum in the early morning (Figure 12).

In total, 3,890 packets were re-aligned. We initially questioned the balance between the uncertainty implicitly introduced by timestamp re-alignment and the value of additional data. We came to the conclusion that re-alignment was crucial, because while it represented a relatively small number of packets (only about 11% of the total packets found to have been properly aligned: Figure 10(a)), timestamp mis-alignment was biased towards packets collected at night or during inclement weather, when network failures were most common. Eliminating 3,890 sensor readings explicitly collected at night and during rain would have hampered our ability to model river temperatures during these important times. This re-alignment was therefore crucial to the success of the network and the underlying ecological science (Figure 12).

7 LESSONS LEARNED

This project began in 2014 and spanned five years from its conception through the design, deployment, data collection, data post-processing, and analysis. It provided a unique opportunity to develop and deploy a specialized wireless sensor networks for scientific applications in harsh conditions. The following are some of the important lessons learned from this project:

(1) Sensors developed to measure temperature across the channel were implemented in a thick cable, which was anchored to the riverbed (Figure 2; Section 3). Despite these steps, some of the sensors were damaged during flood events or by wildlife. More robust sensor cables could have reduced the hazard rate and extended the service life of the sensors.
(2) Even though the gateway was powered by solar rechargeable batteries, it failed during extended dark periods such as nights followed by cloudy or rainy days. To increase resilience to dark periods, the gateway could have been equipped with additional batteries.
(3) The MAC protocol that was implemented in the WizziMote was a transmitter-driven advertisement-based synchronization approach (Section 4). In such a design, the energy

cost of synchronization is predominantly borne by the transmitter. An alternative approach would be a probabilistic approach, such as that based on the birthday paradox [34, 35]. In this approach, the transmitter and the receiver randomly advertise and poll the channel and it can be shown that they have a very high probability of synchronization even with a relatively low number of polls. The energy cost of synchronization is more balanced between transmitter and receiver, though not completely balanced, as the receive and transmit costs are not the same. This approach was evaluated and found to be energy-efficient in a field deployment [35].

(4) An important lesson pertained to our reliance on the closed-source radio library. The WizziMote radio library was not open source. This hindered our ability to address bugs and develop the link layer protocol. For example, since we did not have access to the source code of the radio library, it was impossible to implement the birthday protocol [35]. Additionally, the radio only reliably discriminated 4 out of the 7 channels standardized in the DASH7 protocol stack. This also limited our channel assignment.

(5) The DASH7 protocol allows the transmit power to be dynamically adapted. Dynamically increasing the transmit power during inclement weather when the link quality deteriorated could be adapted into our back-off algorithm to save power and improve successful transmissions. While it would significantly increase the complexity of the transmit protocol, in the long run it could save energy by reducing the number of reattempts during periods of poor link quality.

(6) As mentioned in Section 6, we did not timestamp the data at the sensor nodes, because the clocks in the WizziMotes were not accurate, and there was significant clock skew. Instead, we included a sequence number in the packets. The data were timestamped at the gateway node. However, when packets were stored in the network due to link failures, the timestamps had to be aligned using the sequence numbers (as described in Section 6.3), which was prone to errors resulting in some packets to be discarded (Table 1). One heuristic would be to implement a millisecond counter of delays in the packet header as the packet proceeded from the sensor node to the gateway node. This delay could significantly improve efficacy of the timestamp alignment algorithm [35].

8 RELATED WORK

Wireless sensor networks (WSNs) [11, 12] have been and continue to be used in many different applications. These include monitoring application for early detection of forest fires [13], actuating applications in precision agriculture [14], energy usage control application in smart homes [15, 16], and tracking applications such as animal telemetry [18]. Wireless sensor networks have been used in habitat and environment monitoring [17, 41–43]. One of the earliest comprehensive works in this area was the habitat monitoring on Great Duck Island [41]. The project monitored many environmental parameters, including temperature, barometric pressure, humidity, and others. Furthermore, due to the remote location, sensor power use was an important design constraint. The paper provided significant details on the design of the sensor node, the wireless sensor network, the data aggregation through patch gateways, and power budget and evaluation. Another seminal work on the study of scalability of wireless sensor networks was the measurement study performed in GreenOrbs [42]. Based on detailed measurement study of congestion in a large deployed wireless sensor network, the study proposed novel event-based routing strategies that were appropriate for large-scale sensor networks. Another relevant study was the CitySee project [43], which used wireless sensor network for real-time monitoring of CO_2 in a large urban area. The paper addressed the challenges of deployment. Despite the similarities of these prior works, each of these environmental deployments, including the river deployment in this article, have unique features that address

specific challenges, and as a result the specific techniques and approaches were not necessarily transferable.

Techniques to minimize energy use in wireless sensor networks have been extensively studied [44]. In Reference [45], a **Sparse Topology and Energy Management (STEM)** algorithm was proposed to efficiently wake up nodes from a deep sleep state without the need for an ultra low-power radio. The key innovation was to allow tradeoffs between energy efficiency and the latency incurred to wake up the node. A different tradeoff, between data fidelity and energy efficiency, has been investigated [46]. There has also been a number of studies that have proposed energy-efficient routing schemes [47, 48].

In our work, we have used an advertisement-based node-synchronization method. An alternative approach is a randomized algorithm based on the birthday paradox, which has been proposed for wireless sensor networks [34]. Specifically, the paper adopted the birthday paradox to develop a suite of protocols that save energy during the deployment phase of the sensors as well as the node discovery phase. The approach has been been adopted in a number of follow-up studies [49–51] including for a randomized data transfer protocol [35]. A comparative analysis of the two approaches (advertisement-based and randomized) is beyond the scope of this article.

9 CONCLUSIONS

In this article, we have presented a detailed design, implementation, and deployment of a wireless sensor network that collected river-temperature data at fine spatial and temporal scales. The overarching goal was to use temperature data, along with other data including fish habitat quality data, as well as hydrodynamic and geomorphic features to develop models of the forcings on river-temperatures. These models will better inform our understanding of juvenile salmonid energetics at ecological scales. In this article, we discussed the challenges of deploying the mesh network and how network storage was implemented to address intermittent link failures during inclement weather. We also described the design of the relay node and how a pseudo-thread architecture along with low power modes were leveraged to maximize battery life. We did not achieve 100% success, as a number of the sensors were destroyed during periods of heavy flow and by animals. Nevertheless, we were able to get considerable amounts of data, including particularly valuable and unique data from flood events. These data were valuable, because flood events are difficult to study, as they often scour and fill rivers and can render deployed logging sensors unrecoverable. As our data were uploaded in real-time, we obtained considerable data during these events. These data will help address questions about how river temperatures vary through both space and time and under differing flow regimes. This micro-habitat understanding of fluvial temperatures will inform future models of habitat quality for juvenile salmonids.

ACKNOWLEDGMENTS

The authors wish to express gratitude to the the entire lower Yuba River field crew from Pacific States Marine Fisheries Commission, without whom this study would have been impossible, notably Loren Stearman, Duane Massa, and John Cleveland. The extent to which Chelsea Martinez contributed to this work cannot be overstated; her impeccable organization and reliable field support were crucial to our success. We thank Richard Hambrick, who was instrumental in getting the development of this software off the ground and Megan Werdmuller von Elgg for her editorial feedback on this manuscript. Finally, we wish to thank the folks who helped us with this mountain of field work: Darcy Bostic, Annelise Del Rio, Arielle Gervais, Julius Henry, Sean Luis, Vanessa Lo, Seanna McLaughlin, Brianna Ordung, Lea Pollack, Kathy Russel, Rebecca Walker, Jason Wiener, Katie Woodworth, and Ken Zillig.

REFERENCES

[1] J. V. Ward. 1985. Thermal characteristics of running waters. In *Perspectives in Southern Hemisphere Limnology*. Springer Netherlands, Dordrecht, 31–46.

[2] B. W. Webb and F. Nobilis. 1997. Long-term perspective on the nature of the water temperature relationship: A case study. *Hydrol. Process.* 11 (1997), 137–147.

[3] K. Smith. 1975. Water temperature variations within a major river stream. *Hydrol. Process.* 6 (1975), 155–169.

[4] R. J. Danehy, C. G. Colson, K. B. Parrett, and S. D. Duke. 2005. Patterns and sources of thermal heterogeneity in small mountain streams within a forested setting. *Freshw. Biol.* 208 (2005), 287–302. DOI: http://dx.doi.org/doi:10.1016/j.foreco.2004.12.006

[5] N. J. Hetrick, M. A. Brusven, W. R. Meehan, and T. C. Bjornn. 1998. Changes in solar input, water temperature, periphyton accumulation, and allochthonous input and storage after canopy removal along two small salmon streams in southeast Alaska. *Trans. Amer. Fisher. Societ.* 127 (1998), 859–875. DOI: http://dx.doi.org/doi:10.1577/1548-8659(1998)127<0859:CISIWT>2.0.CO;2

[6] D. Caissie. 2001. The thermal regime of rivers: A review. *Freshw. Biol.* 51 (2001), 1389–1406. DOI: http://dx.doi.org/doi:10.1111/j.1365-2427.2006.01597.x

[7] P. A. Conrads and E. A. Roehle. 1999. Comparing physics-based and neural network models for simulating salinity, temperature and dissolved in a complex, tidally affected river basin. In *South Carolina Environmental Conference*.

[8] D. Caissie, N. El-Jabi, and G. Satish Mysore. 2001. Modelling of maximum daily water temperatures in a small stream using air temperatures. *J. Hydrol.* 251 (2001), 14–28. DOI: http://dx.doi.org/doi:10.1016/S0022-1694(01)00427-9

[9] M. B. Cardenas, J. W. Harvey, A. I. Packman, and D. T. Scott. 2008. Ground-based thermography of fluvial systems at low and high discharge reveals potential complex thermal heterogeneity driven by flow variation and bioroughness. *Hydrol. Process.* 22 (2008), 980–986. DOI: http://dx.doi.org/doi:10.1002/hyp.6932

[10] John S. Selker, Luc Thevenaz, Hendrik Huwald, Alfred Mallet, Wim Luxemburg, Nick van de Giesen, Martin Stejskal, Josef Zeman, Martijn Westhoff, and Marc B. Parlange. 2006. Distributed fiber-optic temperature sensing for hydrologic systems. *Wat. Resour. Res.* 42, W12202 (2006).

[11] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. 2002. Wireless sensor networks: A survey. *Comput. Netw.* 38, 4 (2002), 393–422.

[12] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. 2008. Wireless sensor network survey. *Comput. Netw.* 52, 12 (2008), 2292–2330.

[13] Mohamed Hefeeda and Majid Bagheri. 2009. Forest fire modeling and early detection using wireless sensor networks. *Ad Hoc & Sensor Wirel. Netw.* 7, 3-4 (2009), 169–224.

[14] Subramania Ananda Kumar and Paramasivam Ilango. 2018. The impact of wireless sensor network in the field of precision agriculture: A review. *Wirel. Person. Commun.* 98, 1 (2018), 685–698.

[15] Biljana L. Risteska Stojkoska and Kire V. Trivodaliev. 2017. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Product.* 140 (2017), 1454–1464.

[16] Dae-Man Han and Jae-Hyun Lim. 2010. Smart home energy management system using IEEE 802.15. 4 and Zigbee. *IEEE Trans. Consum. Electron.* 56, 3 (2010), 1403–1410.

[17] J. Wang, D. Li, M. Zhou, and D. Ghosal. 2008. Data collection with multiple mobile actors in underwater sensor networks. In *28th International Conference on Distributed Computing Systems Workshops*. 216–221. DOI: http://dx.doi.org/10.1109/ICDCS.Workshops.2008.85

[18] Roland Kays, Margaret C. Crofoot, Walter Jetz, and Martin Wikelski. 2015. Terrestrial animal tracking as an eye on life and planet. *Science* 348, 6240 (2015), aaa2478.

[19] W. H. Chamberlain and H. L. Wells. 1879. *History of Yuba County, California, with Illustrations Descriptive of Its Scenery, Residences, Public Buildings, Fine Blocks and Manufactories*. Technical Report. Thompson & West.

[20] E. J. Eliason, T. D. Clark, M. J. Hague, L. M. Hanson, Z. S. Gallagher, K. M. Jeffries, M. K. Gale, D. A. Patterson, S. G. Hinch, and A. P. Farrell. 2011. Differences in thermal tolerance among Sockeye salmon populations. *Science* 332 (2011), 109–112. DOI: http://dx.doi.org/doi:10.1126/Science.1199158

[21] A. P. Farrell, N. A. Fangue, C. E. Verhille, D. E. Cocherell, and K. K. English. 2015. *Thermal Performance of Wild Juvenile Oncorhynchus Mykiss in the Lower Tuolumne River: A Case for Local Adjustment to High River Temperature*. Technical Report. Don Pedro Project.

[22] Antonio Cilfone, Luca Davoli, Laura Belli, and Gianluigi Ferrari. 2019. Wireless mesh networking: An IoT-oriented perspective survey on relevant technologies. *Fut. Internet* 11, 4 (2019). DOI: http://dx.doi.org/10.3390/fi11040099

[23] Guido R. Hiertz, Dee Denteneer, Sebastian Max, Rakesh Taori, Javier Cardona, Lars Berlemann, and Bernhard Walke. 2010. IEEE 802.11s: The WLAN mesh standard. *IEEE Wirel. Commun.* 17, 1 (2010), 104–111. DOI: http://dx.doi.org/10.1109/MWC.2010.5416357

[24] Renwei Huang, Zedong Nie, Changjiang Duan, Yuhang Liu, Liya Jia, and Lei Wang. 2015. Analysis and comparison of the IEEE 802.15.4 and 802.15.6 wireless standards based on MAC layer. In *Health Information Science*, Xiaoxia Yin, Kendall Ho, Daniel Zeng, Uwe Aickelin, Rui Zhou, and Hua Wang (Eds.). Springer International Publishing, Cham, 7–16.

[25] Martin Bor, John Edward Vidler, and Utz Roedig. 2016. LoRa for the internet of things. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*. 361–366.

[26] Pere Tuset-Peiró, Albert Anglés-Vazquez, José López-Vicario, and Xavier Vilajosana-Guillén. 2014. On the suitability of the 433 MHz band for M2M low-power wireless communications: Propagation aspects. *Trans. Emerg. Telecommun. Technol.* 25, 12 (2014), 1154–1168. DOI: http://dx.doi.org/10.1002/ett.2672

[27] Yordan Tabakov. 2017. Wizzilab: Connecting things/wizzimote. Retrieved from http://wizzilab.com/wiki/#!hardware/wizzimote.md#WizziMote.

[28] Texas Instruments. 2013. *CC430 Family User's Guide (Rev. E)*. Technical Report. Texas Instruments.

[29] Texas Instruments. 2018. *CC430F613x, CC430F612x, CC430F513x MSP430™ SoC with RF Core*. Technical Report. Texas Instruments.

[30] Maarten Weyn, Glenn Ergeerts, Luc Wante, Charles Vercauteren, and Peter Hellinckx. 2013. Survey of the DASH7 alliance protocol for 433 MHz wireless sensor communication. *Int. J. Distrib. Sens. Netw.* 9, 12 (Dec. 2013), 870430.

[31] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. 2006. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *4th International Conference on Embedded Networked Sensor Systems*. 307–320.

[32] Tijs Van Dam and Koen Langendoen. 2003. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *1st International Conference on Embedded Networked Sensor Systems*. 171–180.

[33] Tong Feng, Shilin Chen, Zhongke Feng, Chaoyong Shen, and Yi Tian. 2021. Effects of canopy and multi-epoch observations on single-point positioning errors of a GNSS in coniferous and broadleaved forests. *Rem. Sens.* 13, 12 (2021), 2325.

[34] Michael J. McGlynn and Steven A. Borbash. 2001. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*. ACM, 137–145.

[35] Jiahui Dai, Dmitry Degtyarev, Jingya Gao, Adrian Wang, Scott Burman, Ken Zillig, and Dipak Ghosal. 2020. Design and implementation of RAP-a randomized asynchronous protocol for data aggregation in wireless sensor networks. In *International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 980–986.

[36] Joseph Polastre, Jason Hill, and David Culler. 2004. Versatile low power media access for wireless sensor networks. In *2nd International Conference on Embedded Networked Sensor Systems*. 95–107.

[37] Maarten Weyn, Glenn Ergeerts, Luc Wante, Charles Vercauteren, and Peter Hellinckx. 2013. Survey of the DASH7 alliance protocol for 433 MHz wireless sensor communication. *Int. J. Distrib. Sensor Netw.* 9 (2013). DOI: http://dx.doi.org/doi:10.1155/2013/870430

[38] Kemal Akkaya and Mohamed Younis. 2005. A survey on routing protocols for wireless sensor networks. *Ad Hoc Netw.* 3, 3 (2005), 325–349.

[39] Roger Alexander, Anders Brandt, J. P. Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P. Levis, Rene Struik, Richard Kelsey, and Tim Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. DOI: 10.17487/RFC6550

[40] Dunkels Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *4th ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*.

[41] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. 2002. Wireless sensor networks for habitat monitoring. In *1st ACM International Workshop on Wireless Sensor Networks and Applications*. 88–97.

[42] Yunhao Liu, Yuan He, Mo Li, Jiliang Wang, Kebin Liu, and Xiangyang Li. 2012. Does wireless sensor network scale? A measurement study on GreenOrbs. *IEEE Trans. Parallel Distrib. Syst.* 24, 10 (2012), 1983–1993.

[43] Xufei Mao, Xin Miao, Yuan He, Xiang-Yang Li, and Yunhao Liu. 2012. CitySee: Urban CO monitoring with sensors. In *IEEE INFOCOM Conference*. 1611–1619. DOI: http://dx.doi.org/10.1109/INFCOM.2012.6195530

[44] Holger Karl and Andreas Willig. 2007. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons.

[45] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani Srivastava. 2002. Optimizing sensor networks in the energy-latency-density design space. *IEEE Trans. Mob. Comput.* 1, 1 (2002), 70–80.

[46] Athanassios Boulis, Saurabh Ganeriwal, and Mani B. Srivastava. 2003. Aggregation in sensor networks: An energy–accuracy trade-off. *Ad Hoc Netw.* 1, 2-3 (2003), 317–331.

[47] Venkatesh Rajendran, Katia Obraczka, and Jose Joaquin Garcia-Luna-Aceves. 2006. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wirel. Netw.* 12, 1 (2006), 63–78.

[48] Arvind Kumar et al. 2017. *Energy Efficient Clustering Algorithm for Wireless Sensor Network*. Ph.D. Dissertation. Lovely Professional University.
[49] Santashil PalChaudhuri and David B. Johnson. 2002. Birthday paradox for energy conservation in sensor networks. *Sleep* 9 (2002), 14mA.
[50] Qiang Wang, Andreas Terzis, and Alex Szalay. 2010. A novel soil measuring wireless sensor network. In *IEEE Instrumentation & Measurement Technology Conference*. IEEE, 412–415.
[51] Tingjun Chen, Javad Ghaderi, Dan Rubenstein, and Gil Zussman. 2018. Maximizing broadcast throughput under ultra-low-power constraints. *IEEE/ACM Trans. Netw.* 26, 2 (2018), 779–792.

Received 27 February 2021; revised 20 May 2022; accepted 24 May 2022