Wireless-Transparent Sensing

Makoto Suzuki Chun-Hao Liao Sotaro Ohara Kyoichi Jinno Hiroyuki Morikawa
Research Center for Advanced Science and Technology, The University of Tokyo, Japan
{makoto, liao, sotaro, jinno, mori}@mlab.t.u-tokyo.ac.jp

Abstract

Even after decades of efforts, wireless sensor networks (WSNs) have not gained huge momentum as people expected. The discussions with researchers and engineers in other fields make us believe that the essential problem is the lossy, unstable, and opaque nature of wireless networks rather than the power consumption or throughput problems. To tackle this key problem, we propose the concept of Wireless-Transparent Sensing that is qualified by the following conditions. First, collected data should be easy-to-use, similar to data collected by wired instruments, i.e., sampling timing should be aligned, and no data losses should occur. Second, the sensor network must be simple and responsive so that it can be managed as easily as wired instruments. Third, the system should sustainably deliver decent performance under diverse circumstances so that users do not have to choose different protocols to meet traffic demands.

To realize and verify the concept, we have developed WTSP (Wireless-Transparent Sensing Platform) with two characteristics of slot scheduling based on the concurrent transmission flooding. First, the sink node schedules slots on the fly; only several slots are determined within a schedule packet, and additional schedule or sleep packets are distributed once the distributed schedule is completed, making scheduling very flexible. Second, the scheduling is service-driven; scheduling is delegated to upper-layer services and each service directly makes a schedule, which allows the sink node to predict communication demands accurately. Using a large-scale testbed, we show that WTSP satisfies the above three conditions in many situations, and surprisingly outperforms or performs comparably to state-of-the-art collection protocols from the perspective of energy efficiency. In addition, we share experiences of three real-world applications with different requirements, namely tomato greenhouse monitoring, structure monitoring, and wireless camera networks, to reveal the practicality of the platform.

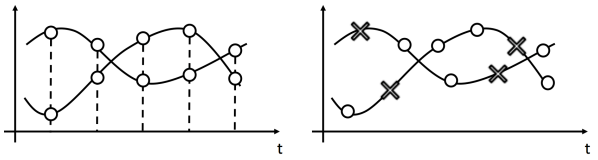


Figure 1. Data collected from a wired instrument (left), and data collected from typical WSNs (right). With WSNs, sampling timings are not aligned among the sensors, and data losses occur frequently.

1 Introduction

The performance of wireless sensor networks (WSNs) has improved significantly, yet most researchers and engineers in many fields have not widely adopted WSNs. They choose dependable wired instruments despite the fact that WSNs are easily deployable. We discussed the differences between WSNs and wired instruments with collaborative researchers of civil engineering and agriculture. Through the discussion, we found that performances such as power consumption and throughput are seldom direct concerns. They have little to complain regarding the performance of state-of-the-art protocols. What makes them hesitate to use WSNs is the lossy, unstable, and opaque nature of wireless.

To overcome this situation, we propose the concept of Wireless-Transparent Sensing that is qualified by the following three conditions:

- **Transparency for data usage:** *Sampling timing synchronization and end-to-end retransmission.* Most WSNs generate lossy and unsynchronized data as shown in Figure 1. In many application fields, analysis methods for lossy data are not established, and users are not accustomed to such sensor data. Most WSNs do not incorporate these schemes though they are beneficial in a variety of applications ranging from low data-rate data center monitoring [22] to high data-rate volcano monitoring [28].
- **Transparency for operation:** *Simplicity and responsiveness of networks.* When disorder happens, identifying and fixing the cause are difficult because mechanisms inside WSNs are hard to understand [32]. Users

International Conference on Embedded Wireless Systems and Networks (EWSN) 2017
20–22 February, Uppsala, Sweden

ISBN: 978-0-9949886-1-4

must take care of many causes such as congestion, hidden terminal problems, and inter-protocol interference [19] during the deployment and operation of typical WSNs. Also, it takes quite a while until users' treatments such as placing more relay nodes are reflected, making the deployment and operation difficult.

- **Transparency for development:** *Adaptability of data collection.* As the demands on WSNs are diverse, many protocols are optimized for different metrics such as power consumption [26] or throughput [27]. Choosing an appropriate protocol based on accurate understanding of the requirements is necessary in conventional WSNs, making it difficult to develop systems that deliver a decent quantity of sensor data.

By satisfying these conditions, *wireless becomes transparent* in the following sense. First, users can use resultant data in the similar way they are accustomed to analyzing. Second, users can deploy a sensing system and fix problems for themselves without deep consideration of the complex nature of wireless networks. Third, users can develop a sensing system without an accurate understanding of the traffic requirements and protocol characteristics.

To realize the concept, we have developed WTSP, a Wireless-Transparent Sensing Platform. WTSP achieves simplicity, end-to-end retransmission, responsiveness and adaptability by (i) adopting CTF-based networks, (ii) scheduling slots with high flexiblity, and (iii) predicting communication demand accurately. CTF stands for the very efficient flooding scheme recently proposed by Ferrari et al. [14]. CTF has many unique characteristics among existing wireless communication techniques. Specifically, CTF can achieve time synchronization with a very simple mechanism; CTF can spread a packet to the whole network within 10 ms even in a several-hop network; and CTF does not incur congestion in principle [12]. These characteristics reduce considerable complexity of the networking mechanisms.

The sink node performs *on the fly* (OTF) scheduling to ensure the flexibility. In WTSP, CTF synchronizes a network, and time is divided into slots whose lengths are several tens of milliseconds. The sink node communicates with non-sink nodes in a request-response manner continuously. The sink node requests with a control packet, which specifies the schedule of the following several slots, and non-sink nodes reply in accordance with the control packet. Right after the current schedule is completed, the sink node distributes a new control packet if further communications are necessary, or distributes sleep packets otherwise.

Moreover, in order to predict the communication demands accurately, we designed the scheduling as *service-driven*. WTSP is designed in a layered fashion and consists of a super scheduler called `ctfnetd` and four network services: versatile collection, stream-oriented dissemination to issue user commands, bulk dissemination for reprogramming, and ping for a quick reachability check. `ctfnetd` arbitrates network services, and network services have their own slot scheduler. This isolation makes it possible to design each of the schedulers independently.

The principle of the scheduling offers great benefits. For example, the sink node assigns slots in a row for maximizing collection throughput when the traffic demand is high, and instructs non-sink nodes to sleep for ensuring energy efficiency when there is no traffic demands. Also, if a packet is lost, the sink node can request a retransmission at the next schedule. We show the software architecture of WTSP, and intuitively illustrate how the scheduling contributes to achieving the above conditions in §2. §3 describes `ctfnetd`, and §4 explains the design of each network service.

Sampling timing becomes problematic when CTF is applied for high-rate sampling applications [28, 21]. Tasks are delayed for a nondeterministic time because CPU monopolization for several milliseconds is required for CTF's timing constraints, which affects the sensing quality. WTSP solves this by applying modern hardware peripherals of sensors and microcontrollers as described in §5. This enables high-accuracy sampling and wireless communication concurrently such as data collection and network management.

We evaluate WTSP using an approximately 100-node testbed [4] in §6. The representative results are as follows.

- WTSP achieves lossless data collection in various situations where 94 nodes send data every 10 s, and 25 nodes send data every 1 s, and so on. From the perspective of energy efficiency, WTSP outperforms or performs comparably to LWB [12] and ORPL [8]. In addition, the throughput of WTSP reaches 1,600 B/s; it is comparable with the state-of-the-art high-throughput reliable collection protocol [10].
- Reprogramming with WTSP is sufficiently quick; it takes 58.3 s in average to reprogram all the nodes.
- WTSP provides high-rate (100 Hz) synchronized sampling with a low jitter of 100 us during communication.

From these evaluations, we find that WTSP is suitable for not only dynamic high-throughput applications but also static low-rate applications because the easy-to-use sensor data and the built-in network management tools are obtained at the expense of a small energy overhead.

To validate the practicality of WTS and WTSP, we developed and deployed several real-world applications as demonstrated in §7; specifically, tomato growth monitoring using light intensity sensors with 84 nodes; structure monitoring of civil infrastructure in a 360 m-long elevated bridge with a six-hop, 61 node-network; and, wireless camera networks with ten nodes, which convey JPEG files.

Versatility oriented routing-based protocols such as ContikiMAC [10] and Orchestra [9], and CTF-based wireless network protocols such as LWB [12], Virtus [13], Splash [6], and Pando [7] have been proposed. We review the related works in §8, and §9 concludes this work.

2 Overview

In WTSP, time is divided into time slots long enough for one flooding over the network as in LWB and Virtus. The slot length is fixed, and one second is divided into 32 slots by default (the slot length is 31.25 ms). At most one CTF is performed in each slot, and a sender and the packet to be transmitted in each slot are determined by the sink node.

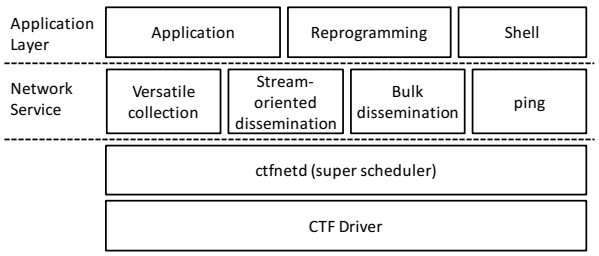


Figure 2. Software Architecture of WTSP

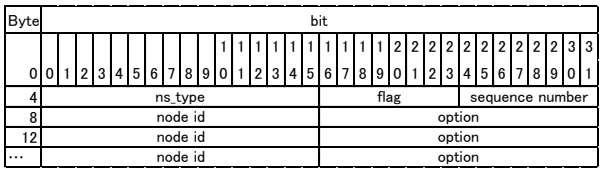


Figure 3. Control Packet Structure

Here, we show the software architecture of WTSP and the operation to illustrate that on-the-fly and service-driven scheduling on CTF is a key to achieving the conditions.

2.1 Software Architecture

Figure 2 shows the software architecture of WTSP. WTSP employs a super scheduler called `ctfnetd` on a CTF driver, and network services sit atop `ctfnetd`. Four network services are implemented. Versatile collection is used to collect data from sensor nodes. Stream-oriented dissemination is used to disseminate user commands such as the start or finish of sensing over the whole of the network. Bulk dissemination is used for large-data dissemination like wireless reprogramming. ping is used when users want to verify the reachability between a specified node and the sink node.

`ctfnetd` and network services collaboratively perform slot scheduling and power management as follows.

- **Scheduling:** `ctfnetd` of the sink node polls each network service to check the existence of communication demands in the order of priority. If there is at least one network service that has a communication demand, `ctfnetd` selects the network service whose priority is the highest. Then, the selected network service makes and distributes a control packet, which specifies the schedule of the following several slots.
- **Power management:** If there are no communication demands from any network services, `ctfnetd` of the sink node polls each network service about the next wake-up slot, and selects the earliest slot as the next wake-up slot. Then, the sink node distributes sleep packets indicating the wake-up slot and the start-of-sleep slot. If sleep packets are lost, the power consumption of the nodes severely increases. To address this problem, multiple sleep packets are sent for a single transition to the sleep mode. If a node receives at least one sleep packet, the node can enter the sleep mode.

A control packet includes information of a `ns_type` field (that indicates a service in use), `nodeid` fields (that indicate transmitters), and `option` fields (that indicate what packets should be transmitted), as shown in Figure 3. The protocols of these network services are based on request-response; the sink node requests non-sink nodes using `ns_type` and `option` fields as arguments, and non-sink nodes respond to this in the assigned slots. A control packet does not have a length field; nodes calculate the slot number using the length field of IEEE 802.15.4 standards.

Non-sink nodes operate as instructed with control packets they received from the sink node. When a node is assigned a slot, the node determines a packet to transmit in the slot using the corresponding `option` field. If the node does not have a corresponding packet, it transmits a packet called the null packet, which has no payload.

2.2 Protocol Operation

Figure 4 illustrates WTSP's operation, and how WTSP deals with packet losses, bursty traffic, and complex traffic with data dissemination in a three-node network including a sink node. Sender, Packet Type, Packet Header, and Loss Node in Figure 4 stand for a sender node in the slot, a type of the packet, the header information of the packet, and nodes that cannot receive the packet, respectively.

For illustrative purpose, the maximum entries of a control packet, time synchronization interval, and redundancy of sleep packets are set to 3, 5 s, and 2, respectively (by default, they are 10, 30 s, and 5, respectively). Figure 4 does not illustrate the synchronization process of the bootstrap signal because it is the same as for LWB and Virtus.

Basic operation: Slots 1 to 5 show the basic operation of WTSP. The sink node (Node 1) requests Nodes 2 and 3 to transmit a packet whose sequence number is 0 (*2:0, 3:0* stand for this). In response to this, Nodes 2 and 3 transmit packets in the following slots. In WTSP's data collection, the `backlog` field (B in the figure) is contained, indicating the number of packets a node wants to transmit.

The sink node understands that `backlog` of each node is zero in Slot 4, and hence does not add further schedules; it transmits sleep packets whose wake-up slot is 161. Though Nodes 2 and 3 lost a sleep packet in Slots 3 and 4, respectively, they can enter the sleep mode as they received at least one sleep packet.

Packet losses: Slots 161 to 167 show how WTSP handles packet losses. During this time, Node 2 lost a control packet from the sink node and cannot transmit a data packet in Slot 162. Besides, the sink node lost a packet from Node 3 in Slot 163. For these reasons, the sink node sends an identical control packet in Slot 164. As a result, the sink node can obtain all the data from each node.

Packets must be stored until the probability of retransmission becomes zero. If the sink node requests a packet whose sequence number is s, this means that the sink node has received packets whose sequence number is below $s-1$. As described in §4, WTSP manages the buffer without explicit acknowledgment leveraging this fact.

Mixed traffic of burst collection and data dissemination: Slots 321 to 338 show how WTSP deals with burst traffic and mixed traffic with data dissemination. As described before, a packet of WTSP's data collection has a

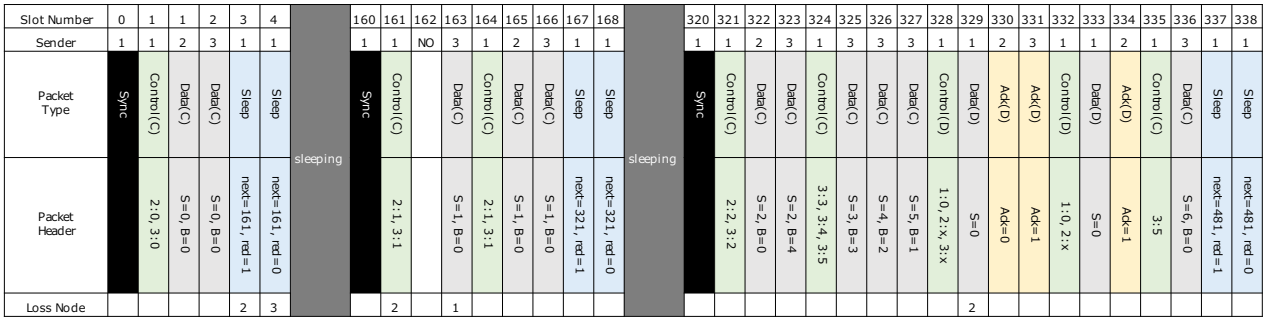


Figure 4. Scheduling Example of WTSP

`backlog` field. Using this field, WTSP efficiently deals with bursty traffic. In Slot 321, the sink node assigns a slot for each node as in Slot 1 and 161. This time, the sink node knows that the `backlog`s of Nodes 2 and 3 are 0 and 4, respectively. As a result, the sink transmits a control packet that assigns three slots for Node 3 in Slot 324.

During this collection process, a user pushes a request of data dissemination. Data dissemination has a higher priority than data collection; therefore, dissemination interrupts data collection. In Slot 328, the sink node transmits a control packet of data dissemination. This control packet means that the sink node transmits a data packet, and requests each node to transmit an acknowledgment message. In Slot 329, the sink node transmits a data packet, but Node 2 lost this packet, and Node 3 received it. Therefore, the sink node receives acknowledgments 0 and 1 from Nodes 2 and 3, respectively. The sink node makes a control packet to transmit the data packet in Slot 333 and receives an acknowledgment 1 from Node 2 in Slot 334. After the sink node successfully verifies Node 2's acknowledgment, the sink node completes the dissemination process. Then, the sink node requests Node 3 to transmit the last collection packet, and sends sleep packets in Slots 337 and 338.

Contrary to appearances, WTSP achieves relatively fast data collection and dissemination. To illustrate the efficiency of OTF scheduling intuitively, we compare the ideal performance of WTSP with the state-of-the-art. Regarding data collection, WTSP achieves approximately 2 KB/s if the maximum payload length is 64 B, and one second is divided into 32 slots. This throughput is comparable with that of the TCP on ContikiMAC (1488 B/s [10]). Though the throughput is much smaller than that of P^3[5], which is a state-of-the-art bulk transport protocol (20 KB/s), P^3 requires session initiation over the entire network, and the number of concurrent sending nodes is limited to two. Unlike P^3, WTSP can collect from any number of nodes in parallel without network-wide agreement.

Similarly, WTSP can perform fast data dissemination. WTSP can complete data dissemination over 100 nodes within several seconds. Moreover, WTSP can detect the completion of dissemination by an explicit acknowledgment check. Contrarily, state-of-the-art dissemination protocols such as Drip [31] and DIP [23] require approximately a minute to disseminate data, and users cannot verify the completion.

As described above, the OTF service-driven scheduling on CTF performs comparably to the state-of-the-art. Moreover, it can support the mixed traffic well unlike with the existing protocols.

3 ctfnetd

As described in §2, OTF scheduling tightly coupled with upper-layer services offers several desired features. Moreover, the principle allows many attractive services as shown in §4. To avoid conflicts among services and ease the development of services, we developed a super scheduler called `ctfnetd`. `ctfnetd` couples scheduling with services and decouples common network management functions from services. Specifically, `ctfnetd` delegates the generation of control and data packets to services at adequate timing through the callbacks of registered interfaces and conducts low-power control, service arbitration, and node management on behalf of network services. Here, we present the design and implementation details of `ctfnetd`.

Scheduling policy: We choose priority-based scheduling among network services rather than fair scheduling. This is beneficial for most WSNs as described in §2 since a user command can preempt other communications. The priority of network services is specified as the order of service registration to `ctfnetd`. To improve the user responsiveness, stream-oriented dissemination, bulk dissemination, ping, and data collection are registered in this order.

Delegation to services: Control packets and some packet fields must be set just before they are transmitted. For example, in a versatile collection service, the sink node cannot obtain the right backlog information for burst packet sending if a packet header is generated when the `send` API is called. The `backlog` field of the first packet becomes 1 even if the node has more packets in the buffer. In addition, in the stream-oriented dissemination service, the sink node cannot obtain the right acknowledgment if a packet header is generated at the reception of a control packet because the data packet has not been transmitted.

Toward this, `ctfnetd` callbacks data and control packet generation function of each service right before the assigned slot. For example, a versatile collection service has its own packet buffer and just accumulates packets in the buffer when the `send` API is called. In the callback function, a packet

```
slots = min(CONTROL_MAX_SLOTS,  // (1)
            get_next_sync() - get_current_slot());
for(i=0; i<service_num; i++){
 slots = services[i].make_control(ctrl, slots); // (2)
 if(slots != 0 && slots != CTFNETD_NULL_SCHED) {
   distribute_control(ctrl, slots);  break; // (3)
 }
}
```

Figure 5. Pseudocode of Slot Scheduling

```
slots = MAX_SLEEP;
for(i=0; i<service_num; i++){  // (4)
 slots = min(slots, services[i].next_wakeup())
}
for(i=0; i<SLEEP_REDUNDANCY; i++){
 distribute_sleep(slots, SLEEP_REDUNDANCY-i); //(5)
}
```

Figure 6. Pseudocode of Low-power Control

corresponding to the sequence number is chosen, the packet header is set, and then the packet is pushed into the CTF driver.

Separation of synchronization: Time synchronization is an essential part of WTSP and is implemented directly in `ctfnetd` rather than being implemented as a service. In the context of OTF scheduling, the interval of control packets is highly variable unlike in the case of LWB or Virtus. Synchronization with fluctuated-interval scheduling packets makes synchronization accuracy unstable. We therefore separated synchronization so that the scheduling packet is distributed with a uniform interval to ensure synchronization accuracy.

Node management: Wireless channel fluctuations, battery depletion, and software bugs may make communication impossible. Assignments of slots for these nodes result in a severe increase of power consumption and a decrease of throughput. `ctfnetd` notifies network services of the death of a node if the node's response does not reach the sink node several times in a row. To this end, `ctfnetd` maintains the number of consecutive packet losses of each node, and notifies each network service if the counter exceeds a predefined threshold.

Implementation detail: Each network service registers five function pointers as below.

- `make_control(control_t* buf, uint8_t max)`: `ctfnetd` callbacks this function of each network service in the sink node before it sends a control packet.
- `make_packet(uint16_t opt, uint8_t* buf, uint8_t len)`: `ctfnetd` callbacks this function in a sender node before the assigned slot.
- `on_recv(uint8_t* buf, uint8_t len)`: `ctfnetd` callbacks this function of a service when a packet linked to the service is received.
- `next_wakeup(void)`: `ctfnetd` callbacks this function of each service to calculate the next wake-up slot.
- `on_member_changed(uint16_t nodeid, bool onoff)`: When `ctfnetd` detects the joining or leaving of a node, it callbacks this function to notify the information.

`ctfnetd` uses the registered interfaces as shown in Figure 5 and Figure 6. First, the number of scheduling slots is determined by the maximum entries of a control packet and a nonconflict condition against time synchronization (Figure 5(1)). Then, `ctfnetd` polls network services about the existence of communication demands in the order of priority through `make_control()` (Figure 5(2)). If at least one network service has demands, the sink node distributes a control packet generated by the network service (Figure 5(3)).

When all network services do not have any demands, the sink node calculates the next wake-up slot through `next_wakeup()` (Figure 6(4)), and distributes sleep packets multiple times as described in §2 (Figure 6(5)). Note that time synchronization continues to be conducted in the sleep mode because synchronization is not a network service on `ctfnetd`. By default, the duty cycle required for time synchronization is only 0.03% because the time synchronization interval is 30 s and the duration of radio-on time of time synchronization is approximately 10 ms.

4 Network Services

`ctfnetd` is expressive enough to implement various efficient protocols on it. When designing network services, not only performance but also the complexity of computation, memory usage, and usability should be considered. Here, we describe the design of each network service.

4.1 Versatile Collection

Two requirements need to be considered while designing an efficient data collection protocol. The first is how to obtain the traffic demand from each node to achieve adaptability. In LWB, users specify the traffic pattern using a single parameter called inter-packet interval (IPI). However, the users must explicitly consider traffic patterns with this approach. In addition, some applications such as wireless camera networks and structure monitoring generate busty traffic. In these applications, the single parameter of IPI alone cannot describe the traffic well.

The service introduces a `backlog` field that shows the number of packets to be collected, as ContikiMAC [10] and Hui's MAC [18] use a pending bit. Using this information, WTSP can assign slots adaptably without forcing users to specify the traffic pattern.

The second requirement is acknowledgment messages for end-to-end retransmission. Senders must retain messages that have the probability of retransmission. Virtus [13] adopts explicit acknowledgment messages after a reception.

WTSP adopts implicit acknowledgments for communication simplicity. Specifically, if a packet whose sequence number is s is requested, this means that the sink node received packets whose sequence number is less than s. More specifically, each non-sink node maintains two variables of `acked_seq` and `pushed_seq`. When non-sink nodes receive a control packet including the assignment for themselves, `acked_seq` is set to $s-1$, where s indicates the requested sequence number (if a control packet includes multiple requests for a node, s means the lowest sequence number among them). `pushed_seq` indicates the latest packet's sequence number pushed into the buffer. The network service

```
while(sent_num < packets){
  xmem_read(addr, buf, 64);
  while(!wtsp_collect_send(TYPE_DATA, buf, 64)){
    PROCESS_YIELD_SLOTS(1);
  }
  addr+=64; sent_num++;
}
```

Figure 7. Application Code of Burst Data Collection

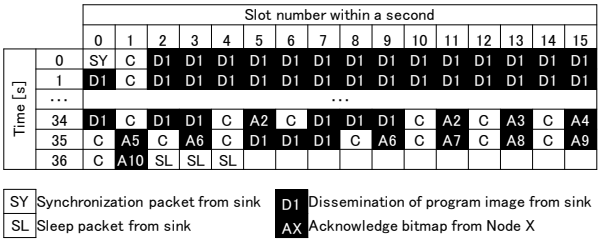


Figure 8. An Example of Scheduling of Bulk Dissemination

does not accept a packet from applications if `pushed_seq - acked_seq` is larger than the buffer size.

The limitation of the method is that the packets received by the sink node remain in the buffer of each sender until the next request; hence, the number of packets that can be accepted is reduced. However, this does not cause any throughput degradation problem if the buffer length is more than the twice the entry number of a control packet because sender nodes can complement packets to the buffer just after the next request.

This protocol does not support selective acknowledgment to minimize the complexity of computation and communication. Therefore, out-of-order packets are discarded.

With these mechanisms, users can easily develop an application that performs burst transportation. Figure 7 shows an application code that collects data from the flash memory with 100% end-to-end reliability. Note that PROCESS_YIELD_SLOTS() is an extension of PROCESS_YIELD() of the Contiki OS, and the program execution is resumed after the specified slots.

4.2 Stream-oriented Dissemination

A consideration in the design of a data dissemination protocol is whether the popular one-way (no-ack) scheme should be adopted or not. Existing dissemination protocols such as Drip [31], DIP [23], and LWB [12] (in the evaluation of a mobile sensing scenario) adopt a one-way scheme based on an eventual consistency model. Eventual consistency is ensured simply by intermittently transmitting data one-way, and this requires a little management information. However, this approach has the disadvantage that the completion of dissemination cannot be detected. Continuing the process impairs power efficiency, and interferes with other network services.

We adopted a two-way, explicit acknowledgment scheme because WTSP can efficiently perform end-to-end communication. In WTSP, an explicit acknowledgment is requested after the sink sends data packets. The sink retransmits the packets that are not acknowledged.

Superficially, collecting acknowledgment from every node appears inefficient. However, as the frequency of dissemination is not very high (at most several times a day), this scheme is more efficient than the always-on one-way scheme for most applications.

Verifying the presence of acknowledgment from every node is computationally heavy because this computational complexity increases linearly with the number of nodes in the network. Toward this, a variable of `consistent_nodes` is maintained. If a new packet is pushed to the sink node's buffer, the variable is set to 0. `consistent_nodes` is incremented when acknowledgment from each node becomes consistent with the latest pushed packet. `consistent_nodes` is decremented in `on_member_changed()` if the dead node is already consistent when the sink node detects the death of a node. As a result, exceptional situations such as sudden death and joining of a node can be supported with low complexity. Note that the service also does not support selective acknowledgment, and hence out-of-order packets received at non-sink nodes are discarded.

4.3 Bulk Dissemination

Stream-oriented dissemination is not suitable for large-data dissemination such as network reprogramming because it does not support selective acknowledgment. Bulk dissemination adopts bitmap acknowledgment for efficient transportation. There are two ways to acknowledge the reception at every node: one-by-one acknowledgment or network-wide acknowledgment. In one-by-one acknowledgment, the sink node requests an acknowledgment of a single node, and moves to the next node after every packet is successfully delivered to the node. In network-wide acknowledgment, the sink node requests acknowledgment of every node, and determines to retransmit packets using logical AND, and repeats this process until all packets are successfully delivered to every node.

One-by-one acknowledgment is less efficient for acquiring acknowledgments because one control packet is required to obtain an acknowledgment. Contrarily, network-wide acknowledgment is less efficient for requesting retransmission if the correlation of the packet losses exists (one retransmission needs multiple acknowledgments). In WSNs, since correlation of packet losses are common [6], we adopted one-by-one acknowledgment.

An example of scheduling of reprogramming is illustrated in Figure 8. In this figure, one second is divided into 16 slots for illustrative purposes. As shown in this figure, the sink node distributes packets in a row and then requests an acknowledgment from each node one-by-one.

4.4 ping

When users deploy WSNs, they want to quickly verify the reliability between non-sink nodes and the sink node to determine whether more relay nodes are required. Moreover, users want to estimate the maximum length of a data packet of the network because the packet length is constrained by

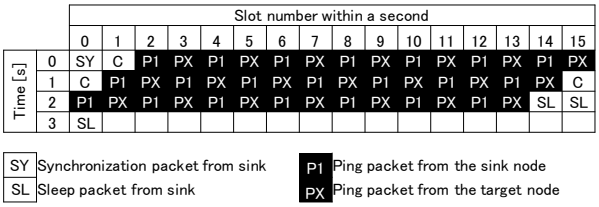


Figure 9. An Example of Scheduling of ping

the hop count of a network and the slot length. It is because CTF must be completed within a slot and the hop count of a network cannot be clear before deployment. The ping service is useful for these purposes.

To realize the ping service on `ctfnetd`, it is necessary to consider the effect of loss of control packets. The measured packet reception rate (PRR) becomes different from the actual rate since non-sink nodes cannot transmit packets if a control packet is lost. For this reason, non-sink nodes should manage the actual number of transmissions and communicate the information to the sink node, unlike the ICMP ping. In addition, to measure the PRR after the ping start, non-sink nodes should reset these statistics every time the sink node initiates the ping service.

Toward this, the ping packet has three fields, *seq*, *tx_cnt*, and *rx_cnt*, where *seq* indicates how many times the ping service has been initiated, and *tx_cnt* and *rx_cnt* are the numbers of times of transmission and reception of ping packets in the service execution, respectively. Non-sink nodes maintain the last received *seq*, and if they receive a packet whose *seq* is newer than that, they reset *tx_cnt* and *rx_cnt*. Upward PRR is obtained by dividing *rx_cnt* of the sink node by *tx_cnt* of the target node. Similarly, downward PRR can be obtained by dividing *rx_cnt* of the target node by *tx_cnt* of the sink node. Note that *tx_cnt* and *rx_cnt* of the target node are not always the same because the target node cannot transmit packets when it does not receive a control packet.

Figure 9 illustrates an example of scheduling of ping. As shown, the slots are assigned alternately, and the process is completed within several seconds.

5 Synchronized Sampling

Both CTF and sensor sampling are timing-critical tasks in applications that require high-frequency sampling. To achieve CTF, sensor nodes must forward packets immediately after the reception. If packet forwarding is deferred, the concurrent transmission becomes destructive. Besides, sensor sampling must be performed just at the desired sampling timing. If sampling is deferred, the quality of sensing decreases. Toward this, Kim et al. completely turned the radio off to ensure the accuracy of sampling timing [21]. This approach can remove sampling jitter due to interruptions but must stop synchronization. Therefore, time synchronization errors accumulate, and furthermore make data collection in real-time impossible.

To meet the two timing constraints, we leveraged two characteristics of modern sensors and microcontrollers. First, recent accelerometers such as Analog Devices ADXL362 and Kionix KX022 have a FIFO module and an external trigger function. When the trigger pins are asserted, these accelerometers sample acceleration and store the samples in their FIFO modules. Second, almost all modern microcontrollers including MSP430 and Cortex-M series have compare-match peripherals. When a timer is configured in the compare mode, the timer module toggles GPIO pins without the intervention of software. By applying Compare Out of a microcontroller to an accelerometer's trigger pin, accurate sampling can be done.

We used a 32,768 Hz crystal to generate sampling timing to keep a microcontroller off while performing high-frequency sampling. Furthermore, we reduced the sampling timing difference among the nodes to the crystal oscillator's resolution by applying Bresenham's line algorithm [1].

6 Evaluation

In this section, we evaluate WTSP experimentally using an implementation on TelosB. First, the collection service is evaluated from the perspectives of efficiency, reliability, and throughput. Next, we show the evaluation of stream-oriented and bulk dissemination using an implementation of real reprogramming service. Lastly, we investigate the accuracy of synchronized sampling.

All experiments except synchronized sampling were conducted in the Indriya testbed [4], and node 1 was set as the sink. We used 92 to 94 nodes depending on available nodes in the testbed.

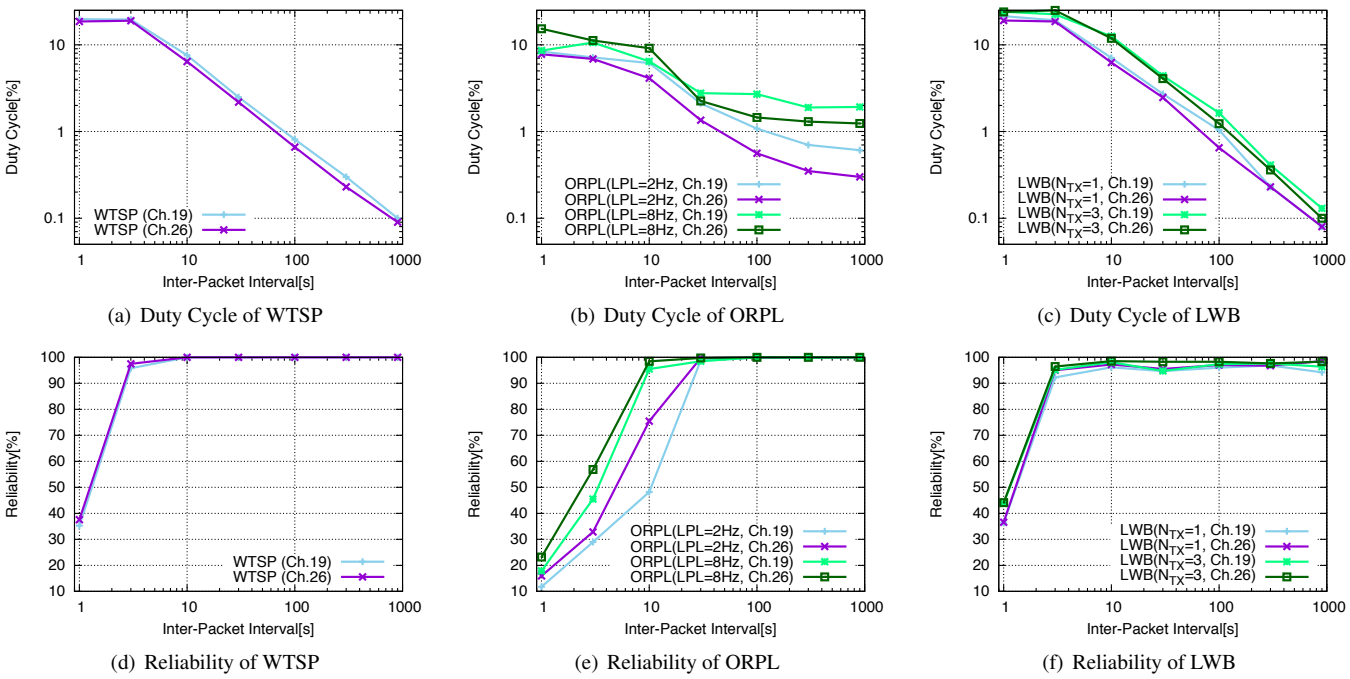
6.1 Collection: Efficiency and Reliability

First, we evaluate the protocols in periodic traffic, a typical traffic demand in WSNs. We verify WTSP's energy efficiency by varying traffic demands and channels. Then, we discuss robustness against interference by using a Wi-Fi coexisting channel, and the overhead of WTSP's scheduling by comparing it with LWB.

Protocols: We demonstrate the performance of WTSP by comparing it with two existing protocols, namely ORPL and LWB, the state-of-the-art asynchronous MAC-based protocol and CTF-based protocol, respectively. ORPL is configured as ORPL-2 (2 Hz LPL) and ORPL-8 (8 Hz LPL).

LWB is configured as LWB-1 and LWB-3, $N_{TX} = 1, 3$, respectively, where N_{TX} is a redundancy parameter (maximum number of times a node transmits during a CTF). This parameter plays an important role in LWB. Increasing N_{TX} makes the communication reliable at the expense of power consumption. In contrast, the number of redundant transmissions can be minimized in WTSP because end-to-end retransmission helps reduce packet losses. The end-to-end mechanism retransmits only the lost packets, improving energy efficiency while ensuring 100% reliability in most situations. Note that the parameter is not clearly shown in [12, 8].

Traffic patterns: We investigated energy efficiency and reliability for periodic traffic demands by varying the IPI (1 s, 3 s, 10 s, 30 s, 100 s, 300 s, and 900 s), and the channels (Ch19 and Ch26). The experiments were conducted for 30 min when IPI $\leq$ 30 s, and for 2 h when IPI $\geq$ 100 s. We



(a) Duty Cycle of WTSP (b) Duty Cycle of ORPL (c) Duty Cycle of LWB

(d) Reliability of WTSP (e) Reliability of ORPL (f) Reliability of LWB

Figure 10. Evaluation Results of Periodic Traffic. WTSP achieves higher reliability in any IPI and channel. WTSP achieves better efficiency than ORPLs when IPI is large, and reliability is higher when IPI is small. WTSP and LWB are less vulnerable to interference. WTSP achieves 100% reliability in both channels when IPI $\geq$ 5.

used a transmission power of 0 dBm, and the payload size was set to 15 B.

Result: Figure 10 shows the results of duty cycle and reliability.

Comparing with ORPL, for light periodic traffic (IPI $\geq$ 300 s compared with ORPL-2, IPI $\geq$ 30 s compared with ORPL-8), the results show that WTSP's efficiency is higher. For example, at IPI=900 s, the duty cycle of WTSP, ORPL-2, and ORPL-8 are 0.09%, 0.30%, and 1.24%, respectively. WTSP's reliability is consistently higher than or equal to ORPL's. When LPL frequency is increased for greater reliability in ORPL, the energy consumption also increased.

In heavy traffic (IPI $\leq$ 10 s), WTSP's duty cycle is higher than those of ORPL-2 and ORPL-8. Still, WTSP's reliability is higher than those of ORPLs. For example, when IPI=3 s, the reliability of WTSP, ORPL-2, and ORPL-8 are 97.49%, 32.84%, and 56.84%, respectively.

When comparing with LWB, the trends between these protocols are consistent. Figure 11(a) and Figure 11(b) show the reliability and the duty cycle in case of IPI=100 s, respectively. From the reliability point of view, WTSP always provides 100% reliability as long as the traffic loads do not exceed the network capacity (IPI $\geq$ 10 s). On the contrary, LWB-1 and LWB-3 fail to achieve 100% reliability even when the traffic load is lower than the capacity. Specifically, when Channel 26 is used, the reliability of LWB-1 and LWB-3 are 97.2% and 98.75%, respectively. From the efficiency point of view, WTSP achieves comparable performance with LWB-1, and significantly outperforms LWB-3. The duty cycle of LWB-1, LWB-3, and WTSP are 0.56%, 1.23%, and 0.66%, respectively. That is, LWB-3 consumes approximately two times energy while providing less reliability than WTSP.

Robustness against Interference: In any settings and protocols, the duty cycle increases and reliability decreases when Channel 19 is used. There is a trend that ORPL suffers more from interference than WTSP and LWB. This trend is because asynchronous MAC suffers from a false positive of clear channel assessment (CCA) checks. By contrast, WTSP and LWB do not suffer from the problem because they do not perform CCA.

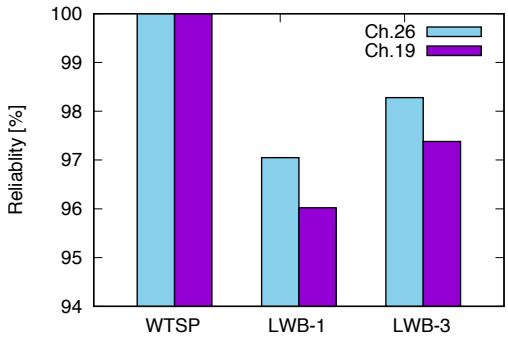
6.2 Collection: Throughput

We investigate the throughput by setting the IPI to 1 s, payload length to 64 B, and varying the number of packet generation nodes from 1 to 70 out of 94 nodes, with Channel 26. The result is shown in Figure 12. WTSP achieves 100% reliability when the number of traffic generating nodes is less than 25, and the goodput does not drop below 1,600 B/s after saturated. On the other hand, ORPL's reliability is not 100% even when only two nodes generate packets, and always lower than that of WTSP.

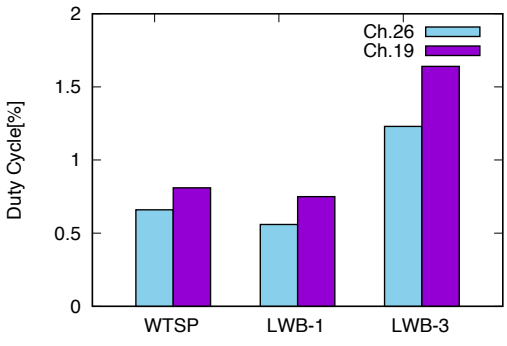
This result shows a characteristic difference between WTSP and ORPL. WTSP outperforms when multiple nodes generate traffic since WTSP does not suffer from congestion and the hidden terminal problem. On the contrary, ORPL's performance significantly deteriorates due to these problems.

6.3 Reprogramming

We implemented a real reprogramming service using stream-oriented and bulk dissemination, and evaluate the



(a) Reliability.



(b) Duty Cycle.

Figure 11. Comparison with LWB. WTSP achieves 100% reliability while LWB cannot. WTSP's duty cycle is almost the same as LWB-1's. LWB-3 consumes power approximately twice as high as that of WTSP and LWB-1. Note that the y-axis of Reliability does not begin from zero.

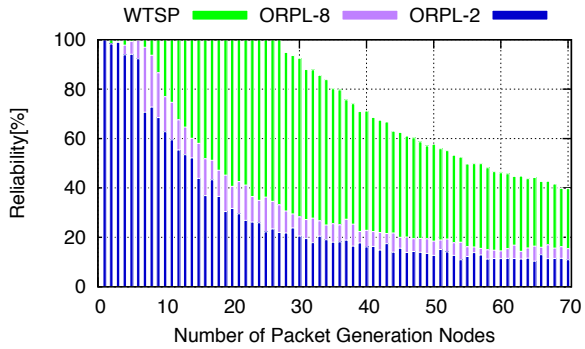


Figure 12. Number of Sending Nodes vs Reliability

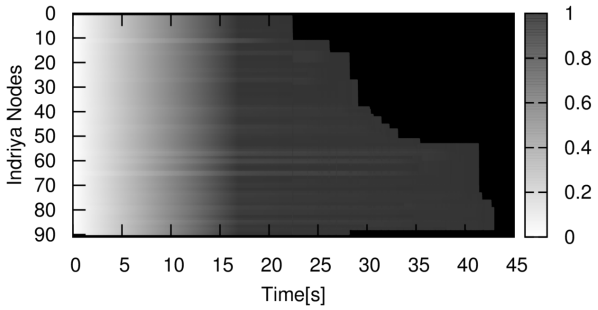


Figure 13. Reprogramming Progress of Each Node

performance. A sink node reads its own 32,704 B program image from the flash memory and divides it into 511 bulk dissemination packets whose payload lengths are 64 B. The last packet has a 32 B payload that consists of interrupt vectors located at the end of program memory. After the completion of bulk dissemination, the sink transmits a packet that instructs nodes to start reprogramming through stream-oriented dissemination. Each node begins reprogramming its own flash memory 10 s after it receives the dissemination packet.

We conducted this three times in the Indriya testbed. On average, the bulk dissemination time was 43.4 s and the reprogramming time was 14.9 s. Figure 13 shows the progress of each node of bulk dissemination. As pointed out in [7], the long tail problem occurs. Besides, WTSP's dissemination is not the fastest (the fastest ever reported is 11.4 s [7]). Even so, we believe the reprogramming time of 58.3 s is tolerable for many applications. Note that Deluge [17] requires several hundred seconds to complete reprogramming [6].

6.4 Synchronized Sampling

We evaluate the sampling timing error between four nodes that are placed nearby using a logic analyzer for 10 min. The synchronization error is defined as the difference between the sampling timing of Node 1 and that of each of the other nodes. Figure 14 shows the result. The maximum absolute error is 91 us, RMS error is 22 us, and the average of the timing difference is -0.4 us.

This result shows that delays due to task scheduling are completely removed. If there is a task scheduling delay, there must be sporadic large jitters [21]. The synchronization error is mainly due to the resolution of the clock as expected.

7 Applications

We have developed wireless sensor network applications using WTSP and performed deployment tests. Here, we present three applications with different requirements, and share deployment experience because real-world application examples are limited in literature though many studies have been conducted based on CTF.

The first application is tomato growth monitoring with 84 nodes placed in a relatively small greenhouse of dimensions 10 m $\times$ 8 m $\times$ 3 m (high-density deployment), and the second is structure monitoring with 61 nodes in a 360 m long, 12-span elevated bridge (harsh radio propagation conditions). The third is a wireless camera network with 10 nodes in a 35 m $\times$ 12 m laboratory. These three applications share communication infrastructure though they have very different demands. All that the developers have to do is

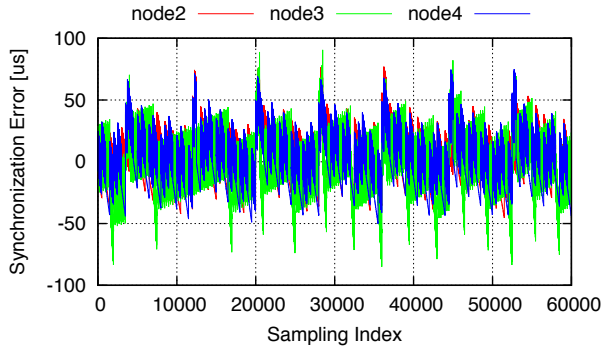


Figure 14. Synchronized Sampling Accuracy

Figure 15. Deployment for Tomato Growth Monitoring

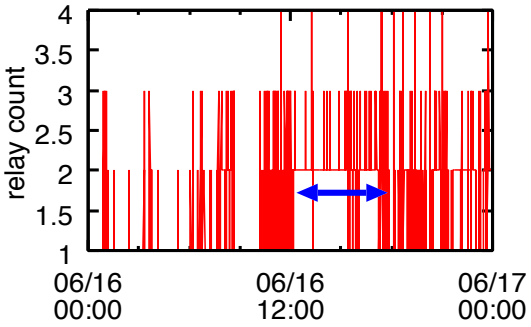


Figure 16. Link Fluctuation

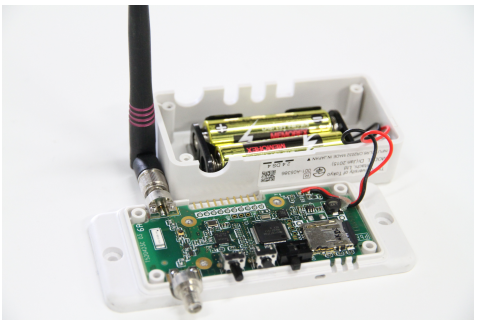


Figure 17. Structure Monitoring Node

to implement the application code. By leveraging WTSP's features, these applications were implemented with a little effort.

7.1 Tomato Growth Monitoring

In this application, we aim to capture the daily variation of the leaf area index (LAI) of foliage for several months to observe plant growth. The LAI is one of the important indicators of plant growth, and is defined as the averaged surface areas of all leaves per unit area. The LAI can be estimated from the light transmittance of the foliage based on the Monsi-Saeki model. Because each sensor output is synchronously sampled over the entire network, light transmittance can be simply calculated by dividing light intensity from under-foliage nodes by that from the over-foliage node.

For estimating the LAI of each foliage in a tomato greenhouse, with a relatively small size (10 m $\times$ 8 m $\times$ 3 m), we deployed 84 sensors for two months (Figure 15). Each sensor node equips a TelosB mote and senses light intensity every 30 s. The average duty cycle is 0.94%, and every packet is successfully collected without any losses owing to the end-to-end retransmission scheme.

We recorded relay counts to the sink node of each packet. We found that link fluctuations happened even in this small greenhouse. Figure 16 shows a time series of relay counts from a certain node. The arrow in Figure 16 indicates the time period when the node rarely communicates with the sink node directly for four hours even though it was possible at the time of deployment. Even in this case, WTSP can seamlessly adapt to the fluctuations because of CTF.

7.2 Structure Monitoring

We developed a structure monitoring node that includes Analog Devices ADXL362, TI MSP430F1611, and TI CC2520 as shown in Figure 17. We deployed 61 sensor nodes over twelve viaducts in an urban expressway bridge. Five sensors are deployed in each span, as shown in Figure 18. Each span is approximately 30 m, and the total length of this deployment field is approximately 360 m.

We conducted a data collection experiment in the field. In this experiment, the theoretical throughput is 1024 B/s because we set slots per second as 16 to ensure the margin of hop count. We measured acceleration over 40 s and collected the data wirelessly. Figure 19 shows the time series of throughput. As shown in the figure, the average throughput is approximately 800 B/s. Data from 35 nodes were successfully collected without any losses. The other nodes had stopped sending data before the completion of the experiment. After investigating the collected data, we found that a software bug in the buffer management part in the application layer caused this problem. Still, we could obtain data from these nodes without any loss before they stopped sending.

Figure 20 shows the average hop count from each node. The average single-hop distance of 60 m is ensured in all areas. This indicates that CTF does not show severe performance deterioration in a network of this size. Though one-hop distance is much shorter than that in the line-of-sight environment (approximately 400 m), this performance deterioration is due to the harsh radio environment. In this experiment, the sensor installation locations are the narrow and constrained spaces underneath the bridge slab surrounded by steel and concrete bridge members.

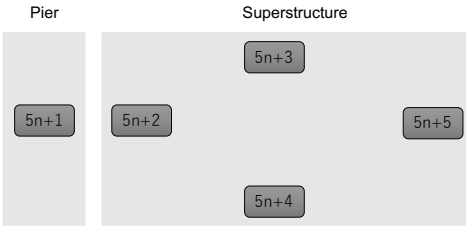


Figure 18. Schematic Plan View of Each Span. 61 sensors are deployed over 12 spans

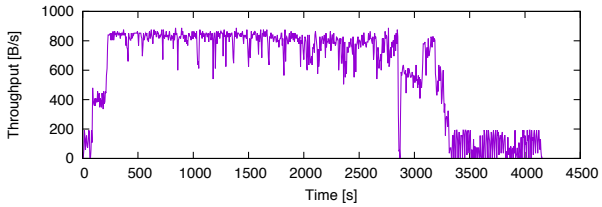


Figure 19. Time series of Throughput

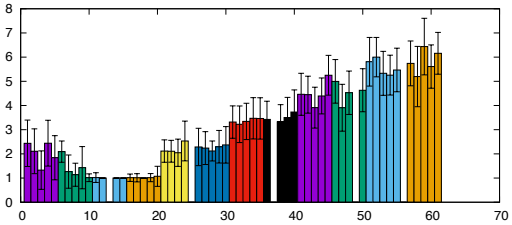


Figure 20. Average Hop Count of Each Node

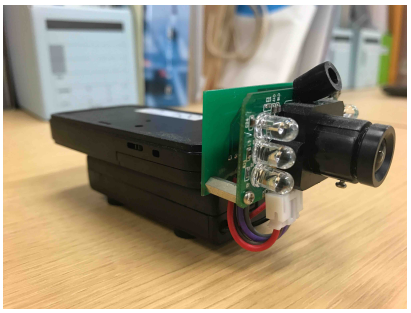


Figure 21. Wireless Camera Node

7.3 Wireless Camera Network

We connected a LinkSprite JPEG color camera [29] to a wireless sensor node that embeds an Epic core module [11] as shown in Figure 21. The camera module takes a photo in response to a request and provides a JPEG file through UART. Epic Core modules are programmed so that they take a photo every hour, and transport the resultant JPEG file to the sink node. Since WTSP supports in-order data collection with end-to-end retransmission, it can collect JPEG files over wireless networks. One of the important benefits, which we did not expect, is that WTSP can transparently convey structured, loss-intolerant data such as a JPEG file.

8 Related Work

Most of the existing protocols support specific traffic demands. CTP [16], TSMP [26], and Dozer [2] have been developed for periodic data collection, whereas Flush [20] and PIP [27] are used for fast, reliable data collection. Protocols that provide 100% reliability have severe limitations. These protocols do not support duty cycling [27, 20, 25, 22], or incur a large delay of several hours [24].

Several protocols have been proposed to support various traffic types with a single mechanism. Specifically, Duquennoy et al. proposed a MAC layer that supports both burst and light traffic. Moreover, they showed that the TCP works on the proposed MAC layer, and it achieves 100% reliability efficiently [10]. Orchestra [9] tries to facilitate both low delay and high reliability by local scheduling without a global view. However, the performance of these protocols would deteriorate severely when several nodes generate traffic simultaneously because ContikiMAC is contention-based and Orchestra does not have a global view. WSNs that perform synchronized sampling generate bursty traffic; hence, these protocols do not suit our purpose.

LWB [12] shows that the performance of collection on CTF is promising. Though LWB considers various traffic patterns including one-to-many, many-to-one, and many-to-many, its target is periodic best-effort traffic. Other proposals consider collection using concurrent transmission, such as CX [3], P^3 [5], and Choco [30], but they do not consider the rich upper-layer services unlike WTSP.

Virtus [13] is the most closely related work to WTSP. Virtus elegantly abstracts the acknowledgment scheme, and a single mechanism achieves reliable one-to-many, many-to-one, and many-to-many communication. However, the throughput and delay of Virtus are adversely affected when packet losses occur because Virtus does not adopt OTF scheduling; the retransmissions are performed in the next round. In fact, the authors experimentally show that the delay dramatically increases up to the extent of several ten seconds with packet losses. We believe the delay due to packet losses should be minimized if Virtus adopts OTF scheduling.

Several works exist that exploit CTF to data dissemination, such as Splash [6] and Pando [7]. These protocols improve the performance of data dissemination dramatically (20 $\times$ or more). However, the integration is not discussed well. There is no description on how to start or finish the protocol in [6]. Pando tries to safely detect the completion by using the RSSI, and this leads to false positive and false negative problems [15]. To the best of our knowledge, WTSP is the first to integrate diverse upper-layer services efficiently.

9 Conclusion

In this paper, we introduced the concept of Wireless-Transparent Sensing and presented the design and implementation of WTSP based on the concept. WTSP leverages on-the-fly and service-driven scheduling on concurrent transmission flooding-based networks to provide diverse network services efficiently. We confirmed the performance of WTSP through the Indriya testbed and the practicality through the diverse real-world applications.

10 Acknowledgment

We would like to thank the anonymous reviewers and our shepherd, Mike Chieh-Jan Liang, for their invaluable comments. We would like to thank Tomonori Nagayama and Naoya Fukuda for their insightful comments on the requirements for wireless sensor networks based on the profound knowledge in their research fields. We would also like to thank Metropolitan Expressway Company Limited for providing experiment fields and advices. Special thanks to Yasutaka Yamashita and Yuki Katsumata for their dedicated efforts on the development and evaluation of the platform. This work was partly supported by Council for Science, Technology, and Innovation, Cross-ministerial Strategic Innovation Promotion Program, "Maintenance, Renovation, and Management of Infrastructure".

11 References

[1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Syst. J.*, 4(1):25–30, 1965.

[2] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN)*, pages 450–459, 2007.

[3] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali. Forwarder selection in multi-transmitter networks. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 1–10, 2013.

[4] M. Doddavenkatappa, M. Chan, and A. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In *Proceedings of the Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, pages 302–316, 2011.

[5] M. Doddavenkatappa and M. C. Chan. P3: A practical packet pipeline using synchronous transmissions for wireless sensor networks. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 203–214, 2014.

[6] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 269–282, 2013.

[7] W. Du, J. C. Liando, H. Zhang, and M. Li. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 365–378, 2015.

[8] S. Duquennoy, O. Landsiedel, and T. Voigt. Let the tree bloom: Scalable opportunistic routing with orpl. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 2:1–2:14, 2013.

[9] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 337–350, 2015.

[10] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy links, low power, high throughput. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 12–25, 2011.

[11] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler. A building block approach to sensornet systems. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 267–280, 2008.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 1–14, 2012.

[13] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Virtual synchrony guarantees for cyber-physical systems. In *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 20–30, 2013.

[14] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 73–84, 2011.

[15] R. Flury and R. Wattenhofer. Slotted programming for sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 24–34, 2010.

[16] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, 2009.

[17] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 81–94, 2004.

[18] J. W. Hui and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 15–28, 2008.

[19] J. Il Choi, M. A. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 267–280, 2009.

[20] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: A reliable bulk transport protocol for multihop wireless networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys)*, pages 351–365, 2007.

[21] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN)*, pages 254–263, 2007.

[22] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. Racnet: A high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 15–28, 2009.

[23] K. Lin and P. Levis. Data discovery and dissemination with dip. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 433–444, 2008.

[24] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks (IPSN)*, pages 421–432, 2008.

[25] J. Paek and R. Govindan. Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks. *ACM Trans. Sen. Netw.*, 7(3):20:1–20:45, 2010.

[26] K. S. J. Pister and L. Doherty. Tsmp: Time synchronized mesh protocol. In *Proceedings of the IASTED International Symposium on Distributed Sensor Networks (DSN)*, 2008.

[27] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale. Pip: A connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 15–28, 2010.

[28] W.-Z. Song, R. Huang, M. Xu, A. Ma, B. Shirazi, and R. LaHusen. Air-dropped sensor network for real-time high-fidelity volcano monitoring. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 305–318, 2009.

[29] sparkfun. LinkSprite JPEG Color Camera. `https://www.sparkfun.com/products/11610`.

[30] M. Suzuki, Y. Yamashita, and H. Morikawa. Low-power, end-to-end reliable collection using glossy for wireless sensor networks. In *Proceedings of the IEEE 77th Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2013.

[31] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks (EWSN)*, pages 121–132, 2005.

[32] M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: A new metric for protocol design. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 73–86, 2007.