5

Sensor Networks Architectures and Protocols

Omprakash Gnawali and Matt Welsh

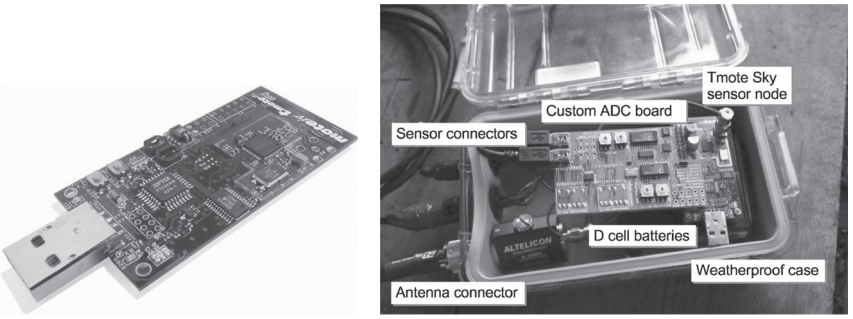
5.1 Introduction

Wireless sensor networks (WSNs) are an important emerging class of embedded distributed systems that consist of low-power devices integrating computation, sensing, and wireless communications. WSNs have been deployed for a wide range of applications, including monitoring microclimates in redwood forests (Tolle et al. 2005), collecting seismic signals from active volcanoes (Werner-Allen et al. 2006), sniper detection in urban settings (Simon et al. 2004), and tracking wildlife (Zhang et al. 2004).

One of the most popular WSN node platforms is the Telos node platform (Polastre et al. 2005a), shown in Figure 5.1. The Telos incorporates a low-power microcontroller (TI MSP430) with 10 KB of SRAM and 48 KB of program ROM; a low-power radio (Chipcon CC2420) that supports the IEEE 802.15.4 standard; and 1 MB of on-board flash memory. Various sensors can be attached to the board; a standard set includes light, temperature, and humidity sensors. An external connector provides digital and analog I/O ports that can be used to mate the node to a wide range of sensors and other devices. The USB connector is used to program the node when plugged into a host, as well as to provide a serial interface. This allows the node to act as a USB wireless transceiver when attached to a *base station* that collects data from and controls the network.

WSN platforms are designed from the ground up for low-power operation. The Telos consumes approximately 41 mW when the CPU and radio are active, but can drop down to a low-power idle state consuming less than 6 μW. Depending on the duty cycle, the device can potentially operate for months on a pair of alkaline AA batteries. However, this low power consumption comes at the cost of extreme limitations on computational horsepower, memory capacity, and radio bandwidth. The CC2420 radio operates at a PHY rate of just 250 Kbps, but

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press



(a) (b)

Figure 5.1. Wireless sensor node platforms.

link throughput (including framing and control message overhead) is less than 100 Kbps in practice, and may be much less under lossy conditions. Moreover, the radio consumes far more power than the CPU, mandating careful control over the radio listen, receive, and transmit modes.

Therefore, network protocols for sensor networks must be designed to operate with lossy and low-throughput links and limited memory for storing routing tables and other state. Protocols must also carefully manage the power consumption of the radio in order to ensure long battery lifetimes.

In this chapter, we provide an overview of protocol design for wireless sensor networks, focusing on how these designs differ from those in more conventional networking environments. We begin with link layer protocols (Section 5.2) including low-power MAC and link quality estimation. We move onto tree-based routing (Section 5.3) and efficient dissemination (Section 5.4), which are fundamental primitives for data collection and broadcast in WSNs. In Section 5.5 we describe reliable transport protocols, which differ in design from conventional approaches such as TCP. Cross-layer protocols that optimize for power consumption across the stack are described in Section 5.7. The emergence of IPv6 for low-power wireless networks (6loWPAN) is discussed in Section 5.8. In Section 5.6, we describe auxiliary protocols for time synchronization and localization, two essential services in sensor networks. Finally, in Section 5.9, we present our thoughts on how sensor network protocols might influence the design of the future Internet.

5.2 Link Layer Protocols

Link layer protocols provide media access, and packet transmission and reception service to the upper layer protocols. In sensor neworks, link layers also provide information about the nodes and links in the neighborhood. Although

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

high channel utilization is important, energy efficiency has been the focus of link layer research in wireless sensor networks.

Idle listening is the largest avoidable energy expenditure attributed to wireless communication in sensor networks. Idle listening is a phenomenon in which a node expends energy keeping its radio on even while no packet transmission or reception takes place. On radio chips such as the IEEE 802.15.4/ZigBee-compatible CC2420 radio, the radio consumes as much power in receive mode as it does while transmitting a packet. Fortunately, this cost is avoidable if we can duty-cycle the radio. However, duty cycling makes the media access problem harder: When a transmitter is ready to transmit a packet, the radio on the receiver might be off. Additional mechanisms are necessary to ensure both the transmitter and the intended receiver have their radios on at the time of packet transmission.

Two main radio duty-cycling approaches have been proposed in low-power MAC design for sensor networks. Coordinated or synchronous duty-cycling orchestrates radio duty cycles across the network in a predetermined schedule. This requires that nodes synchronize their schedules with each other, and may involve the use of a centralized scheduler, such as the base station. Uncoordinated or asynchronous approaches have no explicit coordination across the nodes. Some link layer designs combine these two approaches, but these are not typically used in practice. Uncoordinated schemes are far more common than coordinated schemes because they allow nodes to transmit packets at any time without requiring the overhead and complexity of explicit synchronization. We describe B-MAC and X-MAC, two examples of such protocols below.

5.2.1 B-MAC

B-MAC (Polastre et al. 2004) is a canonical example of a low-power MAC protocol for sensor networks. For coordination between multiple transmitters, B-MAC uses clear channel assessment and back-offs, as described further here. To enable low-power operation, B-MAC uses a technique called *low-power listening* to duty-cycle the radio. B-MAC also supports link-layer acknowledgments.

B-MAC uses *clear channel assessment* to determine the presence or absence of valid packet transmission in the channel. Clear channel assessment involves nodes determining the noise floor and comparing the level of signal with the noise floor to determine if the channel is clear. The noise floor can be different across different nodes, based on the ambient channel conditions. The noise floor can also change over time. B-MAC performs periodic estimation of the noise floor by sampling the ambient noise level in the environment. This sampling is done during times that are unlikely to have packet transmissions – the short wait time between a packet and its acknowledgment, for example. The median of these samples is used as the noise floor estimate for a given time. These

estimates are also averaged over time using exponential averaging to yield a more stable estimate of the noise floor.

When B-MAC receives a request to transmit a packet, it must first determine if the channel is clear. Sampling the channel once and comparing against the noise floor results in a large number of false positives. The key insight used to increase robustness in B-MAC's clear channel assessment algorithm is that comparing the low outliers of multiple samples against the noise floor results in far greater accuracy than using threshold-based channel assessment. So, B-MAC samples the signal level in the channel a few times, computes the outlier, and compares against the noise floor. If the channel is clear, B-MAC transmits the packet. If the channel is not clear, the node backs off for a randomized interval of time. When the back-off timer expires, it performs clear channel assessment again. If acknowledgments are enabled, the transmitter waits for the acknowledgment before receiving or transmitting other packets.

B-MAC uses *low-power listening*, also called *preamble sampling*, to duty-cycle the radio. By default, the radio is left in a low-power sleep state, in which it is unable to receive incoming packets. Every *sleep interval*, each node turns on the radio and checks for an incoming packet from a transmitter. If a packet is detected, the node leaves its radio on to receive the packet. To avoid receiving a partial packet (e.g., if the receiver starts listening in the middle of a transmission), the transmitter must transmit a *preamble* before the actual packet, which is set to the length of the receiver's sleep interval. The long preamble ensures that a receiver will listen at least once during a preamble transmission.

In one of the evaluation experiments, the authors ran B-MAC on a testbed of 14 Mica2 sensor nodes. During the experiment, each node sent one data packet every three minutes to the base station using a collection routing protocol. The result showed that the nodes achieved a worst-case duty-cycle of 2.5 % and less than 1 s data delivery latency over six hops.

B-MAC allows applications to easily configure the MAC parameters, such as the sleep interval and whether link-layer acknowledgments are enabled. Allowing such interaction with the MAC enables applications to achieve desired duty-cycling and energy efficiency depending on the context and need of the application at a given time.

In low-power sensor networks, the sleep interval can be in the order of several hundred milliseconds. This requires transmitting an equally long preamble before each packet, which consumes a large amount of energy. In addition, long preambles increase packet transmission latency. They also cause nearby nodes that are not the intended recipient of a packet to wake up and attempt to decode the incoming packet, thereby wasting energy.

B-MAC makes the tradeoff of reducing receivers' idle listening power consumption by increasing the transmitters' power consumption when sending

packets. This scheme is appropriate for low-data-rate applications in which transmissions occur infrequently. Reducing the overhead of packet transmissions has been the subject of much subsequent work, including the X-MAC protocol described below.

5.2.2 X-MAC

X-MAC (Buettner et al. 2006) is another uncoordinated duty-cycling MAC for sensor networks. X-MAC uses a series of short preambles and embeds the destination of the packet in the preamble. When a node checks for channel activity and receives the preamble, it can quickly determine if it is the intended recipient of the packet. X-MAC introduces a short wait time between the preambles. During these wait times, the intended receiver of the packet can signal the transmitter (through an ACK transmission) that the receiver's radio is on and ready to receive the packet. This signaling allows the transmitter to truncate the series of preambles and transmit the body of the packet, as shown in Figure 5.2.

These mechanisms together improve the performance of unicast data transmissions over the low-power listening scheme used by B-MAC. In an indoor testbed of TelosB nodes, with a sleep interval of 200ms and with five contending packet transmitters, each node sending one packet every second, the

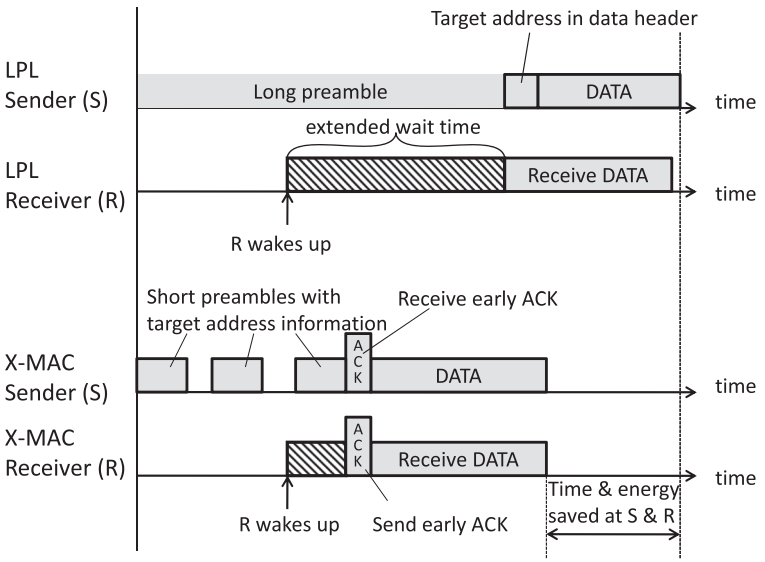


Figure 5.2. Comparison of transmitter-receiver coordination mechanism used in B-MAC (LPL) and X-MAC. Redrawn from Buettner et al. (2006).

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

authors report a duty-cycle of 20% compared to 65% with B-MAC's low-power listening.

5.2.3 Coordinated Duty Cycling

X-MAC still suffers from several limitations, such as that broadcast packet transmissions must still be transmitted for the full sleep interval. A great deal of ongoing research is investigating new protocols that attempt to reduce these overheads. Coordinated duty-cycling approaches can transmit broadcast packets more efficiently.

Coordinated duty-cycling MAC protocols explicitly compute the schedules for packet transmission and reception across the network. The nodes keep their radios on during these schedules and turn the radios off the rest of the time. Two types of schedule organization are common among the coordinated duty-cycling MAC protocols. MAC protocols such as S-MAC (Ye et al. 2002) and T-MAC (van Dam and Langendoen 2003) organize the schedules across the entire network (or a group of nodes) such that all the nodes (or a group of nodes) turn their radios on at the same time and start packet transmission and reception. The second approach organizes the schedules across the network such that the schedules reflect the routing topology or a specific data flow pattern. For example, the MAC protocol used by Dozer (Burri et al. 2007), described in Section 5.7, coordinates the radio schedules only between immediate neighbors, whereas DMAC (Lu et al. 2007) organizes the schedules along a routing path in a staggered pattern such that the packets can be forwarded along a path without interruption. Robust time synchronization and link dynamics, which often necessitate schedule revision at small timescale, are the key challenges to coordinated duty-cycling. Coordinated duty-cycling link layers seem less successful in practice than uncoordinated duty-cycling link layers.

5.2.4 Link Quality Estimation

Discovering nodes and links in the neighborhood provides valuable information to the routing and other protocols in sensor network. Most important function of neighborhood discovery is accurately and efficiently estimating the link qualities to the neighbors.

Most routing protocols in sensor networks use the ETX link metric (De Couto et al. 2003) to quantify the link reliability. ETX is a widely used link and path metric in wireless routing, and is defined as the *expected number of transmissions* for a packet to reach a destination node, assuming link-layer ARQ is in use. A perfect link has an ETX of 1; a larger ETX value implies a less reliable link.

In sensor network link quality estimation, the emphasis is on computing ETX of links efficiently, because each probe transmission costs valuable energy. Early

proposals for computing ETX of links use a periodic link quality probes and compute average these estimates over time (Woo et al. 2003). Just relying on link quality probes or beacons that are sent at a low rate makes the estimate less agile and hence inaccurate over the short term. However, the short-term link reliability is what determines the success or failure of a packet transmission at a given instant.

As an alternative, the so-called *four-bit link quality estimator* (Fonseca et al. 2007) estimates the quality of wireless links using information from the physical, link, and network layers in combination.

The physical layer can provide immediate information based on the radio's ability to decode an incoming radio packet. Many radios report the Received Signal Strength Indicator (RSSI) for incoming packets. The CC2420 radio used by many sensor nodes additionally reports a Link Quality Indicator (LQI) for each packet. This information is available on each packet reception and tends to have high variation: Each packet received can have different RSSI or LQI value. Although physical layer information cannot provide complete description of link quality, it can often be used as a fast estimate of link quality that can be improved using information from the link layer and link quality probes.

The link layer can provide information regarding success or failure of packet transmission on a link. Most sensor network link layer protocols support acknowledgments of unicast packet transmission. If a packet acknowledgment is received on a link, then we can take that as an indication of a good forward link (data transmission) and a good reverse link (acknowledgment transmission). Unlike computing ETX using separate probe transmissions, use of acknowledgments allows the estimator to compute and update the ETX at the time scale of one packet transmission.

The network layer can provide information regarding the importance of a link, such as a link being on a path to the root. Although physical and link layer information can be used to compose an accurate and agile estimator, the network layer has information regarding which links are on the path to the root or on better paths to the root. The link estimator can use this information to evict links, and sometimes even high-quality links, from the link estimation table to make room for links that are likely to be used by the routing protocol.

The four-bit estimator uses narrow and portable interfaces to access these sets of information from the physical, link, and network layers: Only four bits of information need to be exchanged between the link estimator and these layers (Figure 5.3). The four-bit estimator can work on any radio platform that provides the proposed interface. These interfaces are designed to be easily implementable across different platforms and make few assumptions about the specific radio technology or hardware. The four-bit estimator has been implemented on sensor node platforms that use CC1000, CC1100, CC2420, TDA5250, and RF230 transceivers.

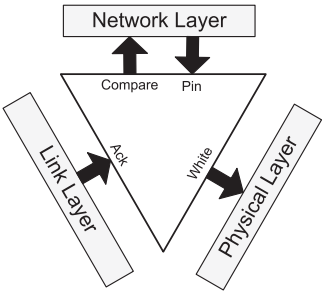


Figure 5.3. A link estimator, represented by the triangle in the center, uses four bits of information from the three layers. Outgoing arrows represent information the estimator requests on packets it receives. Incoming arrows represent information the layers actively provide.

In summary, the link layer is an active area of research in sensor networking. As energy efficiency becomes a focus on other mobile computing domains, some of the ideas from sensor networks link layer might inform designs in those spaces.

5.3 Tree-Based Routing

Collection trees are a core building block for sensor network applications and protocols. In their simplest use, collection trees provide an unreliable datagram routing layer that deployments use to gather data from the entire network to a small number of collection points, such as a single base station. Additionally, tree collection protocols provide the routing topology that underlies transport protocols such as RCRT and Flush, described in Section 5.5.

A collection protocol builds and maintains minimum-cost trees to nodes that advertise themselves as tree roots. Figure 5.4 shows an example of a routing tree formed in the network as the result of running a collection protocol. Collection is address-free: When there are multiple roots, collection sends the packets to root with the minimum cost without knowing its address.

The design goals for collection are:

Reliability: A collection protocol should deliver at least 90% of end-to-end packets when a route exists, even under challenging network conditions; 99.9% delivery should be achievable without end-to-end mechanisms.

Robustness: The protocol should be able to operate without tuning or configuration in a wide range of network conditions, topologies, workloads, and environments.

Efficiency: The protocol should achieve this reliability and robustness while incurring little overhead in terms of management traffic or framing overhead.

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

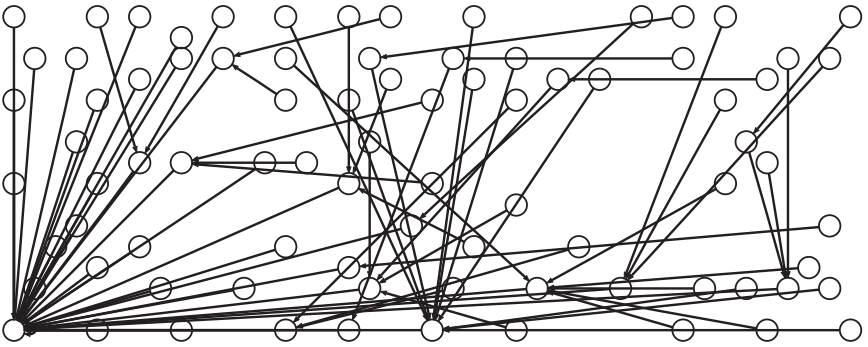


Figure 5.4. Result of running the Collection Tree Protocol on an indoor testbed of 85 nodes. All the nodes form a route to the collection point or root (node at the bottom left corner).

Many collection protocols have been proposed for sensor networks. Some collection protocols, such as MultihopLQI (MLQI 2009), are platform-dependent, whereas collection protocols such as MintRoute (Woo et al. 2003) and CTP (Gnawali et al. 2009) are platform-independent. We will use CTP as an example to study the design of a collection routing protocol.

5.3.1 CTP: Collection Tree Protocol

CTP (Gnawali et al. 2009) is a tree-routing protocol for sensor networks. Building on the functionality provided by the four-bit link estimator (Section 5.2) to accurately estimate link qualities, CTP incorporates two mechanisms to achieve high reliability, robustness, and efficiency. CTP uses an adaptive rate controller for routing beacons. As long as the routing gradient in the network is consistent, nodes reduce the frequency of control packets over time. When the network detects a loop or other inconsistency, nodes send control traffic more quickly to repair the topology. CTP also uses data traffic to actively probe the topology, detecting routing problems and repairing them as needed. This enables CTP to be highly agile and respond to broken links within a few packets.

All the nodes running CTP maintain an estimate of the cost (based on the ETX metric [De Couto et al. 2003] of its route to the root. A given node's cost is the cost of its next hop plus the cost of its link to the next hop: The cost of a route is the sum of the costs of its links. Roots advertise a cost of zero. Each data packet contains the transmitter's local cost estimate. When a node receives a packet to forward, it compares the transmitter's cost with its own. Since cost must always decrease, if a transmitter's advertised cost is not greater than the receiver's, then the transmitter's topology information is stale and there may be a routing loop. Using the data path to validate the topology in this way allows a protocol to detect possible loops on the first data packet after they occur.

Collection protocols typically broadcast control beacons at a fixed interval. This interval poses a basic tradeoff: A small interval reduces how stale information can get and how long a loop can persist, but uses more bandwidth and energy. A large interval uses less bandwidth and energy but can let topological problems persist for a long time. CTP's use of adaptive beaconing breaks this tradeoff, achieving both fast recovery and low cost. It does so by extending the Trickle algorithm (Levis et al. 2004) to maintain its routing topology.

The most relevant property of the Trickle algorithm is the adaptive timer interval. When the routing paths are reliable and stable, the timer interval is increased exponentially, thereby decreasing the rate at which beacons are transmitted. When events such as link dynamics or detected inconsistencies necessitate routing path repair, the Trickle timer is reset to a small interval, thereby increasing the rate at which the routing beacons are sent. More specifically, the routing layer resets the beacon interval to a small value on three events:

1. **It is asked to forward a data packet from a node whose ETX is not higher than its own.** The protocol interprets this as neighbors having a significantly out-of-date estimate and possibly a routing loop. It beacons to update its neighbors.
2. **Its routing cost decreases significantly.** The protocol advertises this event because it might provide lower-cost routes to nearby nodes. In this case, "significant" is an ETX of 1.5.
3. **It receives a packet with the pull bit set.** The "pull" bit advertises that a node wishes to hear beacons from its neighbors, for example, because it has just joined the network and needs to seed its routing table. The pull bit provides a mechanism for nodes to actively request topology information from neighbors.

In a network with stable links, both the first and second events are rare. As long as nodes do not set the pull bit, the beacon interval increases exponentially to one routing beacon every eight minutes. When the topology changes significantly, however, affected nodes reset their intervals to 128 ms and transmit to quickly reach consistency.

Gnawali et al. (2009) evaluated CTP on 12 different indoor testbeds with 20 to 310 nodes. They report achieving 90–100% data delivery reliability with duty-cycled and non-duty-cycled link layers.

5.3.2 In-network Aggregation

Apart from its use for path computation and as an underlying building block for higher-level protocols, tree-based routing is often used to guide the placement of

in-network aggregation mechanism in sensor networks. In-network aggregation enables nodes in the network to combine the data received from multiple nodes into lossy or lossless summaries (depending on the application requirement) and transmit these summaries to the destination, thereby reducing communication overhead. In-network aggregation can be successfully used in sensor network applications that report summaries of data to the user: The user perceives no difference in the reported data, even though the summaries are computed in the network to minimize communication overhead.

TinyDB (Madden et al. 2002), Directed Diffusion (Intanagonwiwat et al. 2000), and Synopsis Diffusion (Nath et al. 2004) are representative examples of systems that employ in-network aggregation to minimize communication overhead. These systems define the operation a node performs to transform data received from multiple nodes into a single data item. Thus, each node transmits one packet to the destination as opposed to forwarding all the packets it receives.

As an example, consider computing the minimum value of all sensor readings in a network. To compute this aggregate, each node receives data from its children in the routing tree, computes the minimum of all those data items and its own sensor reading, and transmits only the computed minimum value to its parent in the tree. When all the nodes in the network run this algorithm along the routing tree, the root receives the minimum value of the entire network. With in-network aggregation, each node in the network transmits only one packet to its parent on each round.

The collection of data readings has been the dominant traffic profile of sensor network applications. Although most applications report all the data to the root, there are promising in-network aggregation algorithms that compute accurate or approximate aggregates, thereby avoiding the communication overhead of transmitting all data to the root.

5.4 Dissemination

Most sensor networks require the ability to send configuration parameters or control messages to the entire network. Dissemination protocols are used to reliably broadcast information from one or a small set of nodes to the entire network in an energy- and bandwidth-efficient manner.

Efficient dissemination protocols are challenging to build. A simple flooding approach, in which each node rebroadcasts the received dissemination packet, can lead to congestion and cause substantial packet loss. Avoiding congestion and collisions to improve efficiency requires moderating the pace at which information is flooded to the network. Of course, such pacing results in higher dissemination latency.

The dissemination protocols in sensor networks have these design goals:

- **Reliability.** A disseminated packet must eventually be received by all nodes in the network.
- **Efficiency.** Dissemination must use as few packet transmissions as possible.
- **Low latency.** Data must be disseminated to the network as quickly as possible.

We describe Drip (Levis and Tolle 2008) and DIP (Lin and Levis 2008) as two case studies of dissemination protocols. Both of these protocols use the Trickle algorithm (Levis et al. 2004) as their underlying mechanism to time the transmission of dissemination packets, so we describe that first.

5.4.1 Trickle

Trickle (Levis et al. 2004) is an eventual consistency protocol that efficiently bring data stored by multiple nodes in the network to a consistent state. In Trickle, nodes store some local state (such as a data object or software binary) with an associated *version number* that is monotonically increasing. When the state is updated by any node, the version number is incremented. Trickle is used to maintain consistency in the version number across the network, whereas the Drip protocol, described later in the chapter, is used to transfer the actual state between nodes.

Trickle is a gossip-based protocol: Each node periodically broadcasts its known version number to its neighbors. If a node hears a later (that is, higher) version number than its own, it pulls the new state from the node and updates its version number accordingly.

To limit overhead, Trickle adapts the rate at which version information is broadcast by sensor nodes. Each node maintains a local timer that determines the rate at which the version information is broadcast. The timer rate is adjusted based on the agreement between nodes in the local radio neighborhood. If all nodes in a neighborhood have the same version, there is no need to rebroadcast frequently. The timer interval is increased exponentially following each rebroadcast by a node, until it reaches a maximum interval duration. If a node receives information with a higher version number, the timer is reset to a small interval, which allows new information to be rapidly disseminated (Figure 5.5). To avoid congestion and redundant transmissions, if a node detects that another node has transmitted the new information before its timer expires, it cancels the scheduled transmission and increases its timer interval.

5.4.2 Drip

Drip (Levis and Tolle 2008) is a data dissemination protocol built on top of the Trickle algorithm. Whereas Trickle allows a node to discover the existence of new versions of data, Drip adds the mechanism to transfer the data object

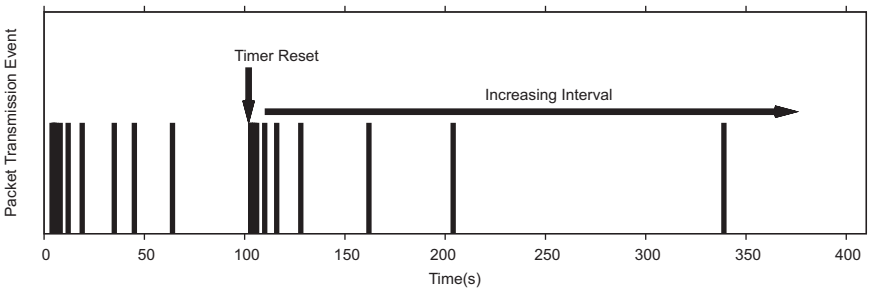


Figure 5.5. Trickle timer used to time the packet transmissions of a dissemination protocol. The Trickle timer starts with a small interval and doubles the interpacket interval after each transmission until the timer is reset.

itself. The canonical use case is to supply all nodes with a new set of configuration parameters. Trickle is used to inform nodes of the existence of the new parameters, whereas Drip ensures all nodes actually get a copy of them.

Once a node learns that its version is out of date, it broadcasts a request for the latest version. A node receiving this request replies by pushing the data object to the requesting node using a series of packet transfers with link-layer acknowledgment and retransmission. Because data transfers happen over a single radio hop, they can be done efficiently without the need for explicit packet routing; using Drip, the entire network can eventually receive the newest version of a data object injected by one node. Figure 5.6 shows the evolution of the communication overhead of Drip; the number of packet transmissions settles at the rate corresponding to the maximum Trickle interval despite the high overhead in the beginning.

5.4.3 DIP

Trickle and Drip are designed to support a single data object and associated version information. DIP (Lin and Levis 2008) is a protocol that can efficiently

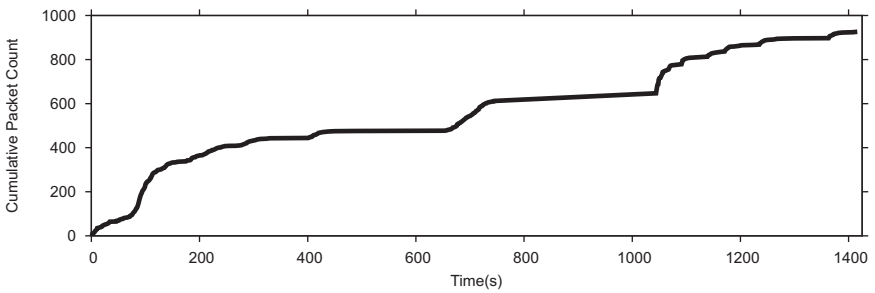


Figure 5.6. Drip's communication overhead grows slower over time, eventually settling at the rate corresponding to the maximum Trickle interval when there are no dissemination requests.

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

disseminate a large number of data objects. Although Trickle can be used for this purpose, the overhead of maintaining a separate timer and performing individual broadcasts for each data object becomes prohibitive as the number of data objects increases.

A simple approach is to transmit the complete set of known version numbers each round, using the Trickle algorithm. However, given the limited size of radio packets on typical sensor node platforms, it is typically not feasible to do this when the number of data objects is large. DIP uses an alternative approach, called the *scan* algorithm, that maintains a single timer and advertises a subset of the version numbers known by the node each time the timer fires. The challenge is deciding which subset of version numbers to advertise. DIP's scan algorithm collects estimates of data item version numbers based on neighbor advertisements. DIP then prioritizes the broadcast of data item versions that are likely to be newer than the ones in the neighborhood over rest of the data items. Although this approach has low communication overhead, it increases the latency for detecting the presence of new version numbers in the network.

DIP uses the *search* algorithm to locate the updated values using a hash tree. DIP advertises a summary of all the versions of data objects stored by a node, which is computed as a hash of the version numbers. When a node receives a hash summary different than its own, it knows that at least one item on the transmitter node is of a different version than its own. However, it cannot tell which item is different. The receiver then advertises its own summary: Instead of a single hash covering all its data version numbers, it transmits a set of summary hashes over smaller ranges, but covering the same range of data where the difference occurred. This allows the original transmitter to narrow down the changed data version to the size of the range of the hash summary in the packet. This iterative exchange of messages with smaller ranges results in the nodes traversing the hash tree to find the data item that has changed. When the nodes determine the exact data item that is different, the nodes exchange the data value corresponding to the new version number.

When a large number of new data items are introduced for dissemination, the search algorithm can require a large number of message exchanges to accurately identify the updated data item. In this scenario, it is more efficient to advertise each data item because new versions can be identified using just a single message. Also, switching to scanning after the search has narrowed the update to a small set of items is more efficient than continuing the message exchange between the nodes to search down the hash tree because lost packets will cause the search to go back up the hash tree and the constant factors of searching are higher than scanning with small ranges. The key insight behind DIP's efficiency is its switching between the search and scan algorithm depending on the network conditions and the number of updates.

DIP's communication overhead and new version detection latency stays asymptotically constant with the order of the number of data objects, but incurs logarithmic communication overhead in identification of the data item that has changed. The authors evaluated DIP on a testbed of 80 MicaZ motes with sixty-four data objects. In one experiment, the authors updated eight of the data items. DIP disseminated these updates to the entire network with 50% and 15% lower overhead and 80% and 60% less time than the search and scan protocols, respectively.

5.4.4 Other Dissemination Protocols

Dissemination protocols are used not only to disseminate commands and configuration parameters, but also to build higher-layer application-specific dissemination services. One interesting example is disseminating executable programs to the sensor networks. Deluge and Maté are good examples of such code dissemination protocols.

Deluge (Hui and Culler 2004) allows new executable binaries to be disseminated to the network, which might be on the size of tens of kilobytes, which is fairly large considering the limited radio bandwidth and high energy cost for packet transmission. The executable program is divided into pages and injected into a source node that acts as the seed for the dissemination protocol. When a new page is received by a node, it advertises the page number. As an optimization, a node advertises page i only if all previous pages $[0, i)$ are also available at the node. As in Drip, a node that learns of the availability of an executable page it does not yet have will transmit a request for the page to be transferred to it. Once the transfer of all the executable pages is complete, Deluge reprograms the node with the assembled program image and reboots the node.

Instead of transferring entire executable images, it is also possible to make use of a simple virtual machine on each sensor node, which can execute compact bytecode programs. Maté (Levis and Culler 2002) is an example of such a system. Maté programs are organized as a collection of capsules, each of which can fit within a single radio packet. When a node receives a capsule with a new version number, the node installs and runs the code in the capsule and forwards the capsule to its neighbors. This process continues until the program is disseminated and installed across the network. This greatly reduces the overhead for reprogramming the network, but restricts programs to using the limited instruction set defined by the Maté virtual machine. However, the Maté VM can be customized with application-specific bytecodes implementing native functions.

5.5 Reliable Transport

Many applications for WSNs require reliable transfer of data from sensor nodes to the base station. This is especially true in domains involving high-resolution

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

signal collection, such as structural (Chintalapudi et al. 2006; Kim et al. 2007b), acoustic (Allen et al. 2008), or seismic (Werner-Allen et al. 2006) monitoring. Transferring a large volume of data from a sensor node over a multihop path to a base station is challenged by packet loss, radio channel contention, and lack of buffer space on nodes acting as routers.

Although TCP is widely used on the Internet for reliable stream-based transport, this approach is not suitable for lossy multihop wireless networks. This is primarily because TCP interprets packet loss as being due to buffer overflow in routers, and tunes the congestion window size accordingly. In WSNs, however, the *rate* at which a source node injects packets into the network is the critical parameter, since contention for the radio channel along a multihop path limits the effective rate at which packets can be relayed. Unlike wired networks, in wireless networks, packet loss is caused by transients in link conditions, collisions, and routing path churn, rather than by persistent congestion. In this section, we describe two approaches to reliable transport in sensor networks, Flush and RCRT.

5.5.1 Flush

Flush (Kim et al. 2007a) is a reliable bulk transfer protocol for multihop WSNs. The protocol operates over a routing tree such as MintRoute or CTP (Section 5.3) rooted at a base station. The base station transmits a request for data object stored on a given node using a broadcast flood (Section 5.4). The node hosting the data object breaks it into multiple packets, each with a corresponding sequence number, and streams those packets to the sink over the multihop path.

Flush relies on several techniques to ensure reliability and high throughput. First, both link-layer ACKs and end-to-end selective NACKs are used to recover from lost packets. Second, data is streamed along the routing path to the sink at a rate that is chosen carefully to avoid intrapath contention. Finally, Flush only supports a single bulk transfer in the network at a time, to avoid interflow interference.

Flush relies both on link-layer ARQ and end-to-end NACKs for reliability. At the link layer, a packet will be retransmitted up to four times before being dropped by the sender. Limiting the number of link-layer retransmissions is necessary to account for changes in the underlying routing topology or node failures. Link-layer retransmission paves over intermittent losses due to radio channel noise and significantly reduces the number of expensive end-to-end retransmissions.

The sink will send a selective NACK containing the sequence numbers of missing packets in the flow after it believes the last source packet has been transmitted, or after a timeout based on an estimate of the end-to-end RTT. Upon reception of the NACK, the source retransmits those packets up the routing tree

in the manner described earlier. This process repeats until the sink has received the entire object.

The key contribution of Flush is its *rate control* algorithm that determines the peak rate at which the source node can inject new packets into the network without inducing loss due to intrapath collisions. In a multihop path, as nodes forward data to the root, those transmissions potentially collide with other transmissions both upstream and downstream along the path. A simple solution would be to avoid any pipelining, but this would eliminate spatial reuse of the radio channel, thereby reducing throughput.

Flush dynamically estimates the maximum sustainable sending rate to maximize throughput and avoid contention along the routing path. Nodes measure their own packet transmission delay and receive feedback from upstream nodes on their delays, and those of other nodes that might interfere with their transmissions. The set of potential interfering nodes is determined based on snooping the radio channel; Flush assumes that a node can overhear packet transmissions from the interfering nodes. Flush relies on the underlying MAC protocol to schedule individual packet transmissions, so this rate control is performed at the transport layer.

Combining these techniques, Flush achieves reliable transfer throughput that closely matches the best possible performance measured using a fixed transmission rate. The key is that Flush *automatically* determines the optimal rate, which depends on the node's depth in the routing tree and the overall network topology. Flush scales well with long routing paths: The protocol has been evaluated using a 48-hop linear chain of nodes deployed outdoors.

5.5.2 RCRT

RCRT (Paek and Govindan 2007) is another reliable transport protocol. Unlike Flush, RCRT handles multiple concurrent reliable flows. This is necessary in cases where all nodes are generating data simultaneously, requiring real-time streaming of the data back to the sink, owing to the limited buffer capacity. In RCRT, the base station performs centralized congestion control for all source nodes, using global knowledge of the performance of each flow. This approach permits global application of policies to drive allocation of network capacity.

Like Flush, RCRT uses end-to-end NACKs for repairing lost packets. The essential congestion control mechanism is based on measuring the *time to repair a loss*. If losses are repaired quickly enough, there is no need for rate adjustment. However, if the repair time exceeds the expected packet round-trip time, congestion is present in the network, and transmission rates are adjusted. RCRT adapts the *aggregate rate of all flows* using an AIMD adaptation scheme. Once the new aggregate rate is determined, individual flow rates are calculated based on the current policy in use. Example policies include a demand-proportional

scheme, in which rates are allocated in proportion to each flow's desired rate, and a fair policy in which all flows receive an equal rate.

The authors evaluated RCRT on an indoor testbed of forty nodes, using tree routing with paths up to eight hops in length. They show that RCRT is able to sustain a per-node traffic demand of 0.8 packets/s, which is 88% of the optimal sustainable rate for their testbed. Further, RCRT does not suffer congestion collapse when load is increased. RCRT's goodput is more than double that of IFRC (Rangwala et al. 2006), another protocol designed for network-wide rate control.

Comparing Flush and RCRT, the main difference is that Flush supports a single reliable flow whereas RCRT supports multiple simultaneous flows and supports policies to balance the amount of bandwidth allocated to each. Flush computes the transmission rate at the sink using feedback from downstream nodes, whereas RCRT performs a centralized search for stable transmission rates at the base station. Unlike Flush, RCRT does not assume that nodes can overhear packet transmissions from other nodes. Both protocols rely on end-to-end selective NACKs for reliability, and both can work with any routing protocol that provides bidirectional paths to the sink.

5.6 Support Protocols

Apart from protocols for data communication, wireless sensor networks often employ a range of *support* protocols that provide services such as time synchronization and node localization. In this section, we will briefly discuss three representative protocols in this class: the Flooding Time Synchronization Protocol (FTSP) (Maroti et al. 2004), localization using acoustic beacons (Simon et al. 2004), and radio interferometric localization (Maróti et al. 2005).

5.6.1 The Flooding Time Synchronization Protocol

Time synchronization is an essential service for sensor networks in which the data acquired by nodes must be accurately timestamped against a global clock. Individual sensor nodes use oscillator crystals with a tolerance of around 40 ppm, causing the local clocks of each nodes to drift substantially. Moreover, this drift varies over time owing to fluctuations in temperature and voltage. Applications such as acoustic or seismic monitoring (Simon et al. 2004; Chintalapudi et al. 2006; Werner-Allen et al. 2006) require time accuracies in the millisecond or microsecond range, so it is inadequate to perform a one-time translation of each node's local clock to a global timebase. Time synchronization must be run periodically.

A simple approach would have a central node (such as the base station) advertise a global time to all nodes in the network, which would set their local

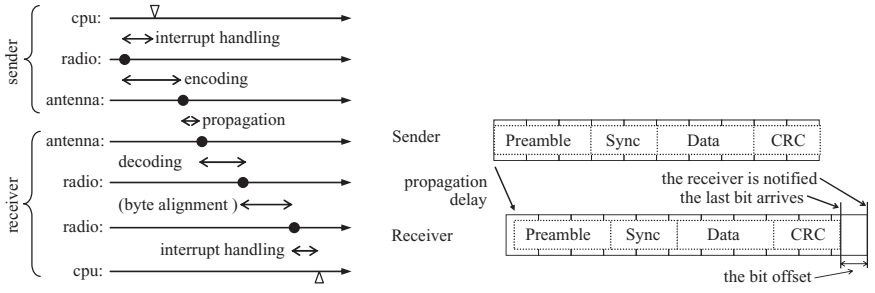
clocks accordingly. However, propagating timebase information throughout a large network is challenging owing to the use of multihop paths and the timing uncertainty of radio communication. Transmitting and receiving radio messages incurs nondeterministic delays due to the MAC protocol, transmission and reception overheads, and interrupt processing delays. These delays must be accounted for when relaying timebase information through the network.

The Flooding Time Synchronization Protocol (FTSP) (Maroti et al. 2004) operates as follows. A single node acts as the root of a synchronization tree; the local clock of the root node is used as the global timebase. Failure of the root node leads to election of a new root. The root periodically beacons the global time value. Nodes within one hop of the root receive these beacons and compute a mapping from their local clock to the global timebase, as described later in this chapter. Each node rebroadcasts the beacon, allowing the global time to propagate throughout the network.

FTSP carefully accounts for the uncertain communication delays by timestamping outgoing beacon packets *after* the MAC delay, just prior to the actual transmission of the first byte of the packet. Likewise, a receiver timestamps the received packet on the arrival of the first reception interrupt. On the Chipcon CC1000 radio used in the study (Maroti et al. 2004), an interrupt is generated for each received byte of the packet. Receivers can determine the interrupt processing jitter by measuring the time between each successive interrupt (Figure 5.7).

To compensate for clock drift, each node measures the offset between the sender's timestamp and its local clock. Given that clock drift is expected to be linear over short time intervals, nodes perform a linear regression on these offsets to correct their local clocks for drift.

FTSP has been evaluated extensively on a testbed of 60 Mica2 nodes. All nodes were placed within radio range of each other, but a 6-hop multihop



(a) (b)

Figure 5.7. To minimize the impact of nondeterministic delays in packet timestamping on time synchronization accuracy, FTSP implementation on the CC1000 radio records timing for each byte-boundary in a packet and computes an aggregate over these times as the packet timestamp. *Source:* Maroti et al. (2004).

https://doi.org/10.1017/CBO9780511921117.006 Published online by Cambridge University Press

topology was induced in software. To measure time synchronization accuracy, a *reference broadcaster* node transmits periodic messages that all nodes in the testbed receive and timestamp using FTSP. Since it is assumed that all nodes should receive the message at the same time (module differences in RF propagation delay, which are negligible in the small testbed), comparing the timestamps of this global event across nodes allows one to assess timing accuracy. The authors show that FTSP exhibits an average pairwise error of 3 μs and an overall maximum error of less than 14 μs.

5.6.2 Localization Using Acoustic Ranging

Accurately determining the position of sensor nodes is another essential service in many deployments. Although many sensor networks involve static nodes that may be placed in the field by hand, it is inconvenient and error-prone to rely on manual surveying or GPS for determining a node's position. GPS only works outdoors and only provides an accuracy of a few meters. For sensor networks deployed in an ad hoc fashion (e.g., dropped from an airplane) or involving mobile nodes, it is critical that the network be able to self-localize.

Localization is a heavily studied topic, and a wide range of techniques have been proposed. One of the most common approaches involves ranging using acoustic time-of-flight (Simon et al. 2004). Sensor nodes are equipped with a sounder and a microphone. A source node broadcasts a radio message immediately followed by an acoustic chirp that can be detected by nearby nodes using their microphones. Receivers timestamp the arrival of the RF message and the chirp. Since it is assumed that the RF propagation delay is negligible, the time-of-flight of the chirp can be readily computed as the time between the reception of RF message and the acoustic chirp. Assuming the speed of sound in the environment is known (which can vary based on temperature and humidity), the approximate range to the source node can be computed.

This process is made more challenging by the limited signal-processing capability of motes, missed or incorrect chirp detections, reflections, and limited acoustic sensing range. To address these issues, the system transmits multiple chirps that are combined in postprocessing to enhance SNR. A digital bandpass filter is also used to suppress noise. In Simon et al. (2004), this ranging technique is shown to be accurate to within 10 cm over a range of up to 9 m.

Once a set of pairwise range estimates are known, they can be combined to determine the relative location of nodes in the network. One approach is to collect range estimates at a central node (e.g., the base station) and perform iterative optimization until a stable configuration is determined. Landmark nodes with known locations are used to anchor the network's orientation to geographic coordinates.

5.6.3 Radio Interferometric Localization

As an alternative to acoustic ranging, it is possible to use RF signals alone to obtain extremely accurate range estimates between nodes. The radio interferometric technique described in Maróti et al. (2005) involves two nodes transmitting sinusoid RF patterns at known frequency offsets. A pair of receiver nodes can determine the beat pattern induced by the interfering signals. This results in a series of equations that relates the phase offset of the beat patterns to the nodes' relative locations. Eight nodes are able to localize themselves in three dimensions. Using a network of 16 nodes, the authors demonstrate the average positional error to be 3 cm with a maximum of 6 cm. However, this technique is not yet appropriate for mobile networks because it involves extensive measurements and calibration. It also requires the use of radios that can be configured to transmit an unmodulated sine wave. Although more recent 802.15.4 radios provide a test mode with this capability, it is not possible to tune the frequency of the sine wave at fine enough granularity.

5.7 Cross-Layer Concerns

Sensor network protocol designs face a tension between the desire to exploit layering and the need for cross-layer optimizations to get the best efficiency and performance. Although this problem is evident in conventional networks as well, in sensor networks it is particularly pronounced due to the extreme resource limitations of sensor nodes.

Sensor network protocols have typically followed a layering principle that separates the physical, link, routing, and application layers, as we have presented in this chapter. However, many designs perforate the layer interface by providing control knobs and feedback to higher levels of the stack. B-MAC (Polastre et al. 2004) exposes a range of control parameters such as the listen interval and preamble length. SP (Polastre et al. 2005b) provides a neighbor table with link state and congestion information. The FTSP time synchronization protocol (Maroti et al. 2004) relies on link-layer packet timestamping well below the FTSP protocol layer itself. Each of these examples illustrates the need for cross-layer information in designing sensor network protocols. Given that sensor networks are not constrained by legacy application software and standards compliance, the community has had the opportunity to explore the protocol design space more broadly and experiment with vastly alternative designs.

Taking cross-layer design to the extreme, Dozer (Burri et al. 2007) is a system for low-power data collection for environmental monitoring. Unlike previous designs that largely separated the MAC, routing, and application layers, Dozer fully integrates these functions to achieve extremely efficient operation. The goal is to permit nodes to spend the maximum amount of time in an

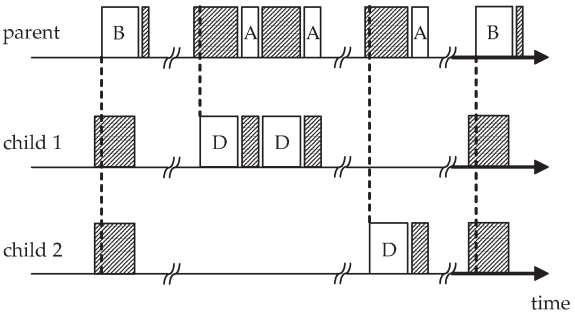


Figure 5.8. Packet transmission and reception time synchronization between a parent and its two children. The parent sends a beacon (B) to determine the transmission slots for the children. The children transmit the data messages (D), which are explicitly acknowledged (A). *Source:* Burri et al. (2007).

energy-efficient sleep mode. In Dozer, nodes make use of tree routing. TDMA is used for scheduling communication; each parent node in the tree defines the TDMA schedule for its children (Figure 5.8). This approach allows the parent to avoid idle listening by waking up only when a child is about to transmit. There is no explicit collision-avoidance mechanism apart from randomization of TDMA schedules across nearby nodes.

Dozer's tree formation protocol is based on periodic beacons that each node transmits to maintain the TDMA schedule for its children. A node wishing to join the tree first listens for beacons and selects a parent based on the potential parent's depth in the tree and the number of children it has (which can be determined from the beacon message). The node joins by sending a *connection request* to the chosen parent, which adds it to the TDMA schedule. Nodes periodically listen for beacons from other potential parents and cache this information so that a new parent can be selected quickly in the event of a parent failure. Nodes that are unable to join the network drop into a *suspend mode* to avoid polling the radio channel continuously.

Dozer incorporates a number of techniques to further improve efficiency. Each node pads its TDMA schedule by a random amount on each round to avoid schedule collisions between sibling nodes in the tree. This avoids the need for explicit coordination of TDMA schedules across nodes. Child nodes predict the length of each round's random padding using a pseudorandom number generator in which the seed is shared with the parent. Although there is no guarantee that collisions will not happen, the randomized schedule makes multiple collisions in successive rounds far less likely.

Dozer has been evaluated in an indoor network of forty nodes over the span of several weeks. The sensor sampling interval was set to be 120 s and the beacon interval was set to 30 s. The network obtained very low message losses

(an average of 1.29%) with an overall average radio duty cycle of just 0.1679%. This yields a mean energy consumption of just 0.082 mW.

5.8 The Emergence of IP

In recent years, there has been increased interest in linking wireless sensor networks with conventional IP-based networks at the IP protocol layer. Initially, it was thought that the IP family of protocols would be too heavyweight to run on resource-limited sensor nodes, or that such protocols would be inappropriate for the new demands of sensor network applications. However, a substantial engineering effort by two IETF working groups – IPv6 over Low power WPAN (6LoWPAN) and Routing Over Low-Power and Lossy Networks (ROLL) – has resulted in efficient IP protocol implementations for sensor networks.

5.8.1 IP Packet Frame

The IETF 6LoWPAN working group has standardized the encoding of IPv6 packets on 802.15.4 networks with its publication of RFC 4944 (Montenegro et al. 2007). The IEEE 802.15.4 standard is likely to be the most pervasive link layer used in low-power wireless networks, such as sensor networks. Transmission of IPv6 packets over these links is challenging due to the limited packet size supported by 802.15.4. Whereas IPv6 packets can be as large as 1,280 bytes, 802.15.4 only supports a maximum packet size of 102 bytes, assuming maximum frame header overhead. By default, the IPv6 header occupies 40 bytes, leaving little room for application payload.

To reduce this overhead, the 6LoWPAN working group has defined a format based on header compression. In this scheme, a transmitter substantially compresses the header of outgoing packets by substituting or eliding header fields that have common values or that can be inferred from other header fields. The 40-byte IPv6 header can be compressed to as little as two bytes.

5.8.2 IP-Based Routing Protocols

Although no IP-based routing protocol for sensor networks has been standardized, there are several IP-based protocols in use in sensor networks both in commercial products and research.

An IP-based routing protocol must support unicast routing between arbitrary nodes in the network. In sensor networks, a combination of tree-based routing and source routing is typically used to enable unicast routing. In Blip, an IPv6 protocol stack implemented in TinyOS, all the nodes in the network first form a tree-routing topology. Then the nodes periodically send their topological information, such as their parent in the routing tree, to the root of the network. When

a node needs to send a packet to an arbitrary node in the network, the packet is first forwarded to the root. The root, which has topological information for all the nodes in the network, then source-routes the packet to the destination.

The recently formed IETF ROLL working group is tasked with standardizing the routing protocols for the low-power networks such as wireless sensor networks. The working group, after a survey of wireless network protocols, concluded that no existing IETF protocol meets all of the requirements of these low power networks. At the time of this writing (October 2010), the ROLL working group has produced a proposed standard for routing protocols to be used in these lossy and low-power networks.

5.9 Sensor Networks and the Future Internet

Wireless sensor networks have broad implications for the future design of the Internet. The sheer number of sensor network nodes that may be deployed in the future raises significant challenges in terms of naming and addressing, communication protocols, resource management, and reliability. Of course, sensor networks differ substantially from conventional Internet hosts. They are extremely resource-limited; connected via wireless mesh networks; and often operate at low duty cycles. Further, the traffic produced by sensor networks is not typically dominated by unicast end-to-end flows; much traffic is multipoint-to-point (e.g., data aggregation up a spanning tree) or point-to-multipoint (dissemination to all nodes in the WSN).

Two opposing views have emerged with respect to the relationship between sensor networks and the Internet at large. At one extreme, sensor networks are treated as special-purpose appliances that would be connected to the Internet via a gateway. The gateway would translate between TCP/IP (and higher-layer protocols, such as HTTP or Web Services calls) and a low-level, possibly proprietary, protocol used within the WSN itself. ZigBee (The ZigBee Alliance 2009) is emerging as one contender for the back-end sensor network protocol and defines both routing and device profiles for a range of applications. The "smart gateway" approach presumes that sensor networks will evolve independently of the rest of the Internet, using specialized protocol implementations that are tailored for the resource-constrained, low-duty-cycle nature of sensor networks. The gateway can provide additional services such as storage and caching.

At the other extreme, sensor networks would be treated as first-class citizens on the Internet, communicating directly via TCP/IP to other Internet-based clients and applications. The ROLL working group of the IETF is developing routing solutions for sensor networks based on an end-to-end IP solution. In Section 5.8, we summarize the various technical directions being explored within this space. The upshot is that it is now possible to communicate with WSN nodes using a variant of IPv6 with special support for the limited memory, bandwidth,

and energy capacity of sensor nodes. This approach significantly narrows the gap between WSNs and standard Internet applications.

The tension between these two competing approaches arises because sensor networks challenge the "end-to-end principle" (Saltzer et al. 1984) that has predominated the Internet architecture for decades. A smart gateway pushes complexity into the network (rather than the edges) to support the limited capabilities of a sensor network, whereas an IPv6-based solution upholds the end-to-end principle.

Regardless of the protocol stack used, bridging between WSNs and the Internet raises a number of special considerations that have yet to be fully resolved. The first is the nature of communication between user applications and sensor networks. End-hosts communicating with a sensor network must recognize that these devices cannot be treated like conventional Internet hosts in terms of the data rate, latency, and nature of traffic that WSNs can support. Although it is technically possible to run a lightweight Web server on a WSN node running an IPv6 stack, this may not be the best way for users to interface to the network. More likely, a programmatic interface (e.g., via RPC) will be required, as well as a portal to access a sensor network providing access to aggregate and historical data. Moreover, making WSNs directly accessible via the Internet will no doubt raise concerns over the impact of malware or buggy protocol implementations. Deployed WSNs may need to be "protected" from DoS attacks, port scanners, and viruses that would potentially disrupt their operation.

Moreover, transport and routing protocols present challenges when running across the boundary between the Internet and a resource-constrained sensor network. As described earlier, sensor network routing protocols are significantly different than their Internet counterparts. As a result, routing paths spanning the Internet and sensor networks will traverse segments with very different characteristics. From the transport protocol perspective, packet loss may occur for very different reasons on different segments of the path. As an example, TCP interprets packet loss as the result of congestion, which is often true in wired networks but less prevalent within a multihop wireless network. As a result, TCP buffers, windows, and timers will either stretch the limited sensor node resources or underutilize Internet end-host resources. Thus, transport protocols must evolve to work well under these asymmetric conditions.

Of course, link and routing protocols only provide the lower layers of a protocol stack. It is still necessary to provide protocol layers for accessing sensor data, tasking sensors, and administrative control. These protocols have yet to be defined, though it may be possible to leverage existing standards, such as IEEE 1451, which provides mechanisms for communicating, with a wide range of sensors and actuators.

Once sensor networks have been deployed in more widespread settings, discovery protocols will be a major concern. Applications must be able to query

the operational characteristics of the sensor network, such as what types of sensors are installed; the locations of those sensors; and whether sensors have been recently calibrated or serviced. Likewise, the network should export metadata to report whether sensor nodes have failed. This is critical when requesting aggregate data from a large network, since the number and placement of failed nodes can have a substantial impact on the results that are returned.

Finally, conventional approaches to naming and addressing – such as DNS – seem to be ill-suited to sensor networks, which may consist of a large number of nodes whose population and capabilities change over time. Rather than addressing individual sensor nodes, a more appropriate paradigm may be to name the *data* using a semantic addressing scheme. A declarative query interface such as TinyDB (Madden et al. 2002) allows a user to request data with given filtering, aggregation, and periodicity parameters.

Although this chapter focuses on static sensor networks, a new class of networks involving mobile nodes is emerging. Examples include wildlife tracking using GPS collars (Zhang et al. 2004) and opportunistic collection of sensor data using cell phones carried by individuals (Mun et al. 2009). Hybrid networks, combining both static and mobile sensors, are another important future direction. Mobility makes it difficult to name nodes and requires a different approach to routing given that nodes may be disconnected from the network for significant periods of time. Furthermore, using sensors that are not necessarily "owned by" the sensor network in which they participate (such as cell phones) raises a number of issues in terms of accountability, security, and privacy.

If history is any guide, the increasing diversity of devices and applications connected to the Internet will soon encompass sensor networks as well. However, the traffic characteristics produced by sensor networks will be substantially different than conventional uses of the Internet, even with the increasing prevalence of other embedded and mobile devices, such as smartphones. Sensor networks will mostly be generators of traffic rather than sinks, so the focus will be on optimizing the outflow of data. Likewise, sensor networks may not be limited by the demand for human-tolerable access latencies, opening up the design space even further.

5.10 Conclusions

Wireless sensor networks are a fundamentally new kind of distributed computing system that present new opportunities and challenges for network protocol design. Their extreme constraints on energy, memory, bandwidth, and computational resources has led to new protocol designs at every layer of the stack, including link, routing, reliable transport, and application interfaces. Overall, sensor network protocol designs strive for low-power operation in the face of variable link conditions, node failures, and changing application requirements.

As a result, much of the research to date has focused on cross-layer approaches that highly specialize the protocol stack for a given application. The recent emergence of lightweight IPv6 implementations for sensor nodes has led to new questions about the role that sensor networks should play in the evolving Internet architecture. Although many open questions remain, we expect that further experience with this technology in both academic and commercial settings will lead to increased convergence with the Internet as a whole.

References

Allen, Michael, Girod, Lewis, Newton, Ryan, Madden, Samuel, Blumstein, Daniel T., and Estrin, Deborah. 2008. VoxNet: An Interactive, Rapid-Deployable Acoustic Monitoring Platform. *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN '08)*.

Buettner, Michael, Yee, Gary V., Anderson, Eric, and Han, Richard. 2006. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pages 307–320.

Burri, Nicolas, von Rickenbach, Pascal, and Wattenhofer, Roger. 2007. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, pages 450–459.

Chintalapudi, Krishna, Paek, Jeongyeup, Kothari, Nupur, Rangwala, Sumit, Caffrey, John, Govindan, Ramesh, Johnson, Erik, and Masri, Sami. 2006. Monitoring Civil Structures with a Wireless Sensor Network. *IEEE Internet Computing*.

De Couto, Douglas S. J., Aguayo, Daniel, Bicket, John, and Morris, Robert. 2003. A High-Throughput Path Metric for Multi-Hop Wireless Routing. *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*.

Fonseca, Rodrigo, Gnawali, Omprakash, Jamieson, Kyle, and Levis, Philip. 2007. Four Bit Wireless Link Estimation. *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*.

Gnawali, Omprakash, Fonseca, Rodrigo, Jamieson, Kyle, Moss, David, and Levis, Philip. 2009. Collection Tree Protocol. *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*.

Hui, Jonathan W., and Culler, David. 2004. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*. ACM Press, pages 81–94.

Intanagonwiwat, Chalermek, Govindan, Ramesh, and Estrin, Deborah. 2000. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom '00)*.

Kim, Sukun, Fonseca, Rodrigo, Dutta, Prabal, Tavakoli, Arsalan, Culler, David, Levis, Philip, Shenker, Scott, and Stoica, Ion. 2007a. Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys '07)*.

Kim, Sukun, Pakzad, Shamim, Culler, David, Demmel, James, Fenves, Gregory, Glaser, Steve, and Turon, Martin. 2007b. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '07)*.

Levis, Philip, and Culler, David. 2002. Maté: A Tiny Virtual Machine for Sensor Networks. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*.

Levis, Philip, and Tolle, Gilman. 2008. *Dissemination of Small Values*. TinyOS Extension Proposal TEP-118.

Levis, Philip, Patel, Neil, Shenker, Scott, and Culler, David. 2004. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*.

Lin, Kaisen, and Levis, Philip. 2008. Data Discovery and Dissemination with DIP. *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 433–444.

Lu, Gang, Krishnamachari, Bhaskar, and Raghavendra, Cauligi S. 2007. An Adaptive Energy-Efficient and Low-Latency MAC for Tree-Based Data Gathering in Sensor Networks. *Wirel. Commun. Mob. Comput.*, **7**(7), 863–875.

Madden, Samuel, Franklin, Michael J., Hellerstein, Joseph M., and Hong, Wei. 2002. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI '02)*.

Maroti, M., Kusy, B., Simon, G., and Ledeczi, A. 2004. The Flooding Time Synchronization Protocol. *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*.

Maróti, Miklós, Völgyesi, Péter, Dóra, Sebestyén, Kusý, Branislav, Nádas, András, Lédeczi, Ákos, Balogh, György, and Molnár, Károly. 2005. Radio Interferometric Geolocation. *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, pages 1–12.

MLQI. 2009. *The MultiHopLQI protocol*. http://www.tinyos.net/tinyos-2.x/tos/lib/net/lqi

Montenegro, G., Kushalnagar, N., Hui, J., and Culler, D. 2007. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. Network Working Group RFC4944.

Mun, M., Reddy, S., Shilton, K., Yau, N., Boda, P., Burke, J., Estrin, D., Hansen, M., Howard, E., and West, R. 2009. PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. *Proceedings of the 7th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '09)*.

Nath, Suman, Gibbons, Phillip B., Seshan, Srinivasan, and Anderson, Zachary R. 2004. Synopsis Diffusion for Robust Aggregation in Sensor Networks. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 250–262.

Paek, Jeongyeup, and Govindan, Ramesh. 2007. RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks. *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys '07)*, pages 305–319.

Polastre, Joseph, Hill, Jason, and Culler, David. 2004. Versatile Low Power Media Access for Wireless Sensor Networks. *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*.

Polastre, Joseph, Szewczyk, Robert, and Culler, David. 2005a. Telos: Enabling Ultra-Low Power Wireless Research. *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS '05)*.

Polastre, Joseph, Hui, Jonathan, Levis, Philip, Zhao, Jerry, Culler, David, Shenker, Scott, and Stoica, Ion. 2005b. A Unifying Link Abstraction for Wireless Sensor Networks. *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys '05)*.

Rangwala, Sumit, Gummadi, Ramakrishna, Govindan, Ramesh, and Psounis, Konstantinos. 2006. Interference-Aware Fair Rate Control in Wireless Sensor Networks. *SIGCOMM Comput. Commun. Rev.*, **36**(4), 63–74.

Saltzer, J. H., Reed, D. P., and Clark, D. D. 1984. End-to-End Arguments in System Design. *ACM Trans. Comput. Syst.*, **2**(4), 277–288.

Simon, Gyula, Maróti, Miklós, Lédeczi, Ákos, Balogh, György, Kusy, Branislav, Nádas, András, Pap, Gábor, Sallai, János, and Frampton, Ken. 2004. Sensor Network-Based Countersniper System. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 1–12.

The ZigBee Alliance. 2009. *The ZigBee Alliance*. http://www.zigbee.org

Tolle, Gillman, Polastre, Joseph, Szewczyk, Robert, Culler, David, Turner, Neil, Tu, Kevin, Burgess, Stephen, Dawson, Todd, Buonadonna, Phil, Gay, David, and Hong, Wei. 2005. A Macroscope in the Redwoods. *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys '05)*.

van Dam, T., and Langendoen, K. 2003. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*.

Werner-Allen, Geoff, Lorincz, Konrad, Johnson, Jeff, Lees, Jonathan, and Welsh, Matt. 2006. Fidelity and Yield in a Volcano Monitoring Sensor Network. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*.

Woo, Alec, Tong, Terence, and Culler, David. 2003. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*.

Ye, Wei, Heidemann, John, and Estrin, Deborah. 2002. An Energy-Efficient MAC protocol for Wireless Sensor Networks. *Proceedings of the IEEE Conference on Computer Communications (Infocom '02)*, pages 1567–1576.

Zhang, Pei, Sadler, Christopher M., Lyon, Stephen A., and Martonosi, Margaret. 2004. Hardware Design Experiences in ZebraNet. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 227–238.