Typhoon: A Reliable Data Dissemination Protocol for Wireless Sensor Networks

Chieh-Jan Mike Liang, Răzvan Musăloiu-E., and Andreas Terzis

Computer Science Department
Johns Hopkins University
{cliang4,razvanm,terzis}@cs.jhu.edu

Abstract. We present Typhoon, a protocol designed to reliably deliver large objects to all the nodes of a wireless sensor network (WSN). Typhoon uses a combination of spatially-tuned timers, prompt retransmissions, and frequency diversity to reduce contention and promote spatial re-use. We evaluate the performance benefits these techniques provide through extensive simulations and experiments in an indoor testbed. Our results show that Typhoon is able to reduce dissemination time and energy consumption by up to three times compared to Deluge. These improvements are most prominent in sparse and lossy networks that represent real-life WSN deployments.

1 Introduction

One of the main end-user requirements for WSNs is the ability to reprogram the network after it has been deployed. In turn, the requirement to reprogram the network generates the need to reliably disseminate large objects (∼50–100 KB) to every node in the network. This combination of large object sizes, 100% reliability, and network-wide distribution is not addressed by other WSN protocols and thus requires a custom protocol. This need has been identified by numerous researchers in the past (*e.g.*, [3,4,5,13,16] among others).

In this paper we present *Typhoon*, a reliable data dissemination protocol that represents a different set of choices in the design space. Our choices are motivated by the observation that *idle listening* is the major consumer of energy during dissemination. Thereby, all protocol decisions should be geared towards minimizing the time that nodes are not transmitting or receiving data packets (*i.e.* competing to request or waiting for the retransmission of a lost packet).

Unlike previous protocols, Typhoon sends data packets via unicast. This approach allows receivers to acknowledge the receipt of individual packets and thereby quickly recover lost packets. While data packets are sent via unicast, interested nodes can still receive them by *snooping* on the wireless medium. Through the combination of unicast transfers and snooping, Typhoon achieves the best of both worlds—prompt retransmissions and data delivery to all the nodes in a broadcast domain through a single transmission. Dissemination latency is also reduced by exploiting spatial reuse, through which nodes in different parts of the network can be transmitting at the same time. We enhance spatial

R. Verdone (Ed.): EWSN 2008, LNCS 4913, pp. 268–285, 2008.

reuse through the combination of two techniques: setting timers in a way that encourages nodes further from the origin to propagate the object and the use of channel switching. Specifically, it has been shown that the minimum node distance necessary to avoid interference among concurrent transmissions is three hops [2]. On the other hand, if nodes switch frequency channels[1] during data transfer it is possible to reduce the distance to two hops in many cases. Typhoon leverages this observation to reduce object dissemination time.

We evaluate the performance of Typhoon through a combination of simulations and experiments on a testbed deployed in an office building. Performance is measured in terms of the time required and the energy expended to deliver an object to the whole network. We vary the size, diameter, and density of the network and test Typhoon using different object sizes and loss rates to understand the effects of these factors on the protocol's behavior. Moreover, we compare Typhoon's performance to that of Deluge—the de facto standard for data dissemination in TinyOS [3]. Our results show that Typhoon can be up to three times faster than Deluge in sparse and lossy networks.

This paper has five sections. We summarize related work in the section that follows and provide a detailed description of the Typhoon protocol in Section 3. We evaluate the protocol's performance and compare it with previous protocols proposed in the literature in Section 4. Finally, Section 5 outlines future research directions.

2 Related Work

The problem of designing protocols for reliably disseminating large data objects has received considerable attention in the past. One can divide existing protocols in two broad categories: randomized protocols in which nodes compete to acquire and subsequently transmit parts of the object, and protocols that avoid contention by scheduling node transmissions.

The genealogy of the first protocol family starts with PSFQ [18], a transport protocol for reliable delivery of objects from a sink to all the nodes in a wireless sensor network. PSFQ uses TTL-scoped broadcast to propagate messages from the sink and hop-by-hop retransmissions to recover from lost messages. Unlike PSFQ, Typhoon uses unicast messages to propagate objects, while leveraging overhearing to deliver packets to multiple receivers within the same broadcast domain. Moreover, PSFQ uses negative acknowledgments, whereas Typhoon uses postive acknowledgments and multiple frequency channels to increase spatial reuse. MOAP [16] transfers the complete object one hop at a time. After receiving the whole object a node can become a secondary source, delivering it to nodes further away from the origin. The design of MOAP is driven by the desire to trade latency for reliability and simplicity. Unlike MOAP, Typhoon uses pipelining in which nodes offer to further deliver *pages*) (*i.e.*, subsets of the object) as soon as they receive them. This approach dramatically reduces the network completion time, defined as the time by which all nodes receive the

[1] Current 802.15.4 radios can switch between 16 non-overlapping channels.

full image, thereby reducing energy consumption due to idle listening. MNP [5] reduces download time by using pipelining and reduces contention in dense networks through the use of a sender selection algorithm. Reliability is achieved through retransmissions, initiated by query messages sent by the packet source to nodes receiving the transmissions. Unlike MNP, Typhoon implements opportunistic overhearing for traffic of common interest. Moreover, Typhoon uses fast acknowledgments transmitted after each packet rather at the end of a page. Finally, Typhoon uses channel switching to reduce contention in the broadcast medium, amplifying the benefits of spatial reuse.

Deluge [3] is the de facto standard for data dissemination in TinyOS. It uses an epidemic protocol that eventually propagates the object to all the nodes in the network. Deluge relies on randomized *Trickle* timers [10] to reduce contention among transmission requests. Objects are transmitted as sequences of fixed-size pages via broadcast to leverage the broadcast nature of the wireless medium. NACKs trigger the retransmission of lost messages after a full page has been transmitted. NACKs also use Trickle timers to minimize the probability that multiple retransmission requests will collide. While beneficial in reducing the number of collisions, random timers can prolong the time required to propagate the image throughout the network. Typhoon also delivers data to multiple receivers whenever possible. On the other hand, receivers send acknowledgments after each data message instead of NACKs after each block transmission. This design choice enables nodes to start offering data to downstream destinations sooner, thereby minimizing completion time and thus energy costs. This is especially important in lossy networks in which the number of retransmissions is expected to be high. Moreover, Typhoon uses channel switching to reduce contention and to allow multiple concurrent transmissions over the same broadcast domain.

Protocols of the second family initially distribute the object to a subset of the network's nodes using a fixed schedule that avoids overlapping transmissions. The object is then broadcasted to the rest of the network. In order to minimize completion time, the initial set of nodes should be the minimum connected dominating set (MCDS) of the graph induced by the wireless network [13]. Calculating that set however is an NP-hard problem even for the unit graph connectivity model [1] and therefore approximation algorithms are necessary. Sprinkler uses a distributed approximation algorithm that computes a connected dominated set that is a multiplicative factor larger than the MCDS [13]. Infuse [4] follows a similar dissemination strategy and combines it with implicit acknowledgments for reliability. Furthermore, Infuse turns off the radios of nodes not participating actively in the dissemination thus reducing energy consumption due to idle listening. GARUDA [14] is a recent protocol that uses an efficient mechanism for constructing an approximate MCDS during the first packet transfer. Moreover, GARUDA nodes publish bitmaps indicating the packets they have received correctly. Downstream nodes use these bitmaps to send (re)transmission requests. Unlike protocols that rely on node coordination to prevent contention, Typhoon minimizes contention through the use of channel switching and implicit

synchronization. This approach does not have the overhead of building the MCDS, is robust to node failures, and simplifies data dissemination to new nodes in the network.

3 Protocol Description

Typhoon is designed to reliably deliver large objects, such as code binaries, to all the nodes in a WSN. In this context, large objects are defined as objects that do not fit in the mote's main memory and can be as large as 50–100 KB. Typhoon divides an object to fixed-size pages (1 KB) which are further divided to fixed-size packets (28 bytes in our implementation) that can be atomically transmitted over the radio.

Even though protocols like Typhoon are unlikely to be invoked frequently, their inherent flooding nature and the need for 100% reliability, irrespective of loss conditions suggest that each invocation of the protocol could be resource intensive and thus its cost should be minimized. As has been argued before, idle listening is one of the largest energy consumers [19]. Therefore, the protocol should make every effort to "push" the object's pages through the network as fast as possible. In turn this means that the protocol should attempt to leverage spatial re-use, transmitting pages from multiple non-overlapping nodes and minimize contention that leads to node back-offs and thereby added latency.

We note that an alternative approach would be to use duty cycling, turning radios off when not in use. In this case network completion time is not as crucial, because energy consumption due to idle listening is minimized. However, we argue that duty cycling is not appropriate for reliable dissemination protocols. First, users want to reduce network downtime due to reprogramming. Second, duty cycling introduces complexity which should be minimized in protocols that serve a critical role to network operations.

3.1 Metadata Dissemination

We assume that the object to be disseminated is injected through an out-of-band mechanism to a single node from which it must propagate to the network. In this regard, the first necessary step is to notify the network about the existence of this new object. Typhoon uses separate mechanisms to disseminate data objects and metadata about these objects. By metadata, we mean information about the existence of a new object, codified into an object ID, size and version. Nodes decide whether they should attempt to download an advertised object by comparing the new object ID and version with those of previously retrieved objects. If a node decides to download the new object, the number of pages is determined by dividing the object's size by the page size.

The reason for using separate mechanisms stems from the difficulty of designing a single protocol that can efficiently serve both purposes. For example, since new nodes may join the network at any time, the metadata dissemination protocol must be always active. This means that, while it should quickly propagate

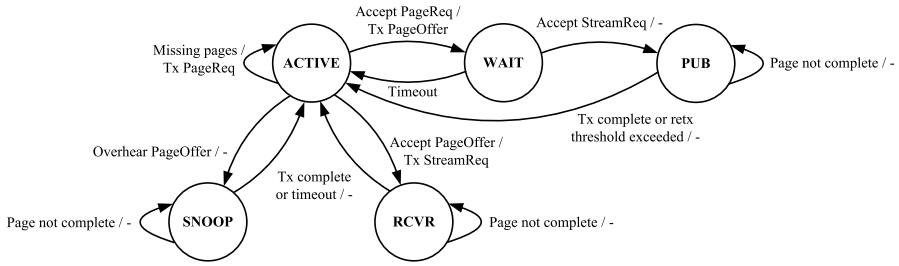


Fig. 1. State transition diagram for Typhoon. State transitions are marked using the condition/action notation in which a transition occurs when a condition is met and results in an action (or no action in case of '-').

updates to the whole network, it must minimize overhead during steady state. On the other hand, for reasons outlined above, the data dissemination protocol should disseminate the object as fast as possible and then terminate. Typhoon uses Trickle [10] to disseminate metadata.

For the remainder of the section we describe what happens once nodes become aware of the existence of a new object and attempt to retrieve it.

3.2 Data Request Handshake

Figure 1 represents Typhoon's state transition diagram. Nodes start in the ACTIVE state and return to this state while they have more pages to download. While in this state, a node will periodically broadcast `PageReq` requests that contain the object's ID and the number of the requested page. Nodes request pages sequentially. By doing so, nodes within the same broadcast domain are more likely to be in the same state, which increases the probability of overhearing traffic of common interest.

The broadcast period is uniformly chosen from $[t_a, t_b]$ to avoid collisions among multiple interested receivers[2]. Nodes that have copies of the requested page and receive a `PageReq` message, each respond with a unicast `PageOffer` message after waiting for a random time uniformly selected from $[t_c, t_d]$. The `PageOffer` message includes the object's ID as well as the number of the page offered. The random waiting period is used to prevent collisions among multiple potential offerers. They then transition to the WAIT state and wait for a `StreamReq` message. If no `StreamReq` arrives within T_s seconds the offerers return to the ACTIVE state[3]. Otherwise, upon receiving a unicast `StreamReq` message, one of the offerers will transition to the PUB state and start the data transfer. That offerer returns to the ACTIVE state after the page has been successfully downloaded or after a number (five) of unsuccessful data packet transfers. These failures are detected because the receiver acknowledges the receipt of individual data packets (see Section 3.3).

[2] We use, $[t_a, t_b] = [400, 500]$ msec.
[3] $T_s = 20$ msec in our implementation.

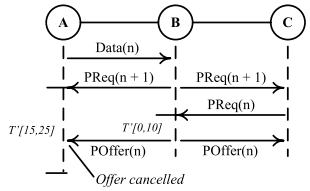


Fig. 2. Pipelining pages through the network

Conversely, a node that receives a `PageOffer` message matching its request, transitions to the RCVR state and signals the source of the `PageOffer` message to initiate the data download by transmitting a unicast `StreamReq` message. The receiver stays in that state while more packets from the requested page need to be retrieved and returns to the ACTIVE state either when the whole page has been successfully downloaded or when a timeout occurs. The second case protects the receiver against failures of the transmitting node.

Nodes that overhear a `PageOffer` message for a page they are missing, will transition to the SNOOP state in which they will attempt to receive the data packets from the offered page. While `PageOffer` messages are sent via unicast, interested nodes can still receive them. For example, the CC2420 radio provides the ability to disable *address filtering* enabling a node to receive all packets irrespective of their destination address. Similar to the RCVR state, the node leaves the SNOOP state when the page transfer has completed or when a timeout occurs. If a node does not successfully overhear all the packets from a page, it discards the page.

In addition to the base scheme described above, Typhoon optimizes its use of timers to enable the pipelining of pages through the network. We describe this optimization using the example presented in Figure 2. In this scenario, node A has finished transmitting page n to node B. In response, node B will transition to the ACTIVE state and transmit a `PageReq` for page $n + 1$. Node A receives this message and starts its timer to transmit the `PageOffer` message. However, node C also receives the request and deduces that node B already has page n (because pages are downloaded sequentially). C then sends its own `PageReq` for page n to B. From the perspective of pipelining, C's request has priority over B's original request, since it pushes pages further downstream. To encourage this behavior, Typhoon sets the timer at B to fire before A's timer[4]. Once B's timer expires, it transmits a `PageOffer` for page n. A overhears that offer and cancels its own `PageOffer`, implicitly deferring to B's data transmission.

3.3 Data Transfer

Typhoon achieves reliable transfer in the face of packet loss, through the use of retransmissions. However, unlike previous protocols that use negative

[4] In our implementation, $[t_c, t_d] = [15, 25]$ msec for a node that has just finished transmitted a page and $[0, 10]$ msec otherwise.

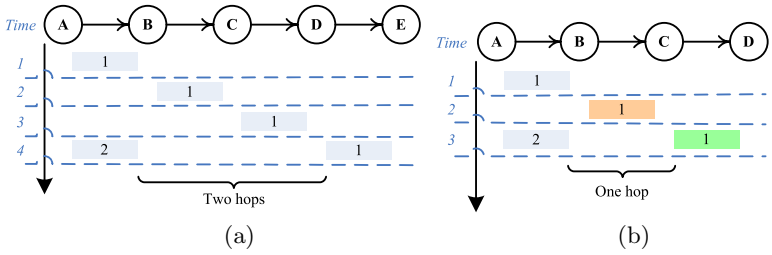


Fig. 3. (a) Propagation of consecutive pages on a linear topology when only one frequency channel is used. Notice that node A has to wait until time period 4 to transmit the second page in order to avoid colliding at B with node C's transmission of the first page. (b) When nodes can use different frequency channels to transmit data packets (indicated by different colors in the figure) the wait time is reduced by one time period.

acknowledgments after all packets in page have been transmitted, Typhoon acknowledges the receipt of individual data packets. If the sender does not receive an acknowledgment, it retransmits the last data packet thus implementing a stop-and-wait ARQ protocol.

A node can generate these acknowledgments in two different ways. First, modern radios offer the ability to automatically generate hardware acknowledgments [17]. The benefit of this approach is reduced latency because the ACK is generated as soon as the radio hardware correctly receives the packet. On the other hand, it is possible for an acknowledged packet to be dropped before it reaches the application. In this case, the hardware acknowledgment results in a false positive. Fortunately, TinyOS2 [8], on which Typhoon is developed, implements a mechanism called software ACK that can trigger this acknowledgment at the system level. It is thus possible to disable the hardware from automatically generating hardware ACKs and achieve equivalent functionality using software ACKs.

An additional benefit of disabling hardware ACKs is that it enables overhearing of unicast packets. This is because enabling hardware ACKs in the commonly-used CC2420 radio also enables destination address filtering, in which case the radio automatically discards all unicast frames not destined to the current node. With address filtering disabled, nodes in the SNOOP state can still receive data packets sent to the unicast address of the node that transmitted the `StreamReq` message, while the explicit receiver will generate ACKs for those data packets.

3.4 Channel Switching

As we already argued, data dissemination protocols should leverage spatial reuse to accelerate the propagation of pages through the network. Spatial re-use is achieved by having nodes retransmit pages as soon as they arrive. However, as Figure 3(a) demonstrates, in order to avoid collisions due to the hidden terminal problem a node must wait for two additional periods (a period is defined as the amount of time necessary to transmit a page) before it can transmit the next page. On the other hand, as Figure 3(b) shows, this bound can be further

reduced if nodes have the ability to transmit at different frequency channels. Channel switching provides another benefit in addition to accelerating the pipelining process. Because nodes exchange `PageReq` and `PageOffer` messages on the default common channel, having data transfers on different frequencies eliminates the danger of ongoing data transfers colliding with these control messages.

Considering the advantages of channel switching, Typhoon incorporates it to the data request handshake described above. Rather than using an explicit agreement protocol in which nodes are assigned specific frequencies, Typhoon employs a randomized scheme to select transmission frequencies. Specifically, the publisher suggests a frequency channel in its `PageOffer` message by randomly selecting from one of the possible channels (*e.g.* 15 in the case of 802.15.4, since one channel is reserved for broadcast messages). If the receiver accepts the offer it replies with an acknowledgment (similar to the ACK used for data packets) and switches to the suggested frequency channel. After receiving the acknowledgment the publisher also tunes to the new channel and the data transfer starts. Note that the receiver transmits a `StreamReq` message after switching to the channel indicated in the `PageOffer` message. Although the channel is randomly chosen, it is still possible to have multiple publishers willing to serve the same receiver on the same channel. Therefore, the `StreamReq` message serves as an explicit indication of the receiver's decision. Although nodes randomly select data transfer channels, it is possible that more than one ongoing data transfers with overlapping radio coverage take place on the same channel. In this case, interference can cause higher packet loss and thus retransmissions and possibly failure to transmit the page due to the loss of multiple acknowledgments. In the second case, the sender and/or the receiver will timeout, return to the ACTIVE state, and retry downloading the original page.

While channel switching provides clear performance benefits, it also introduces new complications. For example, Typhoon uses Trickle for metadata dissemination, and both Typhoon and Trickle can be active at the same time. Since Trickle is not aware of the channel changes it will transmit over the channel selected by Typhoon. This means that if a node is transferring data on a channel other than the default one, the node's neighbors will not be able to receive any metadata sent via Trickle. Realizing this conflict, we implement two schemes to minimize its effects. First, upon receiving the initial notification via Trickle, nodes wait for a random period before they start Typhoon[5]. This delay allows Trickle to propagate the metadata downstream. Second, nodes switch to the default channel immediately after each page transfer, thus allowing the continued dissemination of metadata.

4 Evaluation

4.1 Evaluation Metrics and Methodology

We evaluate the performance of Typhoon using simulations and experiments performed on a testbed deployed in an office building. The results we report

[5] Set to $[400, 500]$ msec in our implementation.

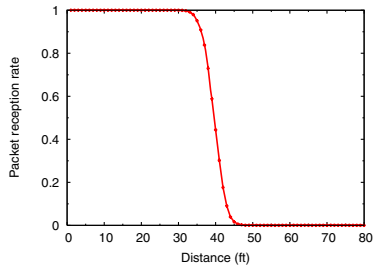


Fig. 4. Packet reception rate as a function of distance from a packet source. The path-loss exponent is 4.

are based on an implementation of Typhoon built on top of TinyOS 2 (T2) [8]. Moreover, we use the standard CSMA MAC protocol used in T2.

We use Deluge, the de facto standard for reliable bulk transfer in TinyOS, as the baseline for our comparisons. Since Deluge provides no guidelines for setting its parameters under different network conditions we use the default parameters provided with Deluge under all cases. All simulations were carried out in TOSSIM, a discrete event based simulator for TinyOS [9]. We leverage two of TOSSIM's features to improve the fidelity of our simulations. First, TOSSIM allows defining signal attenuation levels on a per link basis. We calculate these attenuations using the log distance path loss model [15]. In this model the path loss at distance d from the source, measured in dB, is, $PL(d) = PL(d_0) + 10n\log(d/d_0)$, where n is the path-loss exponent and $PL(d_0)$ is an experimentally measured path loss at reference distance d_0. Path loss exponent $n = 2$ corresponds to free space propagation, while $n = 3, 4$ model environments with reflections and refractions [15]. We use $n = 4$ for all our simulations. Figure 4 shows the packet reception rate at various distances from a source node. Second, we utilize TOSSIM's ability to emulate bursty noise due to interference.

We quantify the performance of Typhoon through two metrics: **(1) Completion time**, which captures the time necessary to disseminate an object. We measure both the time necessary for individual nodes as well as the network completion time, defined as the longest node completion time. **(2) Power consumption.** While completion time quantifies the level of disruption from executing the object dissemination protocol (assuming the network's operation is disrupted during the download), power consumption quantifies the impact of data dissemination on the network's lifetime.

Due to the lack of a direct mechanism for measuring power consumption in TOSSIM, we use the indirect approach of measuring the amount of time the nodes spend transmitting, in idle listening mode, as well as the number of packets it receives. Because the Tmote Sky data sheet [12] publishes only the current drawn in transmit mode (17.4 mA), and in idle listening mode (19.7 mA), we experimentally measured using a Tmote Sky mote [11] the average current drawn while receiving one packet to be 21.7 mA. Note that our energy estimates do not include the costs of reading and writing to flash. The reason is that

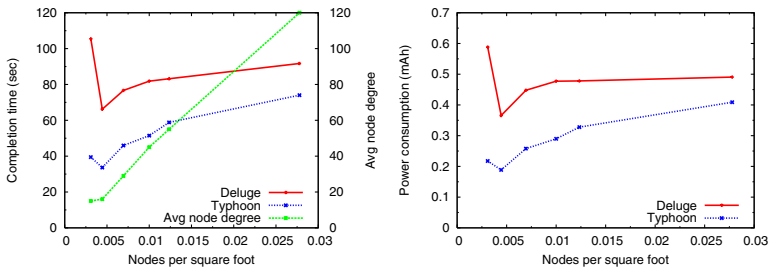
they represent a fixed cost which is orthogonal to the operation of the data dissemination protocol and therefore it provides no insight into the impact of different protocol design decisions.

We run each experiment five times and use the two evaluation metrics to reason about the impact of different factors on the performance of Typhoon. Specifically, we investigate the impact that network density and size, object size, and loss rate have on data dissemination. Moreover, we evaluate the incremental benefits of overhearing and channel switching in Typhoon. Finally we present the behavior of Typhoon in practice through results from a small testbed.

4.2 Effect of Network Density and Size

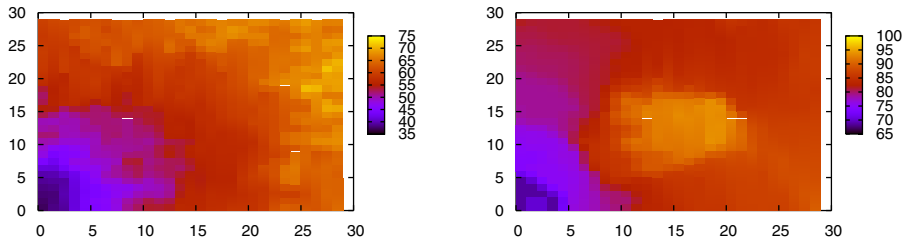
Network density is a critical performance factor since it affects the level of contention when requesting and downloading pages. We first discuss the impact of network density on completion time. Figure 5(a) shows the effect of increasing the number of nodes per square foot by increasing the size of an $N \times N$ node grid, deployed on a fixed 180 × 180-foot field. Also shown in the same figure is the average node degree, defined as the set of nodes with $PRR > 0$, as network density increases. One can make two observations from this figure. First, the performance margin between Typhoon and Deluge increases in sparse networks. This is because Deluge uses timer values that reduce the number of messages sent and increase the probability of overhearing. However, in sparse networks, these timer values increase the idle listening time and thus completion time. Second, Typhoon is consistently faster throughout the density range despite its more aggressive timers. This indicates that channel switching is effective in relieving channel contention.

Both Typhoon and Deluge require nodes to keep their radios on for the duration of the data dissemination. Considering that the radio consumes considerable energy in idle listening state, completion time will influence energy consumption. Figure 5(b) verifies this intuition as it shows that energy consumption follows closely completion time. We found that for both protocols nodes spend less than 7% of their time transmitting further indicating that energy cost is dominated



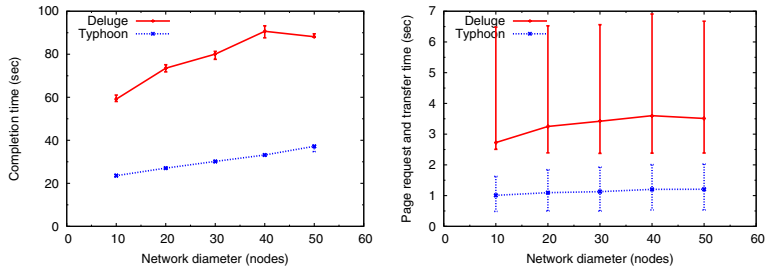
(a) Completion Time (b) Average power consumption

Fig. 5. Average network completion time and average node power consumption for a 20 KB object, as a function of network density. Network nodes are placed on a grid over a 180 × 180-foot field.



(a) Typhoon (b) Deluge

Fig. 6. Node completion time for Typhoon and Deluge on a 180 × 180-foot field. The field had 30^2 nodes uniformly distributed with a density of 0.028 nodes per square foot. A 20 KB object was initially injected at the bottom left corner of the field.



(a) (b)

Fig. 7. (a). Network completion time of a 20 KB object, as a function of the diameter changes in $1 \times n$ linear topology. (b). Page acquisition time, including the page request phase and subsequent data transfer. The vertical lines represent the 5^{th} and 95^{th} quartiles.

by idle listening time. With this result in mind, we present only completion times for the remainder of the evaluation.

Figure 6 illustrates the propagation time for individual nodes in a dense grid. As reported in [3], Deluge propagates the data object faster around the edges than in the middle of the network. The main reason is that nodes in the middle of the network have more neighbors and thus higher probability of collisions. On the other hand, Typhoon generates a uniform wavefront pattern from corner to corner. Although nodes in the middle have more neighbors, the only messages broadcasted on the default channel are the first two handshake messages. The probability of collision is thus lower than Deluge.

Unlike the grid topology in which a node might receive data from different neighbors, the linear topology limits the propagation to only one direction. It is therefore easier to study the effects of network size on completion time using linear topologies.

A number of interesting observations can be made from Figure 7(a) that plots completion time as a function of network diameter in a linear topology. First,

both Typhoon and Deluge benefit from pipelining, and the completion time does not increase at the same rate as the number of nodes. Second, Deluge exhibits faster increase compared to Typhoon. As the network diameter increases, the number of neighboring nodes for some nodes also increases, and thus the probability of contention increases. This has a larger influence on Deluge, because Typhoon sends packets on the common channel only during the page request phase. Figure 7(b), which shows the average time to request and download a single page as the network's diameter increases, verifies this conjecture. From the similarity between the two graphs, it is easy to see that page acquisition time dictates completion time. Furthermore, Typhoon has approximately constant page transfer time in all cases, which suggests that the shorter page request phase underlies the difference in completion time. Finally, Deluge exhibits larger variability in page acquisition time, due to the varying levels of contention that different nodes experience.

4.3 Effect of Object Size

Unlike metadata dissemination protocols for which network diameter dominates completion time, the size of the object transferred affects the completion time of bulk data dissemination protocols. Figure 8 shows the impact of object size on completion time in two cases: a sparse linear topology in which nodes can reach only their immediate neighbors, and a 20×20 grid topology with 10-feet node spacing. In both cases, the completion time grows linearly with the object size with Deluge yielding a steeper slope.

To understand the root cause for this behavior, we briefly present a model for data dissemination in sparse linear topologies. We assume ideal conditions in which pages are transferred in perfect synchrony with no collisions. In this case, the expected completion time for Typhoon is $\hat{T}_t = 2(n-1)P_t + d \cdot P_t$, where n is the number of object pages, d is the network diameter, and P_t is the time to request and receive a page (see Figure 3). Given the description of Typhoon from Section 4, we can estimate P_t and thus $\hat{T}_t$. A page transfer is preceded by

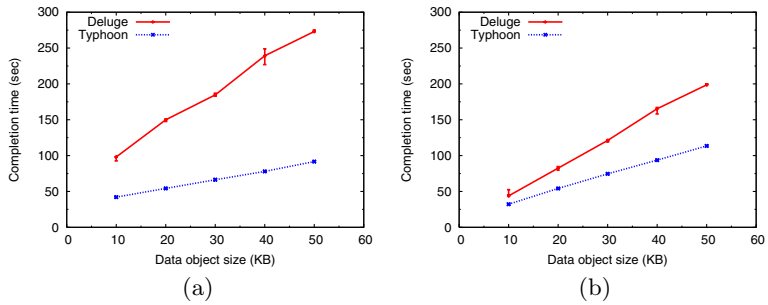


Fig. 8. Completion time as the object size varies in (a) 1×50 sparse linear topology where nodes can reach only their immediate neighbors, and (b) 20×20 grid topology with 10-feet node spacing

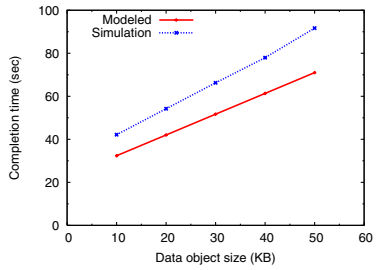


Fig. 9. Modeled and simulated completion time of Typhoon in 1×50 sparse linear topology

the data request handshake. According to TOSSIM, each handshake exchange of a 21-byte message followed by the ACK requires 1.68 msec to complete. Since the handshake consists of three messages and two back-off timers with maximum length of 25 msec each, it should take 55.04 msec. Moreover, according to TOSSIM, a page transfer requires approximately 428 msec and thus $P_t = 483.04$ msec. Figure 9 shows the modeled and simulated completion time for Typhoon with different object sizes. Since the modeled completion time is based on ideal conditions, it represents the lower bound on Typhoon's performance. At the same time, it explains that the lower completion time that Typhoon exhibits is due to the speedup that channel switching offers.

4.4 Impact of Packet Loss

Since reliability is a requirement for bulk data dissemination protocols completion time depends on how fast lost packets are recovered. We perform two experiments to estimate the effect of packet loss on completion time.

First, we increase the spacing between neighboring nodes in a 20×20 grid topology. This increase raises the path loss on the link and therefore decreases the packet reception rate (PRR). Figure 10 illustrates the completion time for this experiment. It is easy to see that Deluge performance deteriorates with distance while Typhoon is able to maintain consistent performance. Specifically, Deluge's completion time increases by over twofold when nodes are 35 feet apart from each other. This is due to the fact that the PRR of the links between neighboring nodes at this distance falls in the so-called *gray region* ($PRR =\sim 95\%$, as Fig. 4 indicates). Extending the inter-node distance even further leads to a precipitous decrease in PRR ($\sim$ 30% at 40 feet), leading to an even worse performance differential.

Second, we simulate the effect of bursty losses due to interference. To do so, we use TOSSIM noise traces collected from environments with heavy 802.11 use [6]. As Table 1 shows, Typhoon's performance degrades by 48% while the completion time for Deluge increases threefold. Two main reasons underlie this trend. First, Typhoon requires all data packets to be individually acknowledged, and it bases the retransmission decision on this acknowledgment instead of a

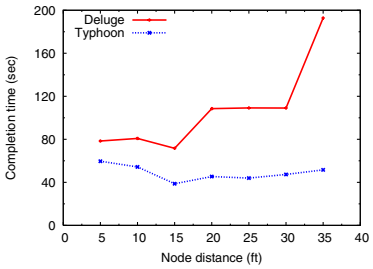


Fig. 10. Completion time as the inter-node distance varies in a 20×20-node grid topology

Table 1. Completion time under different loss environments for a 20×20-node grid topology with 10-feet node distance

	Quiet	Bursty loss
Typhoon	54.30	73.37
Deluge	80.79	241.43

timer. This allows lost packets to be recovered quickly. Second, compared to Deluge, Typhoon is more aggressive in sending packets, so the transfer moves at a faster pace.

4.5 Benefits of Overhearing and Channel Switching

In order to better understand the performance benefits that channel switching and overhearing offer, we selectively disable them in an experiment on a 5×5 grid topology.

Table 2 presents the results of this experiment. Disabling channel switching creates a larger performance deterioration compared to disabling overhearing. This degradation while large is expected because Typhoon assumes that data transfers take place on a channel that is free from interference caused by other data transfers and request handshakes. As a result, being aggressive hurts performance in this case. On the other hand, overhearing provides only modest improvement. The reason is that Typhoon performs opportunistic overhearing, in which nodes can snoop on a page transfer only when they overheard the preceding `PageOffer` message. In other words, if a node misses that message, it loses the opportunity to overhear since the transfer happens at another channel. Moreover, if a node in the SNOOP misses one or more packets from a page due to interference it discards the whole page. At the same time, when overhearing is combined with channel switching, it offers $\sim$ 30% reduction in completion time.

Table 2. Completion time as channel-switching and overhearing are disabled in a 5×5-node grid topology with 20-feet node spacing for s 3 KB object

	Completion time (sec)
Channel-switching and overhearing	6.24
Channel-switching only	8.79
Overhearing only	945.80
None	1016.43

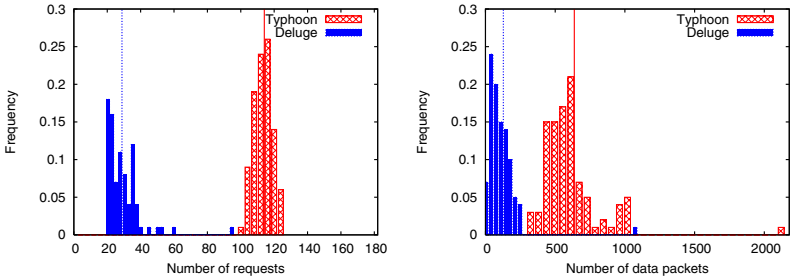


Fig. 11. Probability distribution of (a) Typhoon request messages and Deluge advertisements (b) Typhoon and Deluge data messages. The topology is a 10×10 node grid, with 10-feet node distance, and the object size is 20KB. The vertical lines show the average.

4.6 Protocol Overhead

The major design goal of Typhoon is to minimize completion time. It achieves this goal by being aggressive in requesting and transmitting object pages. Figure 11 illustrates the results of this aggressive behavior by comparing the per-node packet distributions for disseminating the same object using Typhoon and Deluge.

We focus on request and data transfer messages because they constitute the majority of traffic. Typhoon generates approximately three times more traffic than Deluge for both message types. The reason is that, unlike Deluge, Typhoon does not have a request suppression mechanism, so nodes broadcast requests more aggressively. Moreover, we found that 47% of the overhearing attempts failed (*i.e.* node had to discard the partially overheard pages). While one can suggest based on this result that nodes should sleep instead of performing opportunistic overhearing, sleep scheduling introduces complexity and overhead to the protocol. Furthermore, as Section 4.5 shows, overhearing when used in conjunction with channel switching, leads to $\sim 30\%$ reduction in completion time.

4.7 Testbed Evaluation

We complement the simulation results presented above, with experimental results from testing Typhoon and Deluge on a small testbed. While simulations are meant to explore the behavior of the protocols under various conditions, the testbed is used to compare their performance in a realistic environment. Given the two different goals, we do not compare results across simulations and the testbed. Rather, it is the relative performance of Typhoon and Deluge under the same testing scenarios that is of interest.

We test Typhoon on a testbed that consists of 22 motes deployed in an office building according to the topology shown in Figure 12. Due to the shape of the building, the testbed physically resembles a linear topology. Moreover, the center of the testbed around location 119 tends to have relatively bad connectivity to

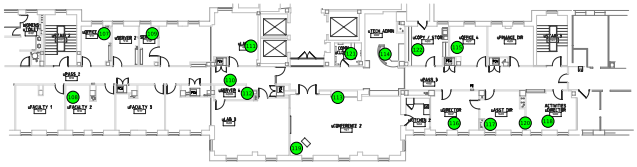


Fig. 12. The testbed floor plan shows the locations of Tmote Connect boxes, which can have either one or two motes attached

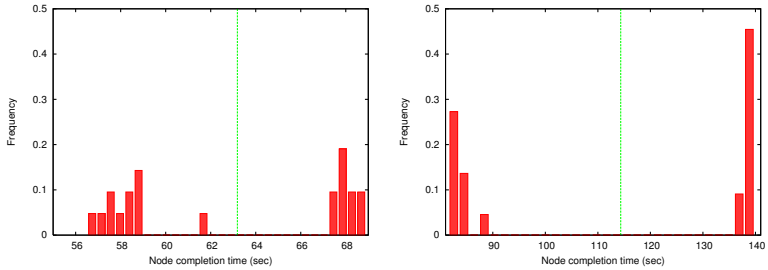


Fig. 13. PDF of node completion time on the testbed for (a) Typhoon and (b) Deluge. The green line shows the network average in each case.

the rest of the network. Dissemination starts by injecting a 20 KB object from location 118 on the right side of the testbed.

The average network completion time was 75.15 seconds using Typhoon and 145.57 seconds with Deluge. To understand how the object propagates through the network, Figure 13 shows the distribution of node completion times. For both protocols, the node completion time is divided into two groups, with one group taking longer to receive the entire object. Analysis of the experiment log shows that the group of slow nodes is located on the left side of the testbed. This is due to the the poor link connectivity in the center of the testbed. For example, in the case of Typhoon, most nodes on the left side of the testbed download pages from location 112. However, since the link connectivity between location 112 and nodes on the right side of the testbed was poor, location 112 becomes the bottleneck.

As explained above, Typhoon uses the Dissemination service to publish metadata and T2 components for reading/writing to the Flash. The combined code footprint of all three components is 14752 bytes of ROM and 413 bytes of RAM. At the same time, the incremental overhead of adding Typhoon to an application that uses Dissemination and the Flash is 3806 bytes of ROM and 112 bytes of RAM.

5 Looking Forward

We have shown how Typhoon leverages frequency diversity to reduce network contention and system-level ACKs to expedite recovery from lost data packets.

The combination of these two techniques provides significant performance benefits across a wide range of network sizes and conditions.

As we move forward, we plan to explore the benefits that dynamic packet size adjustment provides. Preliminary results from our testbed show that changing packet size can affect the packet reception rate by as much as 28%. The intuition is that the probability of bit errors and thereby corruption accumulates as the packet size increases. One should then transmit smaller packets in noisy environments to reduce the number of retransmissions and larger packets in 'quiet' environments to reduce packet overhead. However, the noise level is not known in advance and changes over time. While algorithms exist for dynamically adjusting the packet size to maximize throughput, they are unsuitable for WSNs due to their complexity [7]. We are currently developing algorithms for estimating the underlying bit error rates and dynamically adjusting the packet size that can be implemented on current generation motes.

Acknowledgments

We extend our gratitude to Prabal Dutta and the anonymous reviewers for their insightful comments and their help in improving this paper.

This research was supported in part by NSF grant CNS-0546648 and by the U.S. Department of Homeland Security (Grant Number N00014-D6-1-0991) through a grant awarded to the Center for Study of Preparedness and Critical Event Response (PACER) at the Johns Hopkins University. Any opinions, finding, conclusions or recommendations expressed in this publication are those of the author(s) and do not represent the policy or position of the Department of Homeland Security and the National Science Foundation.

References

1. Clark, B., Colbourn, C., Johnson, D.: Unit disk graphs. Discrete Mathematics 86, 165–177 (1990)
2. Couto, D.S.J.D., Aguayo, D., Bicket, J., Morris, R.: A High-Throughput Path Metric for Multi-Hop Wireless Routing. In: MobiCom 2003. Proceedings of the 9^{th} ACM International Conference on Mobile Computing and Networking (September 2003)
3. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: SenSys 2004. Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (November 2004)
4. Kulkarni, S.S., Arumugam, M.: Infuse: A TDMA Based Data Dissemination Protocol For Sensor Networks. Technical Report MSU-CSE-04-46, Michigan State University - Computer Science and Engineering (November 2004)
5. Kulkarni, S.S., Wang, L.: MNP: Multihop network reprogramming service for sensor networks. In: ICSCS. Proceedings of the 25^{th} IEEE international Conference on Distributed Computing Systems (June 2005)
6. Lee, H., Cerpa, A., Levis, P.: Improving Wireless Simulation Through Noise Modeling. In: IPSN 2007. Proceedings of the Sixth International Conference on Information Processing in Wireless Sensor Networks (2007)

7. Lettieri, P., Srivastava, M.B.: Adaptive frame length control for improving wireless link throughput, range, and energy efficiency. In: Proceedings of IEEE INFOCOM 1998 (1998)
8. Levis, P., Gay, D., Handziski, V., Hauer, J.-H., Greenstein, B., Turon, M., Hui, J., Klues, K., Cory Sharp, R.S., Polastre, J., Buonadonna, P., Nachman, L., Tolle, G., Culler, D., Wolisz, A.: T2: A Second Generation OS For Embedded Sensor Networks. Technical Report TKN-05-007. Telecommunication Networks Group, Technische Universitat Berlin (2005)
9. Levis, P., Lee, N., Woo, A., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS Applications. In: Proceedings of Sensys 2003 (November 2003)
10. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: Proceedings of NSDI 2004 (March 2004)
11. Moteiv Corporation. Tmote Sky. Available at `http://www.moteiv.com/products/tmotesky.php`
12. Moteiv Corporation. Tmote Sky Datasheet. `http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf`
13. Nail, V., Arora, A., Sinha, P.: Sprinkler: A Reliable and Energy Efficient Data Dissemination Service for Wireless Embedded Devices. In: RTSS 2005. Proceedings of the 26^{th} International Real-Time Systems Symposium (2005)
14. Park, S.-J., Vedantham, R., Sivakumar, R., Akyildiz, I.F.: GARUDA: Achieving Effective Reliability for Downstream Communication in Wireless Sensor Networks. The IEEE Transactions on Mobile Computing (to appear, 2007)
15. Rappaport, T.S.: Wireless Communications: Principles & Practices. Prentice-Hall, Englewood Cliffs (1996)
16. Stathopoulos, T., Heidemann, J., Estrin, D.: A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing (November 2003)
17. Texas Instruments. 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver (2006). Available at `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`
18. Wan, C., Campbell, A., Krishnahmurthy, L.: PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks. In: Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (September 2002)
19. Ye, W., Heidemann, J., Estrin, D.: An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In: Proceedings of IEEE INFOCOM 2002 (2002)