Cacades: A Reliable Dissemination Protocol for Data Collection Sensor Network

Yang Peng† WenZhan Song† Renjie Huang†
Mingsen Xu† Behrooz Shirazi† Richard LaHusen‡ Guangyu Pei⋆
†Sensorweb Research Laboratory, Washington State University
‡Cascades Volcano Observatory, U.S. Geological Survey
⋆Boeing Phantom Works.
yang_peng@wsu.edu songwz@wsu.edu renjie_huang@wsu.edu
mingsen_xu@wsu.edu shirazi@wsu.edu rlahusen@usgs.gov Guangyu.pei@boeing.com

Abstract—In this paper, we propose a fast and reliable data dissemination protocol *Cascades* to disseminate data from the sink(base station) to all or a subset of nodes in a data collection sensor network. *Cascades* makes use of the parent-monitor-children analogy to ensure reliable dissemination. Each node monitors whether or not its children have received the broadcast messages through snooping children's rebroadcasts or waiting for explicit ACKs. If a node detects a gap in its message sequences, it can fetch the missing messages from its neighbours reactively. *Cascades* also considers many practical issues for field deployment, such as dynamic topology, link/node failure, etc.. It therefore guarantees that a disseminated message from the sink will reach all intended receivers and the dissemination is terminated in a short time period. Notice that, all existing dissemination protocols either do not guarantee reliability or do not terminate [1, 2], which does not meet the requirement of real-time command control. We conducted experiment evaluations in both TOSSIM simulator and a sensor network testbed to compare *Cascades* with those existing dissemination protocols in TinyOS sensor networks, which show that *Cascades* achieves a higher degree of reliability, lower communication cost, and less delivery delay. [1] [2]

TABLE OF CONTENTS

1. INTRODUCTION

Disseminating command and control information to nodes in the network is often an important part of many sensor network applications. The reliability and speed of data dissemination are particularly important for real-time feedback-driven sensing applications, such as precise agriculture and volcano monitoring sensor web. Due to the important role that each individual sensor node plays in the overall network operations, it is vital that all intended sensor nodes receive the command and control information. Additionally, the data dissemination must be accomplished and terminated within an acceptable time period determined by specific applications. For instance, in a volcano monitoring application [3], science agent software requires to change all seismic sensors' sampling rate from 100 Hz to 120 Hz. It is important that this command should reach all target stations in allowed time period. Network code image update is another important application which demands reliable and fast data dissemination. Several code propagation protocols, such as Deluge [1] and Trickle [2], have been developed. However, as reported in [4], although Deluge works in lab environment, it does not work well in field deployment with sparse topology and can not provide 100% reliable code propagation in a short time period. It is based on gossip routing, which may result in long convergence time particularly in sparse networks. Trickle also has similar problems.

Data dissemination in wireless networks is achieved via each node broadcasting the data to its neighbours, taking advantage of the shared wireless medium. A naive approach of broadcasting is *blind flooding*, in which each node is required to rebroadcast a packet whenever the packet is received for the first time. Blind flooding is therefore simple and easy to be implemented. However, it is likely to generate many redundant transmissions, which can cause severe broadcast storm congestion. Furthermore, it cannot guarantee broadcast reliability. Reliability of dissemination can be defined as the percentage of all nodes that successfully receive the injected packets [5]. Unreliable flooding can result in slow setup for query-response applications, slow discovery of new routes, or incomplete information for target tracking or signal processing. And more importantly, unreliable data dissemination is unacceptable in code update and user command & control operations. Reliable unicast in wireless networks is often implemented via link-layer ARQ (automatic repeat-request, often via RTS-CTS-data-ACK exchange) to guarantee packet delivery. However, using unicast for data dissemination will cause traffic implosion and low efficiency. Prior work has

[1] 978-1-4244-2622-5/09/$25.00 ©2009 IEEE.
[2] IEEEAC Paper #1114, Version 5, Updated 22 Oct 2008

sought to improve broadcast reliability, either by reducing sources of loss or providing broadcast retransmission. Examples of loss reduction include PHY-layer capture [6], MAC-layer optimization [7–9], and application-layer jitter. These approaches mitigate transmission collision, but can not prevent packet loss completely, particularly when the topology is sparse. We will discuss more related works in section 2.

In this paper, we present a fast and reliable data dissemination protocol, named *Cascades*, for the sink-initiated data dissemination. *Cascades* uses the parent-monitor-children analogy to ensure 100% reliability. Each node monitors whether or not its children have received each broadcast message through implicit listening techniques (e.g., snoop child's broadcast) or explicit ACKs. If not, the parent will retransmit until confirmed. In addition, if a child node detects a gap in its packet sequence, it can fetch quickly from any neighbour reactively. Hence, it guarantees that a disseminated message from the sink will reach all intended receivers in wireless network. We conducted experiment evaluations in both TOSSIM simulator and an indoor testbed by comparing to those popular dissemination protocols in TinyOS sensor networks.

The rest of the paper is organized as follows. In Section 2, we review recent research work on wireless broadcast in sensor networks. In Section 3, we describe the proposed *Cascades* protocol for reliable and fast data dissemination. We then evaluate the system performance in Section 4. Finally, we conclude the paper in Section 5.

2. RELATED WORKS

Much research has been done on wireless broadcast in the literature. Their major concerns include energy conservation and reliability. RBP [10] aims at providing adaptive reliability for broadcasts in varying network densities and improving the end-to-end reliability of flooding. It is distributed in the sense that every node makes its own decisions on retransmissions without any global or hard state. However, it does not offer 100% reliability guarantees. Other broadcast protocols such as [11] focus on minimizing the energy consumption of the nodes, through the minimization of the transmission cost of each broadcast in order to prolong the lifetime of the entire network. SmartGossip [12] is a probabilistic protocol that offers broadcast service with low transmission cost. SmartGossip automatically and adaptively adjusts transmission probability based on the underlying network topology. Its goal was to minimize energy consumption by reducing redundant transmissions. However, it cannot provide 100% reliability neither.

Bayesian-Assisted Resource Discovery In Sensor Network, BARD, [13], uses stochastic processes to model both the searching and the routing components in the data dissemination process and it also uses prior history knowledge and the characteristic configuration of diffusion to avoid unnecessary flooding of related queries. When historic information is available, BARD is quickly and efficiently able to perform; however, BARD is limited by traditional flooding techniques when this information is not available. Spatiotemporal Multicasting or mobicast, [14], allows the delivery of messages to a group of mobile destinations. In order to accomplish their goals of delivering the data to a mobile destination, it was necessary that the reliability constraints for mobicast be reduced.

[9] classifies existing broadcast schemes into categories and simulates a subset of each category, providing a condensed but comprehensive side by side comparison. It determines the strength and weakness of each type as well as the situation(s) in which each type is most suitable. In [15], the author investigated and discovered tight bounds on both the broadcast capacity and the information diffusion rate in wireless networks for both dense and extended populations. $\Theta(log(n)^{-\frac{\alpha}{2}})$ was determined to be a lower bound of broadcast rate for a continuous data stream using a multi-hop implementation in extended networks. Furthermore, the authors also proved that constant diffusion, $\Theta(1)$, of data is possible in both dense and extended networks.

PSFQ (Pump Slowly, Fetch Quickly) [16] is a reliable transport protocol. It guarantees the reliability of downstream transmissions by fast fetching packets from neighbouring nodes after a packet sequence gap is detected. When loss event is triggered, one NACK message will be broadcasted. However, PSFQ requires the sink node inject messages into network at a relative slow rate. The authors also indicate the importance of hop-by-hop error recovery where loss detection and recovery should be limited to a small number of hops (ideally one hop) due to the probability of losing packets accumulates exponentially with the number of hops.

GARUDA [17] uses Core-Recovery idea to implement reliable downstream data delivery. Some nodes in the network play the role as loss recovery servers, and other non-core nodes need to have one-hop connection with at least one core nodes. GARUDA works in two-stage recovery: in the first stage, downstream core nodes recover missing packets from upstream core nodes when packet loss is detected; in the second stage, non-core nodes recover missing packet from their neighbour core nodes.

Trickle [2] uses a probabilistic or "gossiping" approach which utilizes meta-data in order to discover and repair missing data. Hence, theoretically, it eventually provides 100% reliability. But it might take a long time to converge. Deluge [1] builds off Trickle [2] and is the default code update protocol in TinyOS. In Trickle, nodes stay up-to-date by occasionally broadcasting a code summary to their neighbours. Trickle's key contribution is the use of suppression and dynamic adjustment of the broadcast rate to limit transmissions among neighbouring nodes. While Trickle addresses single packet dissemination, Deluge extends it to support large data objects. However, according to [4], Deluge [1] is unable to provide reliable network programming in a field deployment.

3. CASCADES PROTOCOL DESIGN

Sink-initiated data dissemination usually carries crucial command & control information or binary image code. Meanwhile packet loss could happen due to poor link quality or transmission collision. Hence, it is necessary to ensure 100% reliable delivery and the dissemination process should terminate in acceptable time period. Otherwise, from a user viewpoint, it is a system failure or even worse. For instance, in a feedback-driven volcano monitoring sensor network [3], the connected science model may generate control commands to increase sampling rate of sensor nodes in real-time when some monitored area becomes more active. Those feedback commands have to be received by all expected sensor nodes in the right order with reasonable delay bound. Motivated by those needs from real systems, we designed the Cascades protocol for reliable and fast data dissemination.

We assume that (1) the network traffic and link quality is dynamic (though nodes are not mobile). (2) communication links could be asymmetric.The MAC protocol can be any existing MAC protocol as long as it supports broadcast. We assume a data gathering routing protocol is available in the system (such as MultihopLQI and Drain in TinyOS) and there are continuous data flow from sensors to the sink. Each node has the knowledge of its parent, and can learn its direct children by tracking recent forwarded(upstream) packets. It is possible that children set may be outdated if children switch parents without notifications. As is shown in appendix and theorem 1, Cascades has considered problems caused by dynamic topology. Cascades uses a topology management component to detect and manage topology changes.

Cascades is designed for environment monitoring sensor networks to achieve following goals:

1. to ensure that all data segments are delivered to all the intended receivers with 100% reliability;
2. to ensure that a broadcast session terminates in a reasonable short time period, instead of gossiping forever [2];
3. to minimize the transmission cost for reliable packet delivery and the signalling overhead for missing packets detection and recovery;
4. to operate correctly even in an environment where the radio link quality is very poor;
5. to operate reliably in practical system applications, where nodes and java client software could die and restart, and there may be multiple clients that inject data into the network.

The *Cascades* protocol ensures 100% reliability by implementing parent-monitor-children analogy, and reactive fetch mechanism [16] if there is gap in packet sequence, which denotes missing packets. The parent-children relationship is extracted from the data collection routing tree on the fly. We also apply a update mechanism to update the relationship and thus make *Cascades* resilient and responsive to dynamic topology. When a sensor node forwards a packet for a child, a counter for that child is set to a maximum value δ. When the update timer fires, the counter is decreased by one. If δ reaches 0, it removes the child from the children set. There is a tradeoff between the responsiveness to topology change and the protocol robustness. With a smaller δ, topology change can be detected with a shorter latency. However, sometimes a sensor node may fail to receive packets successfully from a child for certain period due to network congestion. It is observed from our test bed that sometimes some part of the network is "frozen" for up to a few seconds because of high contention. That is to say, no packets from the nodes in that part of the network arrive at the sink during that period. We choose $\delta = 6$ seconds based on our experimental test.

The essential algorithm works as follows (see detailed pseudo-code in the appendix):

1. For a non-leaf node u, it will send an ACK back to its parent node, once it detects all intended children have rebroadcasted the message (implicit ACK) or confirmed with an explicit ACK; otherwise it retries periodically. If u is a leaf node, it sends back ACK once a new message is received or snooped.
2. If a node detects a gap in its received data sequence, it will broadcast a request (e.g., NACK) message to notify its parent.

Each node rebroadcasts periodically until successful reception is confirmed by all intended children either through rebroadcast or explicit ACK. Observant reader might wonder what if there are asymmetric links: parent can not reach children or children can not reach parent. Notice that, if this happens permanently, a correct data gathering routing protocol shall have updated the data gathering tree due to failed link layer acknowledgement. This is another reason why we extract parent-children relationship from data gathering tree. In our implementation, we have designed a TopologyMgmt module which connects to routing module and learns the knowledge of the parent and the children set. In addition, the TopologyMgmt module snoops packets in neighbourhood, discovers and maintains neighbour connectivity, hence it can detect a node's joining or leaving in short time period and notify the routing and Cascades module. When Cascades module is informed that a child node joins/leaves, it updates C_u. If a node changes its parent, Cascades sends out a message to notify the old parent.

On the other hand, Cascades protocol is a fast opportunistic data dissemination protocol, which is as fast as flooding. The broadcast flow does not completely depend on the data gathering tree structure. A snooped new message from neighbor nodes will be accepted and rebroadcasted if necessary. Therefore, it is possible that a parent first hears new data from its child, before it hears it from its parent. It speeds up the dissemination process. For example, in Figure 1, node a is the parent of node b and f. However, node a's data reaches node b only, then b rebroadcasts the data to its neighbors (also as an implicit ACK to node a). At the same time, node h receives data from node l (though l is not node h's parent), and rebroadcasts the data to its neighbors which includes par-

ent node f. Additionally, due to the opportunistic dissemination protocol, a non-leaf node may suppress its own re-broadcasting as long as all of its children have ACKed explicitly or implicitly to reduce communication cost. For example, in Figure 1, assume node a's DATA does not reach f, but node h's ACK and node g's DATA have reached f, then f sends an ACK to a without further re-broadcast. It is not difficult to imagine that: the opportunistic design makes the protocol not just *faster*, but also more *robust* as a node does not only rely on its parent node to get disseminated data. If the link reliability between a parent and child is temporarily not good, it still has a good chance of getting the data from some other neighbor nodes.

We described the algorithm details with pseudo-code in appendix for readers who are interested in implementing and comparing. Some efficient approaches were utilized to speed up dissemination and reduce communication cost. For example, when node u receives a data message from node x, it adds node x to u's acked children set, even if x is not in u's children set. It will benefit when node x may become a child of node u later, in which case adding x to u's acked children set in advance reduces some communication cost and increases system robustness. Another example is: no matter whether u is the parent of node x or not, node x will send ACK message to u as long as it is in the unacked children set of u. This ACK message, which indicates the current parent of x has changed, will prevent node u from retrying forever.

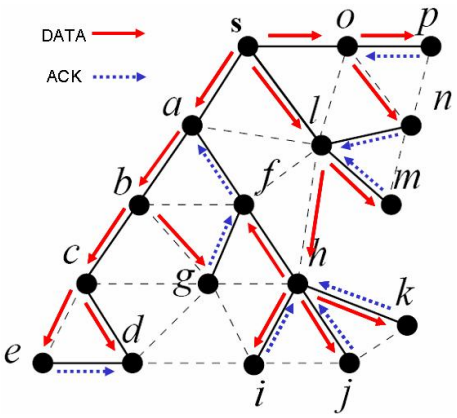

Figure 1. Illustration of opportunistic broadcast flow in a tree with Cascades protocol. The solid lines sketch the tree structure, and the dashed lines shows the other network links.

Theorem 1: Cascades protocol provides 100% reliable data dissemination service from the sink to the entire network.

Proof: The *Cascades* protocol achieves 100% reliability by implementing the parent-monitor-children analogy. In the data gathering tree, each node monitors whether or not its children receive the disseminated data. Each child node notifies its parent of successful reception through implicit or explicit ACKs:

1. Implicit ACK. Parent node snoops packets from children. If a child rebroadcasts the same disseminated packet, it implicitly means that the child has received the packet.
2. Explicit ACK. If the parent failed to snoop the implicit ACK, naturally, it will rebroadcast the data after the timeout period. Then, the child must reply a short explicit ACK message (if the child has previously received same data but it is still in the parent's un-acked children set).

As long as any child fails to ACK implicitly or explicitly, the parent will rebroadcast periodically until every child has confirmed the reception. Hence, reliability is guaranteed as long as the network is consistent.

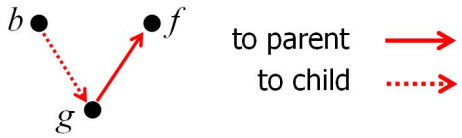

Figure 2. Illustration of possible inconsistent parent-child relationship during route update period.

However, it is possible that, a node switches parents during a data dissemination session. Since each node maintains its own parent and children set distributively, a pair of nodes may have inconsistent information. For instance, as shown in Figure 2, node b takes g as its child, but node g has already switched parents from node b to node f. The possible inconsistencies can be classified as one or both of the following:

1. Case 1: Node g takes node f as its parent, but node f does not take node g as its child.
2. Case 2: Node b takes node g as its child, but node g does not take node b as its parent.

Cascades overcomes these inconsistencies by having each node send a $REQUEST$ message to its new parent for missed data (if it exists) each time it switches parents. Thus, the inconsistencies of Case 1 are mitigated since each child explicitly asks for and is provided with missed data (if any exists) from its new parent. Also, node f adds node g to its children set at this point.

For Case 2, the only negative effect is that node b will retry even though node g is not its child. In algorithm 1, node b will terminate the retry after receiving $ACK(seq, f)$ from node g, as long as it happens. Node b then realizes that node g has switched parents to node f, and removes node g from its children set C_b. Notice that, as mentioned before, those temporary parent-children inconsistency will also be solved by the TopologyMgmt after a predefined time period. Hence, even if there are temporary asymmetric links (e.g., child cannot hear parent or parent cannot hear child), they will not retry forever.

In summary, *Cascades* protocol can handle those inconsistency robustly and provide 100% reliability. ■

In Cascades, we have not discussed how to choose appropri-

ate time interval to accelerate the dissemination speed and minimize communication cost at the same time without sacrifice of reliability. In Algorithm 1, dt_{wait} is a parent node's wait time interval for its children implicit or explicit acknowledgements before broadcasting the message again. If dt_{wait} is too large, the data dissemination rate will be slow. On the other hand, if dt_{wait} is too small, it will cause significant broadcast congestion, waste bandwidth and degrade the dissemination rate due to collisions. Our solution is to let dt_{wait} be adaptive to the network situations. We redefine the dependency in [12] by time-dependency to help decide the value of dt_{wait}.

If node x receives a new packet i from some non-parent nodes before receiving the packet i from its parent, node x will decrease its dependency on its parent by value α. After receiving packet i from its parent, node x will start a timer with interval t. Node x will also decrease its dependency on its parent by value β if it receives packet i from some non-parent nodes within t time period. When timer fires, node x rebroadcast the packet i and keep dependency value unchanged until next new packet's arrival. In our implementation, α is 10 and β is 5, the maximum and minimum dependency value is 60 and 0. The parent node decides dt_{wait} based on the dependency value from its children : dt_{wait} = 100 - (highest dependency value of children) in millisecond unit.

Practical Issues Considerations

It is possible that there are multiple user clients on Internet that are authorized to send control commands to the sensor network. In order to solve such a race condition and guarantee the consistency of command sequence number increment, the command sequence number should be set just before commands are injected into the network. In our implementation, the problem has been solved by an add-on in the SerialForwarder program in TinyOS. SerialForwarder forwards network packets to the Ethernet and distributes commands for multiple clients to the sensor network. Hence, we maintain the sequence number of Cascades message in SerialForwarder, instead of clients, and keep command and control message's sequence number increase consistently.

In a real sensor network deployment, the sensor node and java client software can die and restart by watchdog or user. Like TCP protocol, to guarantee reliability, Cascades has to deal with connectivity management and sequence number synchronization issues. However, unlike TCP which is for one-to-one reliable connection, Cascades is designed for one-to-many communication. The connectivity management for sequence number synchronization in one-to-many communication is expensive and difficult, if not impossible.

In our volcano monitor sensor network system, Cascades component resets its sequence number to 0 if it is idle for 10 minutes. With this mechanism in place, even if the nodes lost synchronization, they can re-synchronize the sequence number in 10 minutes. Then, if a node with sequence number 0 receives a command with larger sequence number, they will adjust its sequence number to this larger sequence number and synchronize with the SerialForwarder.

4. PERFORMANCE EVALUATION

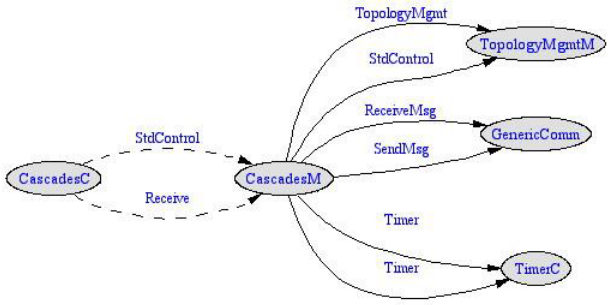

Figure 3. Cascades implementation in TinyOS.

We conducted simulation in network simulator (TOSSIM [18]) to evaluate its efficiency and scalability, as well as experiments in a real sensor network testbed to evaluate its robustness and reliability.

We evaluated the performance by running the Surge application in TinyOS. Surge is a multi-hop data collection application, based on a simple distance-vector routing protocol MultiHopLQI. Each node periodically generates data packets and delivers them to a gateway, hence the data flow forms a tree-like topology rooted at the gateway. TopologyMgmt module maintains the neighborhood connectivity and parent-children information by leveraging MultiHopLQI. As is illustrated in Figure 3, CascadesM is the main module for Cascades: all broadcast data packets and control packets are processed in this module; GeneriComm is link layer component that connects CascadesM to the default CSMA-based MAC protocol.

In our simulation, the Surge application generated and transmitted one data packet every second. This simulated the real situation that upstream data packets will compete bandwidth with downstream broadcasting packets. The sink node injected one packet into the network every τ time interval. We compared the performance of Cascades to Drip. Drip, based on Trickle [2], is a transport-layer component in TinyOS for epidemically disseminating messages throughout the network. After sending a message, Drip will continuously retransmit to ensure that it eventually reaches all expected node in the network. In Drip a node delays its own transmission after receiving duplicate packets from neighbouring nodes.

The following metrics are used to evaluate the performance of Cascades Protocol:

1. Reliability, which measures the number of received packets to the number of sink-injected packets in different time checking points. Time checking point is the time when the number of received packets is examined. r_i is the number of

received packets in node i. At each time checking point, the reliability is defined as $\frac{\sum_{i=1}^{N-1} r_i}{(N-1)\cdot M}$. N is the number of nodes in the network and M is the number of injected packets. When measuring the reliability we do not count in the sink node, so the desired total received packets number is $(N-1)\cdot M$.
2. Hop Delay, which measures the average time elapsed for each packet from the injection in the sink node until the reception in each node in different hops. $T_r[i][j]$ is the reception time in node i for packet j, and $T_s[j]$ is the injection time in sink node for packet j. The average latency $T_d[i]$ in node i is $\frac{\sum_{j=1}^{M} T_r[i][j]-T_s[j]}{M}$. If a packet is lost, the delay for this packet is set to τ, which is the packet injection interval. So the latency in hop m is measured by $H_d[m] = \frac{\sum T_d[i]}{number\ of\ nodes\ in\ hop\ m}$.
3. Communication Cost, the total number of broadcast transmissions used for reliable dissemination in the network, which is $\sum_{i=0}^{N} s_i$. N is the number of nodes in the network and s_i is the number of broadcast transmissions in node i. Communication cost also denotes the energy consumption in the network.

Simulation result on TOSSIM

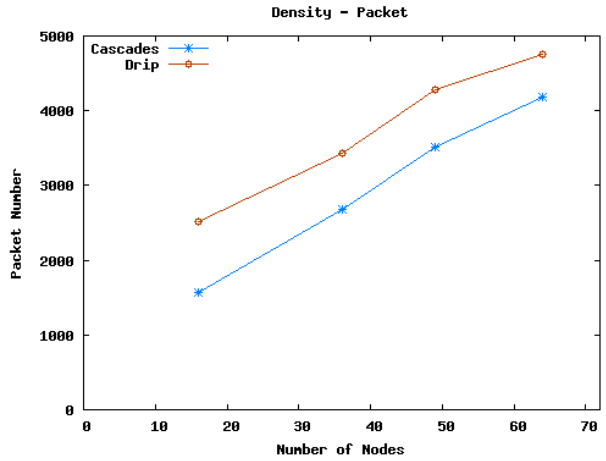

Figure 4. Communication cost in different network densities. X axis is the number of nodes and Y axis is the number of all broadcast data packets. The number of nodes changes from 16 to 64, sink injected packets number $M = 100$, $\tau = 2000$ ms.

Communication Cost—We measured the communication cost with varying network density by increasing the number of nodes in a space of dimensions 450 m $\times$ 450 m. From Figure 4, we can see that Cascades achieves lower communication cost in both sparse and dense networks: In low density network (nodes number is 16), Drip sent 2516 packets to guarantee 100% reliability while it only cost Cascades 1573 packets, 62.5% of the cost of Drip. When nodes number increases to 64, the communication cost is 4178 for Cascades and 4753 for Drip to achieve 100% reliability.

Reliability—In the simulation, the reliability of Cascades almost aligns with that of Drip when injected interval τ is 2000ms and the two protocols can converge soon after sink finished injecting. However, Drip cannot guarantee the reliability with short injected interval($\tau = 400$ms), as is shown in Figure 5. However, with such short injected interval, Cascades still can achieve 100% reliability while we need to extend the injected interval τ to 550ms to make Drip achieve the same result. Two main reasons may lead to this result: 1)In Cascades, we allocate four message buffer entries for caching. The message cache in Cascades helps to recover message sequence gap. 2) Collision reduction schemes in Cascades guarantee a higher probability of message reception.

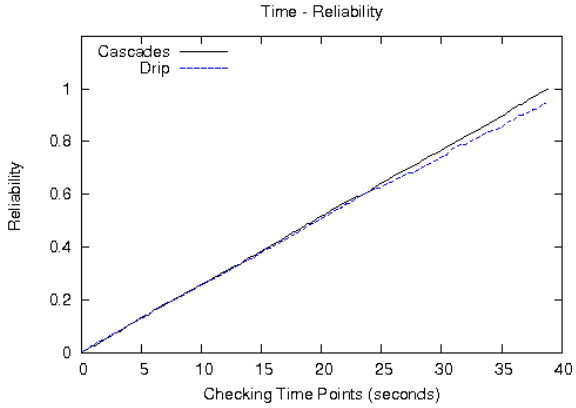

Figure 5. Reliability in different checking time points. X axis is checking time points in second unit and Y axis is reliability. Number of nodes is 64; sink injected packets number $M = 100, \tau = 400\ ms$.

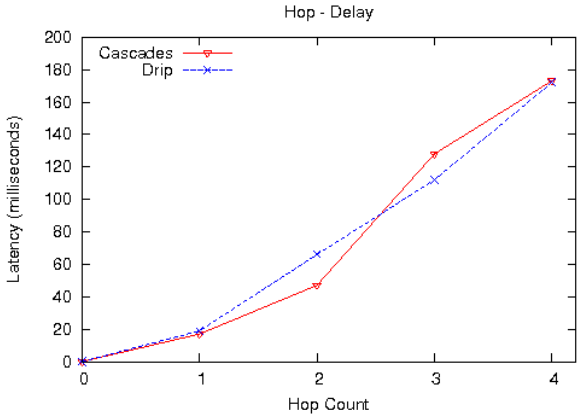

Figure 6. Latency in different hops. X axis is hop count and Y axis is time delay in millisecond unit. Number of nodes is 16; the number of injected packets from sink $M = 100$; $\tau = 2000\ ms$.

Hop Latency—Figure 6 and Figure 7 show the hop latency in dense network (16 nodes) and sparse network (64 nodes) respectively. In sparse network the hop latency of Cascades and Drip is almost the same, but the communication cost of

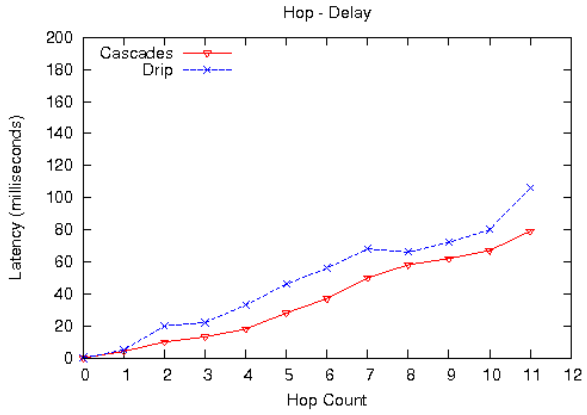

Figure 7. Latency in different hops. X axis is hop count and Y axis is delay time in millisecond unit. Number of nodes is 64, sink injected packets number $M = 100$, $\tau = 2000\ ms$.

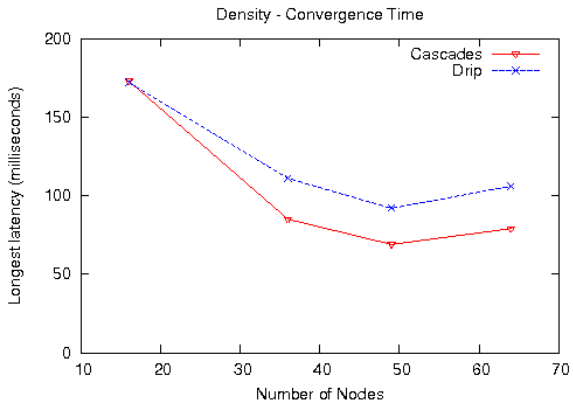

Figure 8. Last hop latency changing with density. X axis is the number of nodes and Y axis is latency in millisecond unit. Number of nodes is 64; the number of injected packets from sink $M = 100$; $\tau = 2000\ ms$.

Cascades is much lower than Drip, as is illustrated in Figure 4. With the increment of network density, the hop latency of Cascades falls below that of Drip. The last hop latency decides the convergence speed of the broadcasting protocol. In Figure 8, we present the average last hop latency of Drip and Cascades. Similar to above discussion, the last hop latency of Cascades was less than that of Drip as we increased the network density. Though the adaptive dt_{wait} and changing of $DTC[seq]$ in our implementation discussed previously help to reduce the communication cost and speed up the dissemination, we still leave this as an open problem to investigate how to choose those parameters that affect dt_{wait}.

Testbed Experiment Results

We did empirical experiments on an indoor sensor network testbed of 15 Micaz motes. The CC2420 radio power of Micaz was set to power level 2 to form a multi-hop network. In the testbed experiment we also compared the performance of Cascades to Bcast which is a blind-flooding protocol: all the nodes in the network will rebroadcast after receiving a packet with a sequence number different from that of the last received packet. There is no retry timer applied in Bcast. Our testbed experiment shows that Cascades performs better than both Drip(gossip based broadcast protocol) and Bcast (blind flooding broadcast protocol) in all three aspects: reliability, communication cost, and delivery delay.

Reliability—From Figure 9 we can see that the average reliability of Cascades is 100% while that of Drip is only 95.1%. Noticed that, in TOSSIM simulation, both Cascades and Drip can achieve reliability with 2000 ms injected interval. The performance of Drip and Cascades is similar in most of time(the slopes are the same), but after losing several packets, the reliability of Drip falls a little. Such packet lost could be explained as: in some cases, Drip cannot guarantee all nodes in the network to receive sink-injected packet within τ period, which is 2000 ms in our experiment. Though such event does not always happen, from user's perspective,

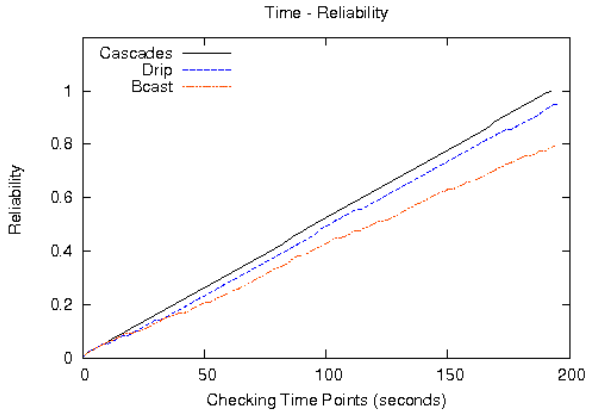

Figure 9. Reliability in different checking time points - testbed result. X axis is checking time points in second unit, Y axis is reliability. Number of nodes is 15, sink injected packets number M=100, $\tau = 2000\ ms$.

it could be viewed as a temporary system failure. Bcast only achieves 79.4% reliability.

Communication Cost—In the experiment, it only costs Cascades 810 packets to converge while 1917 packets for Drip. For each node, the average number of broadcast packets is 54 for Cascades, 72.4 for Bcast and 127.8 for Drip. That means although Cascades uses retry timer for loss recovery, its communication cost is still lower than Bcast which is lack of retry. This is because Cascades exploits the benefits of opportunistic broadcast. Figure 10 illustrates the communication cost with varying time.

5. CONCLUSION

In this paper, we propose a reliable dissemination routing protocol, called *Cascades*, to disseminate data from the sink

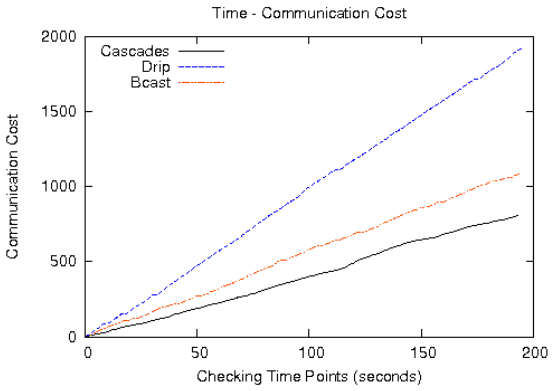

Figure 10. Communication cost in different checking time points - testbed result. X axis is checking time points in second unit and Y axis is the number of broadcast data packets. Number of nodes is 15, sink injected packets number M=100, τ =2000 ms.

to all or a subset of the nodes in a short time period. *Cascades* utilizes the parent-children relationship to ensure truly 100% reliable dissemination. Each (parent) node monitors whether or not its children have received the broadcast messages through implicit listening (e.g., snoop child's broadcast) or explicit ACKs. Hence, it guarantees that a disseminated message from the sink will reach all of the intended receivers in the network. On the other hand, it is also an opportunistic broadcast protocol and the dissemination flow does not depend on the data gathering tree structure, and the path is not deterministic. Opportunistic broacast speeds up the dissemination process and increases the system scalability and robustness.

APPENDIX

We first describe the notations of data structures, messages and functions in algorithm 1.

1. **Data Structures**

(a) C_u denotes the set of children nodes of node u in the sink tree. It will be updated by the TopologyMgmt module if the connectivity changes.

(b) $C_u[seq]$ denotes the subset of children of node u, who have explicitly or implicitly acked the $DATA(seq, *)$. Initially, $C_u[seq] = \phi$. Notice that $C_u[seq]$ may not necessarily be the subset of C_u during the topology update period.

(c) $C_u^-[seq]$ denotes the set of children whose subtree contains the intended receiver but have not acknowledged the $DATA(seq, *)$. In broadcast routing, $C_u^-[seq] = C_u - C_u[seq]$, i.e., the set of nodes in C_u, but not in $C_u[seq]$.

(d) dt_{wait} is the wait time interval for all children's implicit or explicit acknowledgements.

(e) rt_{wait} is the time interval of waiting for the new parent's transmission of missed messages during topology update.

(f) $expectingSeq$ is the next-in-order sequence of data that u is expecting. Initially, it is 1.

(g) $highestSeq$ is the highest sequence of data that u has received. Notice that $highestSeq$ may not equal to $expectingSeq - 1$ if there is a gap in data. Initially, it is 0.

(h) $DT[seq]$ is the data rebroadcast timer of $DATA(seq, *)$.

(i) RT is the data request timer to ask the new parent to send missed data (if any), after parent changes.

(j) $hasData[seq]$ is TRUE if $DATA(seq, *)$ has been received, otherwise FALSE.

2. **Messages**

(a) $DATA(seq, *)$ is a data message with sequence number seq.

(b) $ACK(seq, x)$ is the acknowledgement message of $DATA(seq, *)$ to x from its children or some other nodes (happens only when topology is inconsistent).

(c) $REQUEST(seq, parent)$ is the request message of DATA with sequence number seq to the $parent$.

(d) $NODATA(seq)$ is a message to notify children that messages with sequence number equal or larger than seq have not been received.

3. **Functions**

(a) $isActive(timer)$ is TRUE if the timer is active, otherwise FALSE.

(b) $send(msg, x)$: send msg to node x.

(c) $broadcast(msg)$: broadcast msg.

Algorithm 1 Cascades Dissemination Protocol

Each node u runs the following event-driven procedure:
switch(**event**) {

1. **message** $DATA(seq, C_x^-[seq])$ **received from node** x**:**
/* add into Cu */
$C_u[seq] = C_u[seq] \bigcup x$;
/* Packet with sequence #seq has been ACKed by all children */
if ($C_u^-[seq] == \phi$ && isActive($DT[seq]$)){
  stop timer $DT[seq]$;
}
if (!hasData[seq]) {
  buffer it and copy it to the up layer for process;
  hasData[seq] = TRUE;
  if($seq > highestSeq$) $highestSeq = seq$;
  if($seq == expectingSeq$) {
    while(hasData[expectingSeq]) expectingSeq ++;
    if($expectingSeq < highestSeq$ && !isActive(RT))
      start repeat timer $RT = rt_{wait}$;
  }
  /* Not receiving ACK from each child */
  if($C_u^-[seq] \neq \phi$){
    broadcast($DATA(seq, C_u^-[seq])$);
    start repeat timer $DT[seq] = dt_{wait}$;
  } else {
    send($ACK(seq, parent)$, $parent$);
  }
} else {
  /* receive old message */

if ($u \in C_x^-[seq]$) {
/* my parent does not receive ack from me */
send($ACK(seq, parent)$, x);
}
}
break;

2. **message** $ACK(seq, x)$ **received from node** y**:**
$C_u[seq] = C_u[seq] \bigcup y$;
if ($C_u^-[seq] == \phi$ && isActive($DT[seq]$)){
stop timer $DT[seq]$;
}
/* The parent address in ACK message is not me, delete it from C_u */
if($x \neq u$) $C_u = C_u \backslash y$;
break;

3. **timer** $DT[seq]$ **fired:**
if ($C_u^-[seq] == \phi$){
stop timer $DT[seq]$;
} else {
broadcast($DATA(seq, C_u^-[seq])$);
}
break;

4. **parent switched to node** x**:**
$parent = x$;
if(!isActive(RT)) {
send($REQUEST(expectingSeq, parent), parent$);
start repeat timer $RT = rt_{wait}$;
}
break;

5. **timer** RT **fired:**
send($REQUEST(expectingSeq, parent), parent$);
break;

6. **message** $REQUEST(seq, x)$ **received from node** y**:**
if($x \neq u$) break;
$C_u = C_u \bigcup y$;
if(hasData[seq]) {
broadcast($DATA(seq, C_u^-[seq])$; }
else {
broadcast($NODATA(expectingSeq)$);
}
break;

7. **message** $NODATA(seq)$ **received from node** x**:** if($x \neq parent$) break;
if(!isActive(RT)) break;
if($seq \leq expectingSeq$) {
stop timer RT;
} else {
$expectingSeq = seq$;
stop timer RT;
}
break;
}

ACKNOWLEDGMENTS

The research described in this paper was supported by NASA ESTO AIST program and USGS Volcano Hazard program under the research grant NNX06AE42G. Many thanks to Prof. Rong Zheng from University of Houston for their insightful comments on this paper.

REFERENCES

[1] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[2] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[3] "OASIS: Optimized Autonomous Space In-Situ Sensor Web. http://sensorweb.vancouver.wsu.edu."

[4] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[5] K. Viswanath and K. Obraczka, "Modeling the performance of flooding in multihop ad hoc networks," in *Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2004.

[6] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Cutler, "Exploiting the capture effect for collision detection and recovery." in *The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II).*, 2005.

[7] A. Ephremides and T. V. Truong, "Scheduling broadcasts in multihop radio networks," vol. 38, no. 4, pp. 456–460, April 1990.

[8] R. Zheng, L. Sha, and W. Feng, *MAC layer support for group communication in wireless sensor networks*, 2005.

[9] B. Williams and T. Camp, "Comparason of broadcasting techniques for mobile ad hoc networks," in *ACM MobiHoc*, 2002.

[10] F. Stann, J. Heidemann, R. Shroff, and M. Z. Murtaza, "Rbp: robust broadcast propagation in wireless networks," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2006, pp. 85–98.

[11] M. Agarwal, L. Gao, J. H. Cho, and J. Wu, "Energy efficient broadcast in wireless ad hoc networks

with hitch-hiking," *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 10, no. 6, pp. 897–910, 2004.

[12] P. Kyasanur, R. R. Choudhury, and I. Gupta, "Smart gossip: An adaptive gossip-based broadcasting service for sensor networks," in *The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2006, pp. 91–100.

[13] F. Stann and J. Heidemann, "Bard: Bayesian-assisted resource discovery in sensor networks," USC/Information Sciences Institute, Tech. Rep., July 2004.

[14] Q. Huang, C. Lu, and G.-C. Roman, "Spatiotemporal multicast in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 205–217.

[15] R. Zheng, "Information dissemination in power-constrained wireless networks," in *INFOCOM, 2006: Proceedings of the 25th IEEE International Conference on Computer Communications*, 2006, pp. 1–10.

[16] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Psfq: a reliable transport protocol for wireless sensor networks," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM Press, 2002, pp. 1–11.

[17] S.-J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz, "Garuda: Achieving effective reliability for downstream communication in wireless sensor networks," *IEEE Transactions on Mobile Computing*, June 2007.

[18] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys).*, 2003.

BIOGRAPHY

Yang Peng received his master degree in computer science from Washington State University - Vancouver in July 2008. Before that he was a research assistant of sensorweb research laboratory. His research interest includes reliable data dissemination and smart sensing on wireless sensor networks. His master thesis is "Smart Sensing Design for Environment Monitoring Sensor Networks". He received BS degree from Beijing University of Posts and Telecommunications.

Wenzhan Song is an assistant professor in computer science from Washington State University Vancouver. He is the Principal Investigator of Optimized Autonomous Space In-Situ Sensorweb project, which is supported by NASA ESTO AIST program and a multidisciplinary team involving earth scientists, computer scientists and space scientists. His research interest spans sensor networks, peer-to-peer networks, distributed systems and algorithms. He has published extensively in these areas, including number of journal articles, conference papers and book chapters, and serves many conferences and journals. He received PhD from Illinois Institute of Technology, MS and BS degree from Nanjing University of Science & Technology. He is a member of the AGU, IEEE and ACM.

Renjie Huang is an PhD student in computer science from Washington State University. His research interest mainly focuses communication scheduling and optimization in wireless sensor network. He received his B.S. degree from South China University of Science & Technology and received his Master degree from Huazhong University of Science & Technology.

Mingsen Xu is a PhD student in computer science from Washington State University - Vancouver. He currently is a research assistant of sensorweb research laboratory. His research interest focuses on data correlation and reduction on sensor networks. He received his MS and BS degree from Beijing University of Posts and Telecommunications.

Behrooz A. Shirazi s the Huie-Rogers Chair Professor and Director of the School of Electrical Engineering and Computer Science. His research interests are in the areas of pervasive computing, software tools, distributed real-time systems, scheduling and load balancing, and parallel and distributed systems. He has received grant support totaling more than $8 million dollars from federal agencies, including NSF, DARPA, and AFOSR, and private sources, including Texas Instruments and Mercury Computer Systems. He has received numerous teaching and research awards. He is currently the Editor-in-Chief for Special Issues for Pervasive and Mobile Computing Journal and has served on the editorial boards of the IEEE's Transactions on Computers and Journal of Parallel and Distributed Computing.