UFlood: High-throughput flooding over wireless mesh networks

UFlood: High-Throughput Flooding over Wireless Mesh Networks

Jayashree Subramanian, Robert Morris, and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Lab
{jaya,rtm,hari}@mit.edu

***Abstract*—This paper proposes UFlood, a flooding protocol for wireless mesh networks. UFlood targets situations such as software updates where all nodes need to receive the same large file of data, and where limited radio range requires forwarding. UFlood's goals are high throughput and low airtime, defined respectively as rate of completion of a flood to the slowest receiving node and total time spent transmitting. The key to achieving these goals is good choice of sender for each transmission opportunity. The best choice evolves as a flood proceeds in ways that are difficult to predict.**

UFlood's core new idea is a distributed heuristic to dynamically choose the senders likely to lead to all nodes receiving the flooded data in the least time. The mechanism takes into account which data nearby receivers already have as well as inter-node channel quality. The mechanism includes a novel bit-rate selection algorithm that trades off the speed of high bit-rates against the larger number of nodes likely to receive low bit-rates. Unusually, UFlood uses both random network coding to increase the usefulness of each transmission and detailed feedback about what data each receiver already has; the feedback is critical in deciding which node's coded transmission will have the most benefit to receivers. The required feedback is potentially voluminous, but UFlood includes novel techniques to reduce its cost.

The paper presents an evaluation on a 25-node 802.11 test-bed. UFlood achieves 150% higher throughput than MORE, a high-throughput flooding protocol, using 65% less airtime. UFlood uses 54% less airtime than MNP, an existing efficient protocol, and achieves 300% higher throughput.

I. Introduction

Flooding in wireless mesh networks is a classic problem. Potential uses include software update in large networks, e.g., sensor nets [7, 11], and distribution of information such as entertainment and surveillance video [3, 6]. Flooding is also interesting and challenging: its many degrees of freedom along with interaction with the physical layer have led to much research [4, 18, 20, 23].

This paper targets situations in which a source node has a large amount of data to transfer in its entirety to a set of participating nodes. The nodes are equipped with broadcast radios, and the network is assumed to be spread out enough to require multi-hop forwarding. The targeted performance goals are high throughput (transfer size divided by the time until all nodes have the data) and low airtime (total time spent transmitting, a measure of how efficiently the radio channel is used). Low airtime is important to reduce flooding's impact on other uses of the wireless network, as well as to help achieve high throughput.

A flooding protocol must repeatedly make three big decisions: which nodes should transmit, what data they should transmit, and what physical-layer bit-rates they should use. The answers depend on the radio channel quality between nodes, the number of receivers near each potential sender, and what data potential receivers already hold. The set of best senders changes as nodes accumulate data, in ways that are hard to predict because receptions are not deterministic. Section II explains these challenges in more detail.

This paper describes a new flooding protocol, UFlood, that achieves high throughput and low airtime by careful selection of senders and bit-rates. The core of UFlood's design is its *utility* heuristic, which predicts how much benefit receivers would derive from a given node transmitting. UFlood includes a novel bit-rate selection algorithm that trades off the speed of high bit-rates against the larger number of nodes likely to receive low bit-rates; the algorithm recognizes that bit-rates cannot be selected with purely local information. Finally, UFlood uses a novel technique in which a sender simulates receptions at receivers in order to reduce the required rate of feedback from receivers to potential senders.

We have implemented UFlood using the Click software router toolkit [17]. Experiments on a 25-node 802.11 test-bed show that UFlood achieves 150% higher throughput than MORE [4], a high-throughput flooding protocol, using 65% less airtime. UFlood uses 54% less airtime than MNP [18], a high-efficiency protocol, and achieves 300% higher throughput. UFlood's careful choice of sender via its utility heuristic and its smart feedback mechanism are responsible for its high performance.

This paper makes the following contributions. First, it describes a novel sender selection algorithm for flooding that quickly spreads useful data. Second, it describes the main underlying properties of wireless and of flooding that drive sender selection. Third, the paper proposes the first bit-rate selection scheme for a flooding protocol. Fourth, the paper demonstrates that detailed feedback about the data each receiver possesses is useful even with coding. Finally, the paper describes novel techniques to to reduce feedback overhead.

II. Sender and Bit-rate Selection

This section explains why careful selection of sender and bit-rate for each transmission are important to good flooding performance, and outlines the main factors affecting these decisions. The identification of these factors is one contribution of this paper; previous flooding protocols have incorporated sender selection, but none have considered the full set of factors that we have found to be important.

A. Delivery Probabilities

Figure 1 shows an example in which one sender is more effective than another due to delivery probabilities. Nodes A

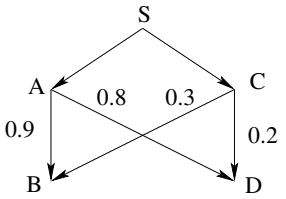

Fig. 1: Illustration of the importance of link-layer packet delivery probabilities, which are indicated by the numbers.

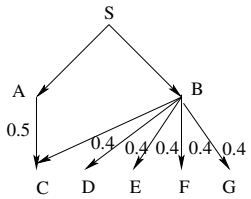

Fig. 2: Illustration of the need to consider the number of receivers, including those with low probability.

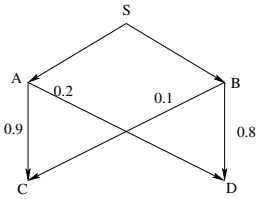

Fig. 3: Illustration of the best sender changing as nodes receive packets.

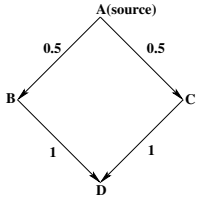

Fig. 4: Illustration of the effect of correlated packet reception.

and C have each received a particular data packet from S. The link-layer broadcast packet delivery probabilities from A and C to each of B and D are indicated by the numbers in the figure. UFlood must decide whether it is better for A or for C to transmit the packet.

If A transmits, the expected number of useful receptions (summed over B and D) is 1.7. If C transmits, the expected number of useful receptions is 0.5. If A transmits first, in all likelihood C will not have to transmit at all, but the converse is unlikely to be true. Thus A is the better sender. This example illustrates why UFlood must pay attention to delivery probabilities when selecting the sender.

B. Number of receivers

Figure 2 shows an example in which a sender with low probabilities to many nodes is a better choice than a sender with fewer high-probability receivers. If A transmits, the expected number of useful receptions is 0.5 (just node C). If B transmits, the expected number is 2.0. B will likely have to repeat the transmission a few times; in all likelihood C will hear one of those transmissions, and A will not have to send at all. The reason is that B has to keep sending until nodes D through G receive the data. If receptions are independent, the probability that C will receive one of the transmissions made by B is over 98%. Thus B is the better sender. This example illustrates why UFlood must incorporate the number of likely receivers in its choice of sender.

C. Receiver state

Figure 3 shows a situation in which the best sender changes as nodes receive packets. A and B have a particular packet, but C and D do not. At that point, A is the best sender. A transmits the packet, and C receives it but D does not. Now B is the best sender: the expected number of useful receptions for A and B are now 0.2 and 0.8, respectively. This example illustrates why UFlood senders must be aware of what information receivers already possess, and must re-evaluate the choice of best sender as a flood progresses.

D. Correlated reception

The usefulness of a node's transmissions depends on whether it has received information that is distinct from other nearby forwarders. Suppose that nodes *B* and *C* in Figure 4 have both received half of the source's transmissions, and that *D* can hear *B* and *C* perfectly but cannot hear the source. At one extreme, *B* and *C* may have received disjoint halves, in which case each of *B* and *C* should forward all the packets they hold. At the other extreme, *B* and *C* may have received exactly the same set of packets. In that case they have the same underlying information to offer, even with coding, so that only one should send. This example illustrates one reason why each UFlood node sends feedback indicating what information it has received.

E. Bit-rate

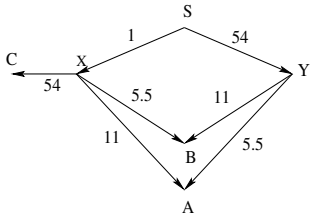

Fig. 5: Illustration of bit-rate selection strategy.

Choice of bit-rate can have a large effect on performance, given the large difference between the slowest and fastest bit-rates in, for example, 802.11b/g radios. The choice is difficult because higher bit-rates will deliver data faster to nearby receivers, but will also increase frame error rates for more distant receivers. Each receiver typically has a highest-throughput bit-rate from a given sender that balances bit-rate with error rate; error rates typically increase dramatically at bit-rates much above that optimum.

Figure 5 illustrates a typical bit-rate selection problem. Each link is marked with its optimum bit-rate. Sender *X* must choose the bit-rate for its next transmission. Bit-rate 54 would maximize total receive rate among *X*'s neighbors: *C* would receive at rate 54, and *B* and *A* would receive very little due to errored frames. *X* might then have to re-send all the data at rate 5.5 for node *B*; since *C* will overhear those packets, that suggests that the right strategy might be for *X* to send at rate 5.5 to begin with in order to satisfy all its neighbors in a single set of transmissions. However, node *B* has a higher bit-rate path from *S* via *Y*. Thus it is better for node *X* to ignore *B*, letting *Y* deliver to *B*, and choose the rate that's best for the slowest neighbor whose best path from *S* is via *X*. That neighbor is *A*, and *X*'s best bit-rate is 11. If node *A* already has the data that node *X* would transmit, *X* should ignore *A* in choosing its bit-rate, and send at 54Mbps.

To summarize, a sender should only reduce its bit-rate to reach receivers that have no faster path from the source via other potential nearby senders.

III. Related Work

Early work on flooding in wireless mesh networks identified avoidance of redundant transmissions as the key challenge [23]. Three main approaches have subsequently been explored: gossip, trees, and network coding.

Gossip. Gossip-based flooding protocols use local interactions to propagate data, without any global structure; Trickle [20] and Deluge [11] are examples. Trickle and Deluge select senders randomly from the set of nodes that have data needed by some neighbor. UFlood uses more information in its sender selection in order to favor senders who will satisfy the most neighbors. MNP [18] selects senders based on the number of neighbors that might benefit from each sender. MNP incorporates one of the considerations used by UFlood's utility (number of receivers), but does not consider delivery probabilities or bit-rate.

Trees. Flooding packets over a tree imposed on the network can help avoid redundant transmissions and help ensure that only certain nodes transmit [14, 22]. UFlood avoids using a tree so that it can exploit opportunistic receptions and adjust to changing conditions, though these could also be done in the context of a tree.

Network coding. Coding allows a flooding protocol to satisfy different gaps in the data at different receivers using a single transmission; random network coding (RNC) allows intermediate nodes to generate coded packets before they have received the entire transfer [1, 8, 10, 21].

The MORE protocol [4] uses RNC in order to eliminate the need for feedback from receivers to senders, except for an indication at the end of a transfer that the receiver has all the data. MORE assigns a rate (called TX_credit) to each node that governs what fraction of received data it forwards. TX_credit is based on delivery probabilities, and its calculation assumes independent reception at different nodes. UFlood, in contrast, makes dynamic decisions about which nodes should send, based on feedback about the current states of receivers. This allows UFlood to adapt to actual reception patterns as a flood progresses, and also allows it to cope with situations in which reception is not independent. This paper demonstrates that UFlood's combination of coding with detailed feedback significantly decreases the amount of transmission required for flooding.

Bit-rate selection Much is known about wireless bit-rate selection for point-to-point links [2, 9, 19] and for WiFi multicasting [16, 24], where all the receivers are within the radio range of the sender. To the best of our knowledge, there are no existing bit-rate selection mechanisms for wireless flooding or multicast protocols for multi-hop mesh networks.

IV. Design

The goal of UFlood is to distribute a large quantity of data from a single source to all nodes in a wireless mesh network. The design relies on the following assumptions.

- Each node has a radio that operates at a fixed power level, on a single channel, with a non-directional antenna.
- Some node pairs can communicate directly, but others must relay through multiple hops.
- Nodes are relatively stationary and are willing to forward data for each other.
- The network size is on the order of dozens of nodes, and there is a path with non-zero delivery probability from the source to every other node.
- The radios have a carrier sense mechanism that avoids collisions.

A. Design Overview

The main elements in UFlood's design are: (i) sender selection, (ii) bit-rate selection, and (iii) efficient handling of feedback to support sender selection. At a high level, UFlood works as follows.

The nodes cooperate to measure and distribute the delivery probability between each node-pair at each possible bit-rate. Each node runs a bit-rate selection algorithm (Section IV-B) to calculate the best bit-rate for itself and for each other node.

A UFlood transfer starts at the source node, which has the data to be flooded. The source considers one *batch* of K *native packets* (uncoded packets) of the original data at a time. The source starts a batch by transmitting K packets, each coded over the K native packets in the batch (Section IV-C). All nodes then go through the following cycle until every node indicates to the source that it has received the entire batch.

Each node calculates its own *utility*, and the utility of its immediate neighbors, roughly once per data packet time; the utilities reflect how valuable it would be for each node to transmit (Section IV-D). Each node that decides its utility is higher than that of all of its neighbors transmits a *burst* of re-coded data packets. Each receiver may broadcast a feedback packet after receiving a burst, describing the coded data it holds (Sections IV-E IV-F IV-G IV-H); the feedback informs utility calculations for the next burst.

This process continues until all nodes signal the source node that they are able to decode the batch, at which time the sender proceeds to the next batch.

B. Bit-rate Selection

A UFlood sender may have many neighbors, each with a different optimum bit-rate from that sender. In choosing a bit-rate, a sender is essentially choosing which neighbors to send data to, since neighbors with optimum rates much below the chosen rate will receive mostly corrupted frames. The best choice depends on whether each low-bit-rate neighbor depends on the sender: if the sender is a neighbor's best source of data, the sender should reduce its bit-rate. For this reason, the core of UFlood's bit-rate selection algorithm is a decision about whether each neighbor depends on the sender as the best path from the source to that neighbor.

UFlood decides if a neighbor depends on a sender by finding the highest throughput unicast route from the source to each node, using the ETT (Expected Transmission Time) metric [5]. If a sender is the last hop in the highest-throughput route from the source to a neighbor, then the neighbor depends on the sender for data. A sender chooses the lowest of the optimum

bit-rates to the neighbors that depend on the sender and which lack data that the sender has. Each node calculates the best bit-rate for every node, including itself.

C. Network Coding

UFlood uses randomized network coding (RNC) over each batch. RNC often allows individual transmissions to be useful at multiple receivers even if those receivers are missing different parts of the batch. Each of the source's transmissions is coded over all the native packets in a batch, forming a "coded packet," as in MORE [4]. If the K native packets are $n_1 \ldots n_K$, and $c_1 \ldots c_K$ are K randomly chosen integers, then a data packet transmission is $p = c_1 n_1 + c_2 n_2 + \ldots + c_K n_K$. The arithmetic is byte-wise, so that the first byte of p is c_1 times the first byte of n_1 plus c_2 times the first byte of n_2 ... plus c_k times first byte of n_k. The arithmetic is carried out in the finite Galois field $GF(2^8)$. Each coded broadcast includes the K coefficients $(c_1 \ldots c_K)$ used to construct p. A packet coded from the native data is called a *first-generation* packet; F_i denotes the *ith* first-generation packet generated by the source node.

A non-source sender broadcasts packets re-coded over all the first generation coded packets it has received in the current batch, using new random coefficients. These packets are non-first-generation packets S_i. All nodes include, in each transmission, coefficients relative to the original native packets. Once a node has received K linearly independent packets in the current batch, it decodes them to obtain the native packets. At that point the node starts to act as a *source-like* node, sending first-generation packets coded from the native data. The reason to send first-generation packets when possible is that they are more likely to be linearly independent of each other than are re-coded packets.

A node sends an acknowledgment packet via reliable unicast routing to the source when it is able to decode a batch. Once the source hears such an acknowledgment from every node in the network, it moves to the next batch. Each node stops sending acknowledgments when it receives a packet from a later batch.

D. Utility

Once the source has sent a full set of K coded packets for a batch, other nodes will be in a position to send further re-coded packets to help spread the flood. UFlood's utility heuristic chooses good senders based on the factors in Section II.

A node's *utility* is the expected total amount of useful data per unit time that would be received if that node transmitted. Node A estimates the utility of node B (possibly itself) as follows:

$$U_A(B) = \sum_{C \in N_B} P_{B,C,b^*(B)} \cdot b^*(B) \cdot I_{B,C} \qquad (1)$$

N_B is the set B's neighbors—nodes with delivery probability greater than 0.1 from B at B's best bit-rate $b^*(B)$. $P_{B,C,b^*(B)}$ is the delivery probability from B to C at bit-rate $b^*(B)$. $I_{B,C}$ is 1 if a coded packet from B would be linearly independent of the packets C already has, and 0 otherwise.

Equation 1 captures the considerations in Section II. Summing over neighbors favors senders with many receivers. Weighting by delivery probabilities favors senders with high-quality links to receivers. Multiplying by transmit bit-rate favors senders with faster links to receivers. The $I_{B,C}$ factor favors transmissions likely to be linearly independent of data already held by receivers.

UFlood's utility is a locally greedy heuristic: it does not account for the possibility that a sender with only a few low-quality links might deliver packets to nodes that would then be able to transmit to many receivers on high-quality links. Equation 1 also assumes that good spatial re-use will result from allowing high-utility nodes that can't hear each other to send concurrently.

E. Feedback Content

UFlood's feedback helps potential senders calculate $I_{B,C}$. The feedback design is particularly challenging:

- Feedback has the potential to consume more bandwidth than the data itself: the full description of a node's state might consist of K coding coefficients for each of up to K packets, or 4096 coefficients for $K = 64$. Thus UFlood uses an abbreviated feedback representation, and spaces out feedback in time (Section IV-G).
- Delayed and lost feedback can cause nodes to have inaccurate views of each others' state. This may cause no node to think it has the highest utility, leading to idle time. It can also cause useless transmission of data already possessed by receivers. Idle time and useless transmissions can drastically reduce performance, so UFlood predicts feedback by "interpolating" between periodic feedback receptions (Section IV-F).

A feedback packet from node B contains:

1) A count of the linearly independent packets B holds, also called the *rank* of B.
2) A bitmap identifying each distinct first-generation packet that contributed (via coding) to any of the packets held by B.
3) The rank of each of B's neighbors.

A typical packet with the above contents has about 80 bytes of payload, far less than would be required for K coefficients for each packet B holds.

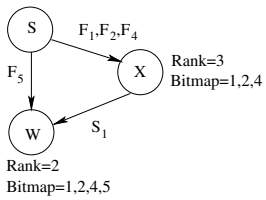

Fig. 6: Illustration of feedback in UFlood. F_i and S_i are first and non-first-generation packets, respectively.

For example, consider Figure 6. Suppose node W has received packet F_5 directly from the source, and has also

received a packet S_1 sent by node X that X generated by re-coding packets F_1, F_2, and F_4 from the source. Then W's feedback will indicate a rank of two, and its bitmap for the source will have entries 1, 2, 4, and 5 set.

This feedback is sufficient to conservatively estimate $I_{B,C}$, without needing to know the actual coefficients, as follows:

$$I_{B,C} = \begin{cases} 1 & \text{if } L_{B,C} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$L_{B,C} = \begin{cases} 0 & \text{if rank}(C) \geq K, \text{ or} \\ b_1 & \text{if rank}(B) > \text{rank}(C) \text{ (\textbf{I1}), or} \\ b_2 & \text{if rank}(B) \leq \text{rank}(C) \text{ and } B \text{ has more bits} \\ & \text{set in its bitmap than } C \text{ (\textbf{I2}), or} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $L_{B,C}$ is the maximum number of coded packets that C can receive from B that would be linearly independent of packets C already has, b_1 is rank(B)−rank(C) and b_2 is the number of bits that are set in the bitmap of B but not set in the bitmap of C. This calculation is a conservative estimate: if $L_{B,C}$ is greater than zero, then a transmission from B is highly likely to benefit C, while if zero, there is still some chance that a transmission would be beneficial.

As an example of condition I1, consider Figure 6. Suppose node W has two packets, and its bitmap has bits set at positions 1,2, 4, and 5. A transmission from node X with a rank three is likely useful at W. The only way this could fail to be true is through an unlucky choice by W of its recoding coefficients.

As an example of I2, consider Figure 6. Suppose node X has three packets, but none is coded over F_5. Then a transmission from W, which has only two packets will be linearly independent of the packets X already has since it is coded over F_5. This is the reason why sending the rank of nodes alone as feedback is not enough information to know whether a sender's transmission is useful to its receiver.

Once a node receives K linearly independent coded packets, the receivers of its transmissions will end up setting many bits in its feedback bitmap, which will make I2 rarely true. For example, suppose node X has received $F_1 ... F_K$. X then transmits twice; Y receives only its first packet and Z receives only its second packet. Now Y and Z have the same rank (one) and the same set of bits set in their feedback bitmaps $(1, 2, \ldots, K)$, so neither condition I1 nor I2 is true. However, they could benefit from each other's transmissions, because they each have at least one linearly independent packet for the other. To address this situation, UFlood nodes that have received enough packets to decode the whole batch, begin to transmit first-generation packets, coded from the native data. Such nodes are called "source-like" nodes. Each feedback packet contains 256 bits for each source-like node from which the feedback sender has received packets. Condition I2 applies to the entire set of bits.

F. Feedback Interpolation

Nodes must often calculate utility using out-of-date feedback, since feedback can be lost and nodes send feedback relatively infrequently. Nodes attempt to correct stale feedback by *interpolating*. For every data transmission that A knows of since B's last feedback, A simulates the effect of that transmission on B's feedback with probability equal to the delivery probability from sender to B. If A simulates that B received the packet, and decides that the packet would have been linearly independent of the packets B's feedback indicates it already has, A increments rank(B), and sets the bits in B's feedback bitmap corresponding to the source's packets that contributed to the data transmission.

G. Feedback Timing

UFlood sends bursts of data packets without feedback to reduce feedback overhead. When a node A decides it has the highest utility, it sends a burst of $\min_{C \in N_A} L_{A,C}$ packets. This is the most packets that A can send without causing some neighbor to have higher utility than A. A receiver sends a feedback packet when it detects an idle channel for three packet durations and it guesses (via interpolation) that at least one of its neighbors could send a packet that would be useful for it.

The overall burst sequence is as follows. The current sender sends a burst of packets. Other nodes calculate the sender's burst length (or observe it in the sender's packet headers) and wait long enough for the burst to have ended. Then all the nodes recalculate utilities, and the best node sends a new burst. This process can proceed for a while without feedback, all nodes using interpolation instead. At some point interpolation will predict that all nodes have enough packets to decode the whole batch. No node will send, nodes that have not in fact received enough packets will observe an idle channel, they will send feedback, and that will cause some other node to become a sender. If all nodes can decode the batch, they will send acknowledgments to the source, which will start a new batch.

H. Idle Time

Loss of feedback packets and unlucky packet-loss simulation in interpolation can cause no node to believe it has the highest utility, and thus idle time. UFlood has two mechanisms that help it avoid idle time. First, feedback packets include neighbor ranks, which increases the likelihood that all nearby nodes will compute utilities using consistent information even if they don't all hear feedback directly from the same set of nodes. Second, sending bursts of packets without needing to wait for feedback reduces the opportunities for idle time.

If idle time does occur, UFlood recovers by having any node that thinks it has the second-highest utility start transmitting if it hears no packet from the best node for three packet times, and by having nodes send feedback packets when they detect an idle channel. Idle time can also occur when most nodes are able to decode a batch, those nodes' interpolation has incorrectly guessed that all other nodes can also decode, and the rules for sending feedback don't trigger feedback from the few nodes that don't have enough packets. UFlood handles this problem by arranging nodes in a tree rooted at the source, and having each parent reset its interpolated state for any children

that do not send an acknowledgment to the source soon after the parent has decoded the whole batch. This causes the parent to become a sender and drive the children towards completion.

I. Hidden Terminals

Two nodes that cannot hear each other might both send and collide at common receivers. UFlood reduces the chances of this in the following way. As described in Section IV-E, feedback packets contain the ranks of neighbors. Thus feedback from common receivers will cause two-hop neighbors, and thus potential hidden terminals, to be aware of each other. When a node is deciding if it has the highest utility, it compares not just against neighbors but also against two-hop neighbors with which it shares receivers that could benefit from both senders. In many cases this suppresses potential hidden terminals.

V. Evaluation

This section evaluates the performance of a UFlood implementation on a 25-node 802.11 test-bed deployed across 3 floors of an office building. We measure two quantities: *throughput* and *airtime*. The throughput is the transfer size divided by the time for all the nodes to receive the whole transfer. Airtime is the sum over the total time each node spends in transmitting packets.

Our experiments compare UFlood with two existing protocols: MORE [4] and MNP [18]. Our main result is that UFlood, on average, achieves 150% higher throughput than MORE, using 65% less airtime. UFlood achieves 300% higher throughput than MNP using 54% less airtime.

A. Experimental Setup

The UFlood, MORE, and MNP implementations use the Click software router toolkit [17], running as a user-space process on Linux. Each node has a 500 MHz AMD Geode LX800 CPU.

The test-bed nodes use 802.11b/g with a transmit power level of 12 mW. The test-bed is large enough that many nodes cannot hear each other. Figure 7 shows the layout of the test-bed. Figure 8 shows the distribution of inter-node delivery probabilities at 5.5 Mbits/s. The graph shows that the test-bed has a wide range of link qualities even at such low bit-rate.

Each run involves the source distributing 2 Mbytes of data. The default batch size is $K = 64$ packets and 32 such batches are flooded. A data packet contains 1024 bytes of coded data plus protocol overhead (e.g., coding coefficients).

Most of the results below report distributions of results over all choices of source node, in order to emulate the effect that different topologies would have. Each point in each distribution represents the average of seven runs with a given source. All graphs include the overhead of UFlood's feedback.

B. Protocols used for Comparison

We compare with MORE because MORE is the highest-throughput existing flooding protocol known to us. We compare with MNP because it is the most efficient existing protocol known to us (it has low airtime).

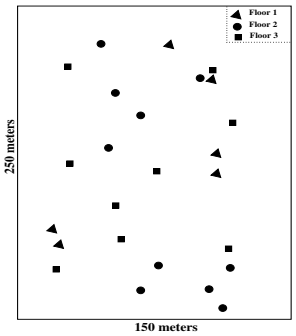

Fig. 7: Physical layout of the test-bed.

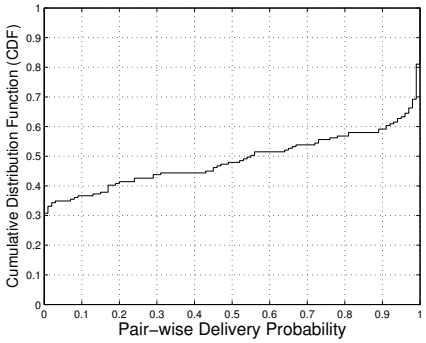

Fig. 8: CDF of pair-wise delivery probabilities.

The MORE software is the multicast implementation used in [4]. We re-implemented MNP as described in [18], and added network coding so that our evaluation could focus on sender selection alone. The MORE and MNP implementations use only the 5.5 Mbps bit-rate; that is the fixed bit-rate that gives them the highest throughput on our test-bed. We also compare with a version of UFlood that operates at just 5.5 Mbps, called UFlood-R.

C. Throughput and airtime

Figure 9 shows the CDF over all sources of the throughput achieved while flooding a 2 MByte file, comparing UFlood with UFlood-R, MORE and MNP over all possible sources. On average, UFlood's throughput is 150% higher than that of MORE. UFlood also achieves 300% higher throughput than MNP, though it should be noted that MNP is not designed for throughput.

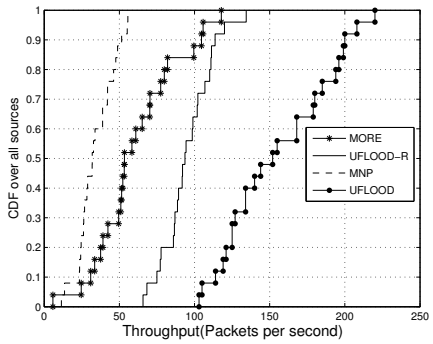

Fig. 9: UFlood achieves the highest throughput.

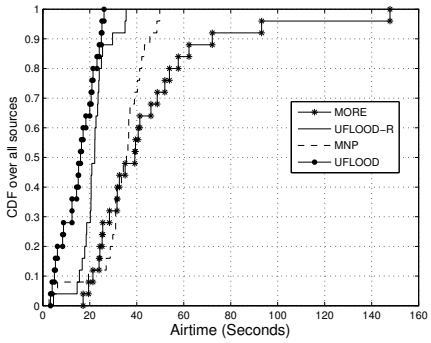

Fig. 10: UFlood consumes the least airtime.

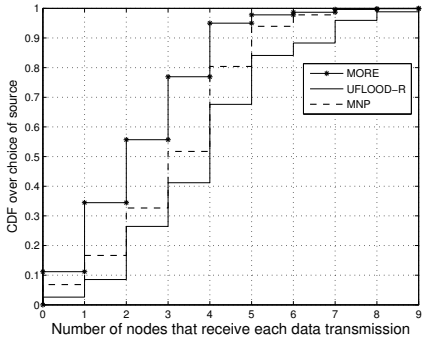

Fig. 11: UFlood reaches the most receivers with each transmission.

The gap between UFlood-R and UFlood shows the benefit of bit-rate selection. The gaps between UFlood-R and MORE and MNP show the benefit of UFlood's sender selection techniques. UFlood also benefits from better handling of hidden terminals than MORE, which sometimes suffers from persistent collisions. The performance improvement of UFlood over MNP is also due to UFlood's better handling of asymmetric links and its efficient feedback implementation (both interpolation and timing) which allows UFlood to send packets with less *idle time*; MNP requires explicit requests from receivers to trigger each transmission.

Figure 10 shows the airtime used during the 2 MByte transfer. UFlood uses 54% less airtime than MNP, and 65% less airtime than MORE. UFlood's lower airtime contributes to its higher throughput, and also reduces its impact on other wireless users.

D. Sender Selection

This section investigates how UFlood-R's sender selection (utility) allows it to send fewer packets than MORE or MNP in order to get the same amount of overall work done. We examine UFlood-R rather than UFlood in order to focus on sender selection alone, without bit-rate selection.

1) Number of receivers

One reason for UFlood-R's good performance is that it selects senders with many likely receivers. Figure 11 shows the CDF of the number of nodes that receive each data packet transmission. On average, UFlood-R transmissions reach 50% and 20% more nodes than MORE and MNP transmissions,

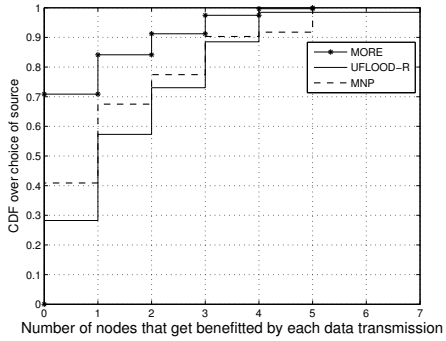

Fig. 12: UFlood's transmissions benefit many receivers

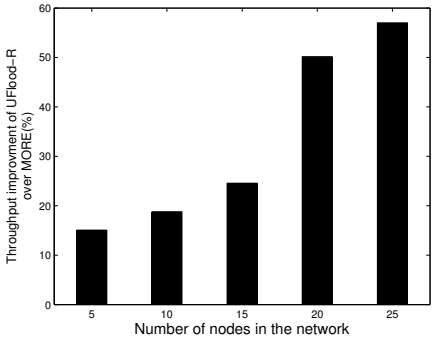

Fig. 13: Throughput of UFlood-R vs. MORE for different network sizes.

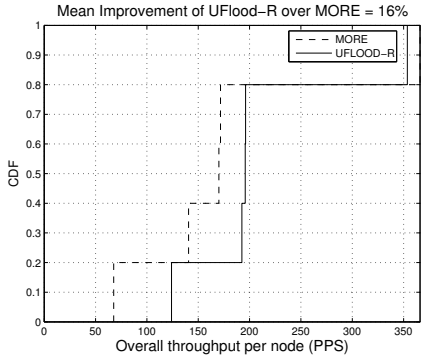

Fig. 14: Throughput in a dense network. CDF over all sources

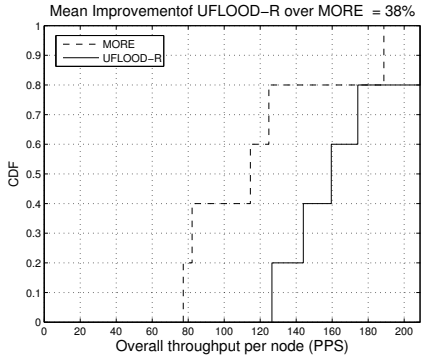

Fig. 15: Throughput in a sparse network. CDF over all sources.

respectively. That is, UFlood-R chooses senders with more receivers.

One reason MNP's senders reach fewer receivers than UFlood-R's is that MNP does not directly account for sender-to-receiver delivery probabilities. It is true that MNP favors senders that hear requests from many receivers, but link asymmetry and accidents of delivery can easily cause poor senders to receive more requests than good senders. UFlood-R, in contrast, uses measured forward link probabilities from sender to receivers in calculating utility.

2) Useful receptions

A second reason for UFlood-R's good performance is that it selects senders whose transmissions are likely to be useful at likely receivers, given the receivers' previous receptions. Figure 12 shows the CDF of the number of nodes that benefited from each data transmission. UFlood-R transmissions are useful to twice as many receivers as those of MORE, and 20% more than those of MNP. That is, UFlood-R transmissions are more likely to be linearly independent of data that receivers already hold, and are thus more likely to be useful in decoding the batch. This helps UFlood-R use fewer transmissions and complete flooding more quickly than MORE and MNP.

One reason UFlood-R out-performs MORE is that UFlood-R's sender decisions adapt as nodes receive packets. MORE decides which nodes should send via its TX_credit mechanism: the probability with which each MORE node transmits after receiving a packet (TX_credit) is fixed during a transfer. This causes problems towards the end of each batch, when a few nodes will likely be missing packets, but which nodes they are is hard to predict statistically; thus the best sender to satisfy those nodes is often not the one with the highest TX_credit. In contrast, UFlood-R uses feedback to adjust its choice of sender as a batch progresses, reflecting actual receptions.

UFlood's utility establishes priority among senders, rather than weighting them as with MORE's TX_credit. This helps in cases where one sender can be heard by a superset of the nodes that hear another sender. UFlood will usually cause just the former node to send, while MORE's TX_credit may cause both to send.

UFlood's feedback mechanism helps when there is correlated reception. Senders that have received the same packets as potential receivers suppress themselves. Similarly, when multiple potential senders have received a similar set of packets, feedback helps ensure that they do not forward needless copies of the data.

UFlood-R has an edge over MNP because MNP's sender choice is effectively based on more coarse-grained delivery probability measurements: whether or not senders and receivers hear single query and response packets.

E. Network density

We studied the effect of density using two 5-node networks: a dense network where all nodes can hear each other, and a sparse network where each node has a link to at most two other nodes. Figures 14 and 15 show the performance of UFlood-R and MORE in the two networks. The throughput advantage of UFlood-R is much larger in the sparse network. The low delivery probabilities in the sparse network cause different nodes to receive different packets, which increases the importance of feedback-based sender selection.

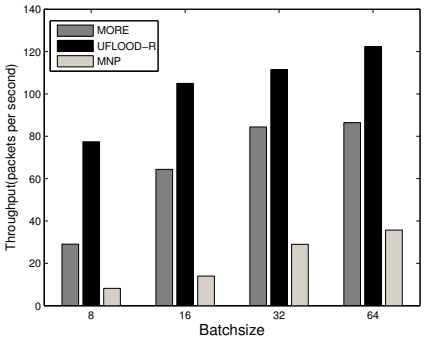

Fig. 16: Throughput for various batch sizes.

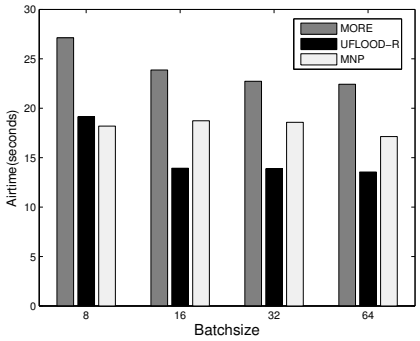

Fig. 17: Airtime for various batchsizes.

F. Batch size

Figures 16 and 17 show the performance of UFlood-R, MORE, and MNP on a 15-node network, varying the batch size. The graphs show that larger batches increase throughput and decrease airtime. Larger batches increase the effectiveness of coding, and also decrease the effect of the period of time towards the end of each batch when progress is slow while satisfying the last few nodes. Some of the latter effect could be reduced by overlapping successive batches.

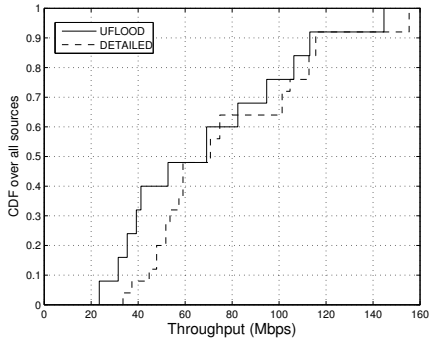

Fig. 18: Detailed Vs. UFlood's compact feedback representation. Compact feedback looses only 11% throughput due to conciseness.

G. Efficiency of Compact Feedback

UFlood uses several techniques IV-E- IV-H to reduce feedback traffic. In order to evaluate whether such techniques compromise UFlood's performance, we compare it with a detailed feedback mechanism (DETAILED), where nodes' feedback include detailed coefficients instead of compact representation. As mentioned in Section IV-A, such a feedback packet might be huge and often require multiple transmissions. Thus, we transmit the feedback for both UFlood and DETAILED schemes using ethernet to detach the overhead due to multiple transmissions.

In DETAILED feedback, each node broadcasts the coefficients of all its coded packets after every data transmission in the network. Nodes calculate utility for every transmission based on the up-to-date feedback from all the nodes in the network, which means their is no need for either bursty transmission or feedback interpolation. Figure 18 shows that the combination of techniques used by UFlood to reduce feedback traffic, leads to a 11% reduction in throughput compared to DETAILED. Considering the practical difficulties in using huge and frequent feedback in wireless mesh networks, this loss is acceptable.

H. Feedback overhead

Measurement shows that the overhead imposed by UFlood's feedback is about 3%. This is because, UFlood sends compact feedback, and sends it only when needed.

VI. Conclusion

This paper describes UFlood, a wireless mesh flooding protocol with a design incorporating the following novel ideas. First, *utility*, a heuristic to select senders on the basis of the expected rate of new information the senders will deliver to receivers, accounting both for delivery probabilities and previously received data. Second, the first bit-rate selection scheme for wireless mesh flooding. Third, a demonstration that detailed feedback about the data each receiver possesses is useful even with coding. Fourth, reduced feedback overhead with a compact representation and mechanisms to send feedback only when required. Experiments on a 25-node wireless mesh test-bed show that UFlood, on average, achieves 150% higher throughput than MORE, a high-throughput flooding protocol, using 65% less airtime. UFlood achieves 300% higher throughput using 54% less airtime than MNP, the existing flooding protocol to reduce airtime.

Acknowledgments

We thank Dina Katabi and John Guttag for helpful comments and suggestions. This work was supported in part by the National Science Foundation under grant CNS-0721702 and by Foxconn.

References

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, July 2000.

[2] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05*.

[3] Boundless. `http://www.boundlesss.com/wireless_video_surveillance.html`.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *SIGCOMM*, 2007.

[5] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. In *MobiCom*, 2004.

[6] Firetide. `http://www.tessco.com/yts/partner/manufacturer_list/vendors/firetide`.

[7] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava. Sensor network software update management: A survey. *Int. J. Network Management*, 15(4), 2005.

[8] T. Ho, M. Medard et al. A Random Linear Network Coding Approach to Multicast. *IEEE Trans. on Info. Theory*, 52(10), Oct. 2006.

[9] G. Holland, N. H. Vaidya, and P. Bahl. A rate-adaptive mac protocol for multi-hop wireless networks. pages 236–251, 2001.

[10] Y.-E. A. T. F. G. I. Hou, I-Hong; Tsai. Adapcode: Adaptive network coding for code updates in wireless sensor networks. In *INFOCOM*, 2008.

[11] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, 2004.

[12] S. Jaggi, P. Sanders et al. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6), June 2005.

[13] A. Kamerman and L. Monteban. Wavelan-ii: a high-performance wireless lan for the unlicensed band. *Bell Labs Technical Journal*, 2:118–133, 1997. doi: 10.1002/bltj.2069.

[14] I. Kang and R. Poovendran. Maximizing network lifetime of broadcasting over wireless stationary ad hoc networks. *Mobile Networks and Applications*, 10(6), 2005.

[15] I. Kang and R. Poovendran. Iterated local optimization for minimum energy broadcast. In *WIOPT*, 2005.

[16] B.-S. Kim and S. W. Kim. Dynamic rate adaptation for wireless multicast. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, 2009.

[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems* , 3(18), Aug. 2000.

[18] S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. In *Proc. 25th IEEE International Conference on distributed Computing Systems*, 2005.

[19] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 rate adaptation: A practical approach.

[20] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and

maintenance in wireless sensor networks. In *NSDI*, 2005.

[21] S. Li, R. Yeung, and N. Cai. Linear Network Coding. *IEEE Trans. on Info. Theory*, 49(2), Feb. 2006.

[22] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25, 1995.

[23] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, 8(2/3), 2002.

[24] N. Xiong, L. T. Yang, Y. Zeng, M. Chao, and J. H. Park. Implementation of rate control in distributed wireless multicast by neural network prediction. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 02*. IEEE Computer Society.