Practical Opportunistic Routing in High-Speed Multi-Rate Wireless Mesh Networks

Wei Hu, Jin Xie, Zhenghao Zhang
Computer Science Department
Florida State University
Tallahassee, FL 32306, USA
{hu,xie,zzhang}@cs.fsu.edu

ABSTRACT

Opportunistic Routing (OR) has been proven effective for wireless mesh networks. However, the existing OR protocols cannot meet all the requirements for high-speed, multi-rate wireless mesh networks, including: running on commodity Wi-Fi interface, supporting TCP, low complexity, supporting multiple link layer data rates, and exploiting partial packets. In this paper, we propose Practical Opportunistic Routing (POR), a new OR protocol that meets all above requirements. The key features of POR include: packet forwarding based on a per-packet feedback mechanism, block-based partial packet recovery, multi-hop link rate adaptation, and a novel path cost calculation which enables good path selection by considering the ability of nodes to select appropriate data rates to match the channel conditions. We implement POR within the Click modular router and our experiments in a 16-node wireless testbed confirm that POR achieves significantly better performance than the compared protocols for both UDP and TCP traffic.

Categories and Subject Descriptors

C.2.2 [**Computer-Communications Networks**]: Network Protocols – *Routing protocols*

General Terms

Algorithms, Design, Performance

Keywords

Opportunistic routing; Multi-rate; Partial packet recovery.

1. INTRODUCTION

Wireless mesh network is an attractive solution to extend network coverage at low cost. In this paper, we consider mesh network that consists of stationary mesh routers communicating with each other via wireless links. The challenge in such networks is to offer high performance over wireless links that are far less reliable than wired links due to noise, interference, and fading. It has been known that packet overhearing can be exploited to improve the network performance, because the packet forwarding path can be cut short when a packet is opportunistically received by nodes closer to the destination. In the literature, this is often referred to as Opportunistic Routing (OR).

To date, although many OR protocols have been proposed, such as MORE [3], ExOR [5], MIXIT [4], SOAR [6], Crelay [27], none can meet all the following five requirements we identify for wireless mesh networks.

Working with Existing Hardware: For practical reasons, a wireless mesh network prefers existing technologies, such as Wi-Fi. Protocols that require specialized hardware such as MIXIT may have weaker applicability.

Supporting TCP: Many important aspects of TCP, such as congestion control, are optimized under the assumption that the lower layer forwards individual packets. Therefore, protocols that send packets in batches, including ExOR, MORE, and MIXIT, have inherent difficulty in supporting TCP [7]. We believe it is important for a protocol to efficiently support TCP, given the ubiquitous deployment of TCP and the practical infeasibility of modifying the TCP implementations at end devices.

Low Complexity: To support high link layer data rates, the packet processing in an OR protocol should preferably be simple. Protocols such as MORE, MIXIT, and Crelay have high computation complexities in packet processing due to network coding or error correction and cannot support high data rates.

Multi-Rate Support: A wireless link layer usually supports multiple data rates; typically, lower data rates result in longer range and better reception. The actual speed of the network is often largely determined by a good rate selection algorithm. However, existing OR protocols often overlook the rate issue and assume a single operating rate.

Partial Packet Recovery: Partial packets, i.e., packets that contain just a few errors, often exist in wireless transmissions. By default, such packets are retransmitted; however, it is clearly more efficient to repair such packets with partial packet recovery schemes. The existing OR protocols with partial packets recovery are MIXIT and Crelay; however, as mentioned earlier, both suffer high computation complexity.

In this paper, we propose Practical Opportunistic Routing (POR), a new OR protocol which meets all the above requirements. POR achieves this by adopting a software solution that intelligently forwards individual packets with a simple yet efficient per-packet feedback mechanism, augmented by block-based partial packet recovery, efficient rate selection, and a novel path calculation method. We implement POR in the Click modular router [23] and test it in a

MobiHoc'13, July 29–August 1, 2013, Bangalore, India.

	runs on Wi-Fi	TCP friendly	low complexity	multi-rates	partial packets
MORE	√	×	×	×	×
ExOR	√	×	√	×	×
MIXIT	×	×	×	×	√
SOAR	√	√	√	×	×
Crelay	√	√	×	×	√
POR	√	√	√	√	√

Table 1: Protocols on practical requirements of OR.

16-node wireless testbed. Our results show that POR achieves significantly better performance than other compared protocols. We also demonstrate running TCP on top of POR, which, to the best of our knowledge, is the first demonstration of unmodified TCP on top of an OR protocol.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives an overview of POR. Section 4 describes the packet forwarding protocol. Section 5 describes routing. Section 6 describes rate selection. Section 7 describes our experimental evaluation. Section 8 concludes the paper.

2. RELATED WORK

In recent years, various OR protocols have been proposed and implemented, e.g., MORE [3], ExOR [5], MIXIT [4], SOAR [6], and Crelay [27]. We summarize the core differences between POR and the existing protocols in Table 1. OR has also been studied theoretically, such as in [15] [16]. We note that our focus is the design and implementation of a practical network protocol, which is different from finding theoretical network capacities [16]. The routing algorithm proposed for *anypath* in [15] provides valuable insights; however, the algorithm relies on the assumption that the packet receiving information at a node is known to every node, which can be difficult to implement in practical networks.

Partial packet recovery has attracted much attention in recent years [1, 2, 26]. We note that the existing works are typically developed for one-hop links while POR is developed for multi-hop networks. Rate selection is a classic topic in wireless networks. One of the major components of POR is also handling multiple data rates. However, POR is designed for multi-hop networks, therefore is different from rate selection algorithms for single-hop links such as [10, 11, 12, 13, 14]. Rate selection for multi-hop networks has been studied in [17, 18, 19] with focus on isolating collision loss from channel loss; we note that POR solves many additional problems in addition to rate selection and the solutions in [17, 18, 19] should complement POR. Like OR, network coding is another interesting approach that can improve the performance of wireless mesh networks. Schemes such as COPE [8] have been proposed which combine multiple packets from separate flows into one packet to reduce the number of transmissions. We note that network coding typically depends on the existence of multiple flows intersecting at a common node to create coding opportunities, while OR schemes such as POR can support both single flow and multiple flows because OR is based on packet overhearing. It is possible to combine POR with network coding, which we leave to future works due to the limit of space. It has also been proposed to adopt Time Division Multiple Access (TDMA) in the Medium Access Control (MAC) layer for mesh networks, which has shown notable performance improvements over the Carrier Sensing Multiple Access (CSMA) protocol adopted by Wi-Fi [9]. We note that POR mainly functions in the network layer and can work in collaboration with the new TDMA MAC to achieve even higher performance.

3. OVERVIEW

In POR, a packet is forwarded along a *path*, which is an ordered list of nodes. Any POR path must satisfy the *feedback constraint*; that is, any node on the path must be able to receive from its next hop node to ensure the correct reception of possible feedbacks from its downstream nodes. For simplicity, POR adopts source routing, i.e., the complete path is specified in the header of a packet. Packet in POR is divided into blocks; the checksum of each block is transmitted along with the packet such that a node can determine whether or not a received block is corrupted. A node may attempt to transmit blocks in a packet if none of its downstream nodes has received such blocks; the received blocks are not transmitted. A node announces the receiving status of a packet in a *feedback frame*; to avoid transmitting a packet that has been overheard, a node will not start to transmit a packet until a feedback from its downstream node has been received or until a timeout. With POR, nodes learn the link qualities at different data rates where the link quality is represented by the *Block Receiving Ratio* (BRR) defined as the fraction of correctly received blocks in a packet. A node may send small probe packets at selected rates such that its neighbors may discover the link qualities at such rates and report them back. Based on the link qualities, a node chooses the best data rate to minimize the packet delivery time of a path. The path is selected according to a greedy algorithm.

4. PACKET FORWARDING

The core of POR is its packet forwarding protocol discussed in this section. The goal of the protocol is to achieve high performance by exploiting packet overhearing and partial packets, and the challenge is to achieve this goal at low complexity, low overhead, while being friendly to TCP. To this end, we adopt a simple solution in which nodes send individual packets and determine the best strategy to forward each packet based on the per-packet feedback received from their downstream nodes. We discuss the details of the protocol in the following; for each aspect, we first describe the policy then add remarks to explain the policy. We refer to a source and destination pair as a *flow*.

4.1 Dividing Packets into Blocks

Policy: The source node divides a packet into blocks and calculates the checksums of the blocks which are transmitted in the frame header along with the packet. In our current implementation, each block is 150 bytes; the last block may be less depending on the size of the packet. When a received packet passes the packet level checksum test, there is no need to check the checksum for each block; otherwise, the packet is a partial packet and the node computes the checksums with the received blocks and compares them with those received in frame header to locate the corrupted blocks. In our current implementation, the checksum is 16 bits defined by $x^{16} + x^{15} + x^2 + 1$,

Remark: We note that dividing the packets into blocks allows POR to exploit the partial packets. POR does not use 32-bit checksum due to the better tradeoff between overhead and detection capability of the 16-bit checksum.

4.2 Sending a Packet at a Forwarder

Policy: A forwarder refers to a node on the path that is not the source or the destination. When a forwarder receives a packet with a sufficient amount of correct blocks, set as 50% in our current implementation, it adds the correct blocks of the packet to its queue. The node will not transmit the packet immediately because one of its downstream nodes may have received the packet or some blocks

of the packet correctly. Therefore, it will wait for a feedback of the packet for up to a timeout, which is set to be 20 ms in our current implementation, before sending any block of the packet. If the node receives the feedback from the downstream nodes and if the feedback indicates that it has all the blocks currently missing at the downstream nodes, it will schedule a transmission with such blocks; it removes the packet if the feedback indicates that the downstream nodes have no missing blocks. If it does not receive any feedback before the timeout, it will schedule a transmission for the entire packet if it received the packet correctly. The node retransmits a packet or blocks in a packet when a retransmission timer expires if it has not been informed by the downstream nodes that the packet or the blocks have been received correctly; each retransmission is separated by at least 15 ms and a packet is retransmitted up to 5 times, after which it is dropped.

Remark: We note that the key element of POR to exploit overhearing is to hold a packet and wait for the feedback. Although a node may hold a packet for up to 20 ms, in many cases, it may have received a feedback and can send the packet much earlier. In addition, through private communication with *Madcity*, a mesh network service provider, a path in a mesh network typically has no more than 4-5 hops, such that holding a packet for 20 ms will not increase the network delay excessively as the Internet packet delay is often in the order of 100 ms. We also note that a node does not keep packets with too few correct blocks because it will have to send feedbacks for such packets; our current policy is a simple heuristic to reduce the amount of overhead.

4.3 Exploiting Link Layer ACKs

Policy: In POR, when a packet is transmitted, it is encapsulated into a frame with the MAC address of the next hop node as the destination such that the next hop node may reply with a link layer ACK if the packet is received correctly. If a node receives an ACK for a packet, it immediately removes the packet from the queue; otherwise, the node will start the retransmission timer.

Remark: The key advantage of exploiting the link layer ACK is that it is sent immediately after the packet such that the node can determine whether the packet has been received correctly much faster than relying on the feedbacks. We note that encapsulating a packet in a frame with a specific MAC address does not prevent any other downstream nodes from overhearing the packet because the nodes process all overheard packets regardless of the MAC address. One potential problem may arise when a node receives a packet correctly and automatically sends the link layer ACK, but then is forced to drop the packet because the buffer is full. In this case, the sender of the packet has also removed the packet and the packet may be lost in the network. POR solves this problem by making sure that the buffer is rarely full with the congestion control mechanism described in Section 4.6.

4.4 Sending Feedback

Policy: In POR, a node may schedule a feedback when: 1) it has received a partial packet and is the next hop of the sender, or 2) it has received either a complete or a partial packet but is not the next hop of the sender. The feedback of a packet is 8 bytes containing the packet source node and desalination node IDs both in 2 bytes, the sequence number in 2 bytes, and a bitmap of the received blocks in 2 bytes where a bit '1' indicates that the corresponding block has been received correctly. A feedback may be sent immediately or may be combined with other feedbacks belonging to the same flow, in a *feedback frame*, to reduce the overhead. A feedback frame contains only feedbacks and is addressed to the previous hop node. A feedback frame may be sent under the following conditions: 1) 15 ms since a feedback was scheduled, or 2) 8 new feedbacks have been scheduled. A feedback frame may be acknowledged in the link layer; POR does not have other acknowledgment mechanisms for the feedback frames. To improve the reception, a feedback frame is always transmitted twice and may be sent at a lower rate than the data packets; in our current implementation, the rate is highest rate with Packet Receiving Ratio (PRR) no less than 0.8 on the link to the previous hop node. A node also "propagates the good news" when needed; that is, if the node receives a feedback from its downstream node, it will do a bitwise *or* with its own bitmap of the packet and use the new bitmap as the feedback.

Remark: A node does not schedule feedback to its previous hop node for complete packets because it should have sent the link layer ACK. A node waits for up to 15 ms before sending the feedback because its previous hop node holds the packet for up to 20 ms before transmitting the packet.

4.5 Buffer Management

Policy: In POR, a node keeps a single buffer which holds packets of all flows. In our current implementation, the buffer can hold 40 packets. In the buffer, packets belong to different flows are served round-robin. Packets belong to the same flow are served according to the sequence numbers, i.e., the packet with smallest sequence number will be sent first. If a node received a link layer ACK or a feedback for a packet indicating the downstream nodes have obtained all the blocks, it removes the packet from the buffer.

Remark: POR has a shallow buffer because a large buffer may lead to very long delay at the bottleneck links; the best action when congestion occurs should be reducing the sending speed at the source. POR sends packets with smaller sequence numbers first because this helps reducing the out-of-order delivery which may reduce the performance of TCP; we note that packets with larger sequence numbers may be received first due to overhearing.

4.6 Congestion Control

Policy: In POR, if the queue length exceeds a threshold, set to be half of the buffer size in our current implementation, the node will set a "congestion" bit in the feedback frame and the upstream node will cease to transmit any new packet to this node if it is the next hop. The upstream nodes are still allowed to transmit blocks for partial packets. Once the queue length is below the threshold, the node may send a feedback again to allow the upstream node to transmit.

Remark: This simple mechanism basically allows the node to drain its queue before its buffer overflows. The congestion bit will propagate from the bottleneck link back to the source because the upstream nodes will experience the congestion condition in turn when they cannot send packets to their downstream nodes.

5. ROUTING

POR adopts a customized routing module primarily because the path cost metric with POR is different from the existing metrics with its per-packet feedback and multiple data rates. We assume the link quality information of all links in the network at all rates are available to a node when it calculates paths, noting that the link quality information can be propagated by solutions such as that in OSLR [20]. Every node should compute the same path for each source and dentation pair if the nodes have consistent link quality information; as a protection against possible inconsistent link quality information due to packet loss, POR still embeds the entire path calculated by the source node in the header, which will not incur too much overhead as the paths are typically not very

long in mesh networks. We note that the routing module involves non-trivial calculations but will not hurt the overall simplicity of the POR protocol because routes are updated infrequently.

5.1 Path Metric

The path metric used in POR is different from existing path metrics such as ETT and ETX with two key distinctions. First, POR considers both the *forward cost* for sending data and *backward cost* for sending feedback, because the feedback cost in POR is not negligible. Second, we note that the links can be of different qualities at different times due to channel fluctuation. In this paper, we refer to each quality as a *state* of the link. The path metric in POR can capture the capability of nodes to adapt to the link qualities by changing rates, while existing metrics typically assume the rates are fixed.

We consider a given path denoted as $(v_1, v_2, ..., v_L)$. The path cost calculation is carried out iteratively, starting from the node closest to the destination. Therefore, when calculating the cost of path $(v_1, v_2, ..., v_L)$, the forward and backward costs of path $(v_i, v_{i+1}, ..., v_L)$ are known for $2 \leq i \leq L$, denoted as C_{v_i} and B_{v_i}, respectively.

5.1.1 Path Cost in Given State

We begin by finding the path cost when the links involving v_1 are in a certain given set of states denoted as τ.

Forward cost: We first consider the forward cost, denoted as $C^{\tau}_{v_1}$, which is defined as the consumed air time in data transmission in order to deliver a unit size block to the destination when the links are in state τ. The cost of delivering a packet is clearly proportional to the forward cost defined for a block. We assume the feedback is perfect and no node sends duplicate data. We denote the BRR of link $v_1 \rightarrow v_i$ at rate ρ_j as $\mu_i^{\rho_j,\tau}$. We use $C^{\tau}_{v_1,\rho_j}$ to denote the air time consumed to deliver a unit size block to v_L following path $(v_1, v_2, ..., v_L)$, under the condition that v_1 transmits at rate ρ_j. Let ρ^* denote the rate such that $C^{\tau}_{v_1,\rho^*} \leq C^{\tau}_{v_1,\rho_j}$ for any j, which is clearly the optimal rate v_1 should use. Assuming ρ^* can be found by the rate selection algorithm, $C^{\tau}_{v_1}$ is clearly just $C^{\tau}_{v_1,\rho^*}$. We note that when v_1 sends a block, if v_i receives the block correctly and is the one with the largest index, which occurs with probability $\mu_i^{\rho_j,\tau} \prod_{t=i+1}^{L}(1-\mu_t^{\rho_j,\tau})$, v_i will take over the responsibility of forwarding the block to the destination; if none of the downstream nodes of v_1 receives the block, which occurs with probability $\prod_{i=2}^{L}(1-\mu_i^{\rho_j,\tau})$, v_1 must repeat the process. Therefore,

$$\begin{aligned} C^{\tau}_{v_1,\rho_j} &= \sum_{i=2}^{L} \mu_i^{\rho_j,\tau} \prod_{t=i+1}^{L}(1-\mu_t^{\rho_j,\tau})(\frac{1}{\rho_j} + C_{v_i}) \\ &\quad + \prod_{i=2}^{L}(1-\mu_i^{\rho_j,\tau})(\frac{1}{\rho_j} + C^{\tau}_{v_1,\rho_j}), \end{aligned}$$

which can be reduced to

$$C^{\tau}_{v_1,\rho_j} = \frac{\frac{1}{\rho_j} + \sum_{i=2}^{L} \mu_i^{\rho_j} \prod_{t=i+1}^{L}(1-\mu_t^{\rho_j}) C_{v_i}}{1 - \prod_{i=2}^{L}(1-\mu_i^{\rho_j})}. \quad (1)$$

Clearly, ρ^* and $C^{\tau}_{v_1}$ can be found by a linear search.

Backward cost: We next consider the backward cost at v_1, which is denoted as $B^{\tau}_{v_1}$ and is defined as the consumed air time in feedback transmission in order to deliver a packet to the destination when the links are in state τ. The backward cost should be considered because the routing algorithm may otherwise select a path with too many nodes that may generate too many feedbacks. There are two differences between the forward and backward costs calculation. First, for the backward cost, only the complete packets are considered, because the feedback generating mechanism in POR is complicated for partial packets and the partial packet percentage is usually not very large. We note that with this simplification, the PRR is the same as the BRR. Second, unlike the forward cost that considers only the data transmission time, the backward cost considers the transmission time of the entire frame, because the feedbacks are small such that overhead such as frame header becomes significant. Note that when v_1 sends a packet $\mathbf{P}$ at rate ρ^*, if no downstream node receives $\mathbf{P}$, no feedback is sent. If at least one of the downstream nodes receives $\mathbf{P}$, the probability that v_i is the node with the largest index that receives $\mathbf{P}$ is

$$\gamma_i = \frac{\mu_i^{\rho^*,\tau} \prod_{t=i+1}^{L}(1-\mu_t^{\rho^*,\tau})}{1 - \prod_{i=2}^{L}(1-\mu_i^{\rho^*,\tau})}.$$

If $i = 2$, no feedback frames will be generated because v_2 will send a link layer ACK; otherwise, each of v_3 to v_i will generate a feedback and v_2 will generate a feedback in case it did not receive $\mathbf{P}$ correctly with probability $1 - \mu_2^{\rho^*,\tau}$. Therefore, the backward cost can be calculated according to

$$B^{\tau}_{v_1} = \gamma_2 B_{v_2} + \sum_{i=3}^{L} \gamma_i [B_{v_i} + (1-\mu_2^{\rho^*,\tau})\eta_2 + \sum_{t=3}^{i} \eta_t] \quad (2)$$

where η_t denotes air time v_t uses to send one feedback. η_t can be calculated according to the simplifying assumption that each feedback frame contains exactly 8 feedbacks, such that η_t is simply one-eighth of the feedback frame duration. We note that the feedback constraint of POR is enforced by the backward cost calculation, because η_t will be very large for some node v_t if v_t cannot find a good rate to send feedback.

5.1.2 Combining Path Cost in Multiple States

As mentioned earlier, a link may be of different qualities at different times and we call each quality a state. Eq. 1 and Eq. 2 find the path cost when the links involving v_1 are in a specific set of states. The actual cost of the path is simply the weighted average of the path costs in all possible sets of the link states, where the weight of a set is simply the probability that the links are in this particular set of states. If the probability that any individual link is in any state is known, the probability that the links are in a set of states can be calculated as the product of the individual link state probabilities by assuming the links are independent.

A practical challenge is to limit the computation complexity, as the number of sets can be very large. Therefore, in our current implementation, we make two simplifications. First, we assume v_1 can only reach nodes v_2, v_3, and v_4 when applying Eq. 1 and Eq. 2. Second, we assume each link has up to 3 states. Therefore, the total number of sets is capped at 27.

We note that the first assumption is a simple heuristics that usually works reasonable well in practice because overhearing at more than 3 hops away is usually rare. We support the second simplification by experimental data. In our experiments, we first measure the link qualities in 1-second intervals and generate for each link a vector where each element represents the BRR of one rate in this interval. In total, we measure 10 intervals, i.e., we have 10 vectors for each link. We note that if a link has only one state, all 10 vectors should be similar; otherwise, the vectors may appear as several clusters where each cluster corresponds to one state. To verify this, we run a clustering algorithm on the vectors. We define the *clustering distance* as the total distance between the vectors and the centers of the clusters they belong to. Fig. 1 shows the clustering

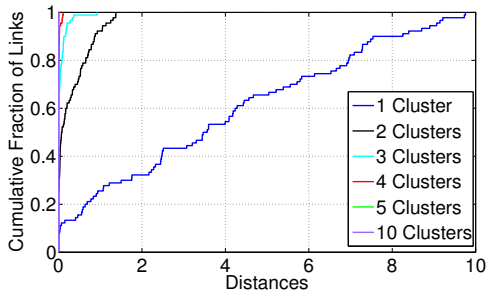


Figure 1: CDF of clustering distance.

distance when we set the maximum number of clusters to be 1 to 10, where 10 has the minimum clustering distance because there are only 10 vectors. We can see that: 1) the clustering distance can be quite large for many links when the number of cluster is 1, which means that many links have more than one state, and 2) the clustering distances are very small if the maximum number of clusters is 3 or above, which means that the links typically have no more than 3 states. The same clustering algorithm is used in POR implementation to classify the link states; we note that the probability that a link is in a certain state can be determined by the number of vectors in the cluster.

5.2 Finding the Path

We define the Optimal Practical Opportunistic Routing (OPOR) problem as finding a path with minimum cost according to the cost defined in Section 5.1. We report our findings about the theoretical aspects of the OPOR problem in [28]. In this paper, due to the limit of space, we describe the simple greedy routing algorithm we adopt, which is inspired by the Dijkstra's algorithm. The algorithm maintains a set π which contains the nodes whose paths to the destination have been determined; the set of nodes not in π is denoted as $\tilde{\pi}$. Each node keeps up to w candidate paths, where w is a constant set as 20 in our current implementation. Initially, π contains only the destination node. The algorithm iterates until all nodes have been added to π. In iteration i:

1. A node denoted as v_i is added to π, if v_i was previously in $\tilde{\pi}$ but has a candidate path with the least cost among all nodes in $\tilde{\pi}$.

2. The candidate paths for all nodes in $\tilde{\pi}$ are updated. That is, for a node in $\tilde{\pi}$ denoted as v_j, the algorithm evaluates its old candidate paths as well as new paths which starts with the link $v_j \rightarrow v_i$ then reach the destination following the candidate paths of v_i, and keeps w candidate paths with lowest costs.

6. RATE SELECTION

POR employs a novel rate selection module to select the data rate a node should use for a given path. The rate selection module considers the link qualities of multiple hops, instead of considering only the link to the next hop node, because a packet may be received by multiple downstream nodes in POR. One challenge, therefore, is to design the protocol to track the link qualities of multiple links. We note that while a node may be able to track the link qualities to its next hop node based on link layer ACKs, it may not be able to track the link qualities to nodes further on the downstream directly. Therefore, in POR, a node sends small dummy packets as probes at selected data rates and the neighboring nodes can measure the link qualities and report them back in special feedback frames. The key distinction between path calculation and rate selection is that the rate selection adapts to the instantaneous states of the links. A node may carry traffic for multiple flows; however, as each flow follows the same set of rules for rate selection, in the following, we will describe rate selection for one flow. The only additional issue with multiple flows is that the set of probed rates is the union of the probed rates of all flows.

6.1 Candidate Rates

Policy: We refer to the rates that should be probed as the *candidate rates*. The policies for determining the set of candidate rates at any moment are:

1. A rate in *hiatus* is not eligible, where a rate is in hiatus if it has been probed within the last 2 seconds.

2. A rate is eligible if the path cost according to its *estimated BRRs* is lower than the current path cost. For a rate, we define its *lower neighbors* as the rates lower than itself. We define the *immediate lower neighbor* as the highest lower neighbor currently in hiatus. If the immediate lower neighbor exists, the estimated BRRs are set as those of the immediate lower neighbor; otherwise, the estimated BRRs to the next two hops are assumed to be 1 and others 0. To calculate the path cost at any rate, the link BRRs of the target rate are plugged into Eq. 1 and Eq. 2; we note that the calculation does not have to iterate through multiple link states because the path cost for rate selection is the cost under the instantaneous link state.

3. If there are multiple eligible rates, the lowest 3 rates are added to the candidate set.

Remark: We make the following observations about candidate selection.

Observation 1. The hiatus mechanism prevents repeatedly probing a same rate.

Observation 2. If a rate is not in hiatus, its estimated BRRs are likely better than its actual BRRs because the estimated BRRs are either assumed to be the BRRs of the lower rates or simply 1 to nearby nodes. As a result, the path cost of the optimal rate based on its estimated BRRs are lower than the actual cost, which will ensure that it will be probed.

Observation 3. When determining the estimated BRRs for a rate without an immediate lower neighbor, a node assumes ideal BRRs only 2 hops away, because overhearing at more than 2 hops away is very rare if the path has been selected correctly; on the other hand, assuming ideal BRRs at nodes further downstream may be overly optimistic and may result in more unnecessary candidate rates.

Observation 4. No more than 3 rates are probed at any time to limit the overhead in probing. The probed rates will be in hiatus and eventually all eligible rates will be probed.

Observation 5. Adding the lowest 3 rates to the candidate set is a simple heuristic based on the following observation. We note that rate change is needed when channel changes. If the channel condition deteriorates, the optimal rate should be lower than the current rate and giving lower rates higher priority is clearly correct. If the channel condition improves, the optimal rate is higher than the current rate; in this case, the BRRs of the current rate are likely very good such that many lower rates will not appear better than the current rate, therefore not eligible, even when assuming ideal BRRs.

Observation 6. We define the *tracking time* as the time the node switches to the optimal rate when channel changes. We argue that *the tracking time is fast when the channel condition deteriorates and is bounded when the channel condition improves*, which is the desired behavior. We note that

- If the optimal rate is not in hiatus when the channel changes, as explained earlier, it will be probed, and will appear better than the current rate. In POR, the optimal rate is probed after a delay in the order of 100 ms. The node may spend time on rates that appear better than the optimal rate based on measurements taken before the channel change; however, it will soon realize that such rates are worse than the optimal rate. Therefore, the tracking time in this case is in the order of 100 ms.

- If the optimal rate is in hiatus when the channel changes, there are two cases. If the channel condition deteriorates, the path cost based on the old measurements will be better than the true path cost of the optimal rate; therefore, the optimal rate will be set as the current rate even before its measurements expires, and the tracking time should be small. If the channel condition improves, it may happen that the old measurements of the optimal rate do not indicate that it is a good rate; however, it will likely be probed after it comes out of the hiatus, such that the tracking time, although longer, is still bounded.

6.2 Probing

Policy: In our current implementation, a node transmits probe packets which contain one block at each candidate rate every 5 ms; for each probed rate, it sends a total of 20 probe packets each with a unique sequence number, after which it stops probing and will be expecting a special type of feedback called the *link quality feedback*. A node sends a link quality feedback every 100 ms, which contains all the measured BRRs from all links collected in the last 100 ms. A node will update the link quality records after receiving the feedback; it assumes the BRR is 0 if it cannot receive any feedback for a rate to another node 200 ms after it sends the first probe packet.

Remark: The probe packets have sequence numbers and are sent at fixed intervals; therefore, a node knows when the probing finishes and can send the feedback in time, even when it fails to receive the last probe packet in the batch. A node waits for 200 ms for a link quality feedback because by this time, it should have received the feedback unless the other node did not send the feedback or the feedback was lost, in either case an assumption of 0 as the BRR is reasonable. We experimentally evaluate the overhead of POR including probing in Section 7.

6.3 Rate Update

Policy: POR constantly monitors the link qualities at the current rate and updates the current rate if its path cost is higher than another rate currently in hiatus. POR updates the BRR of the current rate on the link to the next hop based on feedbacks and the link layer ACKs. POR updates the BRRs of the current rate on links to nodes further downstream based on the feedbacks from such nodes. The path cost of the current rate is reevaluated every time the BRR entries for the current rate is updated. The set of candidate rates is also refreshed every time the path cost of the current rate is updated.

6.4 Tests in Emulated Wireless Channel

We did a separate set of experiments specifically for the rate selection module because rate selection is difficult to evaluate in real-world wireless channels. We note that the rate selection module can be verified if it can be shown that the node always converges to the optimal rate in a timely manner. The challenge is that the wireless channels may constantly fluctuate such that it is often not possible learn the actual optimal rates during the experiments. Therefore, we design emulated channel experiments in which the nodes are very closely located such that real packet losses are rare; artificial packet losses and partial packets are introduced by inserting errors to certain packets according to the emulated channel conditions. We deploy a 3-node path and investigate the rate selection at the source node, because packet overhearing at nodes more than 2 hops away are rare such that rate selection is mainly dependent on the link qualities to the next 2 hops. Initially, the emulated channels are set at certain qualities and each node is set to run at the optimal rate under such link qualities; the link qualities are then changed randomly at a random time and stays the same for the rest of experiment.

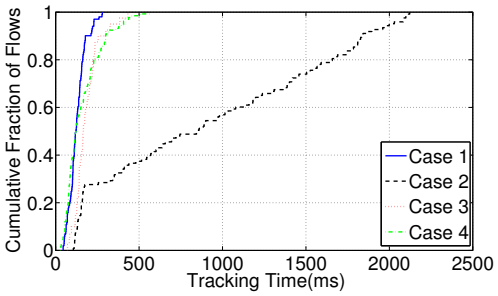


Figure 2: CDF of tracking time in 4 different cases.

We show in Fig. 2 the CDFs of tracking time in 4 cases:

- Case 1: The channel becomes better while the optimal rate is not in hiatus at the change. The tracking time is reasonably small, i.e., less than 300 ms, which is because the rate selection module will likely probe the optimal rate shortly after the channel change if the optimal rate is not in hiatus.

- Case 2: The channel becomes better while the optimal rate is in hiatus at the change. The tracking time is much larger and can be as large as around 2000 ms, which is because the optimal rate will be probed after it comes out of hiatus, while it can stay in hiatus for as long as 2000 ms after the channel change.

- Case 3: The channel becomes worse while the optimal rate is not in hiatus at the change. The tracking time is similar to Case 1 because rate selection module will likely probe the optimal rate shortly after the channel changes.

- Case 4: The channel becomes worse while the optimal rate is in hiatus at the change. It is interesting to notice that the track time is much smaller than that in Case 2. This is because at the time of the channel change, the existing record of the optimal rate, which was taken before the channel change, is better than its actual condition; as a result, the optimal rate will appear as a good rate to be attempted and will be subsequently set as the current rate before it comes out of hiatus.

Overall, we can see that the experiments confirm our observation that the rate selection module will converge to the optimal rate after a channel change, while the convergence is much faster if the channel becomes worse.

7. EVALUATION

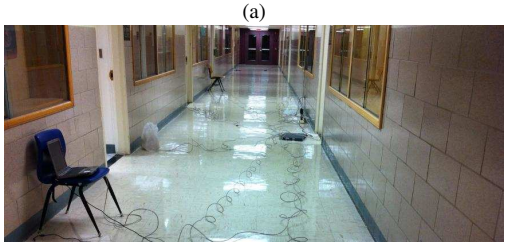
We implement POR within the Click modular router [23]. In this section, we describe our experimental results under single-flow UDP, multi-flow UDP, and TCP. We compare POR with two other schemes with the implementation from [24]: MORE, which is the benchmark OR protocol, and SPP, which is the traditional shortest path routing protocol without exploiting overheard packets and partial packets.

7.1 POR for A Single UDP Flow

7.1.1 Experiment Setup

Our testbed consists of 16 nodes which are laptop computers with the Cisco Aironet wireless card [22], scattered in one floor of a university building as shown in Fig. 3. We set wireless card in the 802.11a 5 GHz band where is no other traffic. To reduce the communication range and create multi-hop networks, we set the transmission power to be the minimum at 1 dBm and allow link rate at 24 Mbps or higher; we note that the experiments still cover a wide range of data rates from 24 Mbps to 54 Mbps. We connect all nodes with an additional Ethernet network which is a requirement to run MORE and SPP [24]. As MORE does not support multiple data rates, we set all nodes at the same rate of 24 Mbps in MORE experiments, because the results in earlier experiments with similar setup show that 24 Mbps tends to be the best among all rates for single rate networks. For SPP, we enable the auto-rate algorithm of the Madwifi driver [21] to adapt to the best rate of individual link. In each experiment, we send UDP packets of 1500 bytes from a source to a destination, referred to as a *flow*. We collect the results of 105 flows with path length no less than 2 according to POR; the number of flows with path lengths from 2 to 7 are 40, 32, 18, 8, 4, and 3, respectively. Each experiment lasts 65 seconds. Before running an experiment, we collect measurements of link quality, which are fed to SPP, MORE, and POR for routing computations.

(a)



(b)

Figure 3: (a). The layout of the testbed. (b). Testbed in the hallway.

7.1.2 Throughput Comparison

We run SPP, MORE and POR in turn for the selected flows. Fig. 4 shows the scatter plot of POR v.s. SPP and POR v.s. MORE, where each data point represents a flow with the throughputs of POR and the compared scheme as the y-coordinate and x-coordinate, respectively. We can see that POR is significantly better than SPP or MORE in most flows.

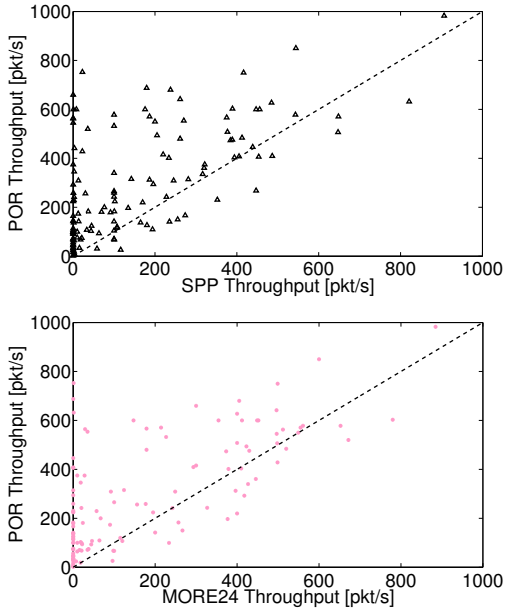


Figure 4: Scatter plot of the flow throughput.

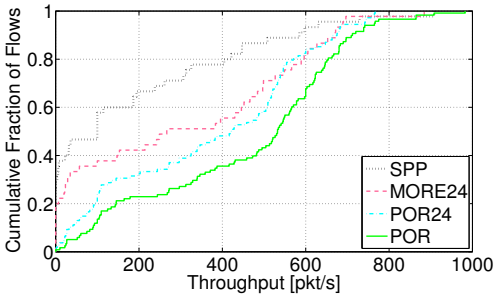


Figure 5: CDF of the flow throughput with POR24.

7.1.3 Throughput Analysis

The throughput gain of POR may be due to many reasons, such as exploiting overheard packets, supporting multiple rates, partial packet recovery, and better path selection. To verify the sources of the gain, we run more experiments with different configurations of POR.

First, we configure POR to run at a fixed rate of 24 Mbps, same as MORE, referred to as POR24. Fig. 5 shows the CDF of the throughputs of SPP, MORE, POR24, and POR. We observe that: 1) POR achieves significant gain due to good rate selection, which is evident from the performance difference between POR and POR24, and 2) POR efficiently exploits overheard packets with its feedback mechanism because POR24 can still achieve notable gains over MORE under the same rate.

Second, we run a modified version of POR referred to as cpPOR, which disables the reception of partial packets, where 'cp' stands for complete packet. Fig. 6 shows the CDF of the throughputs of SPP, MORE, cpPOR, and POR. We observe that partial packets recovery leads to significant gain, which is evident from the performance difference between POR and cpPOR.

Third, we run POR_M, in which the path cost is calculated assuming each link has only one state. Fig. 7 shows the throughputs of SPP, MORE, POR_M, and POR. We can see that when the throughput is high or low, POR_M performs similar as POR; in between, POR achieves higher throughput than POR_M. We believe

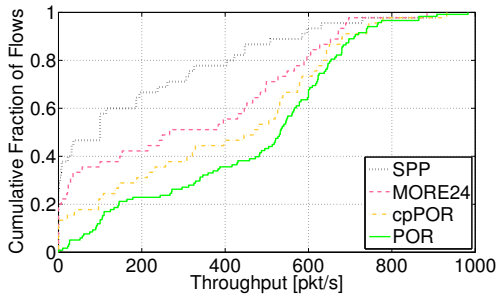


Figure 6: CDF of the flow throughput with cpPOR.

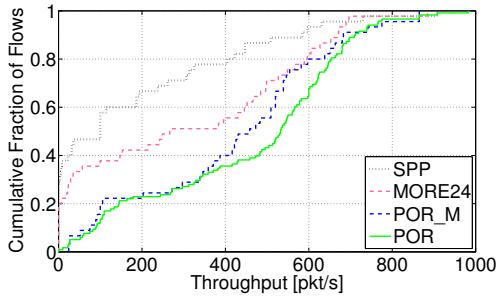


Figure 7: CDF of the flow throughput with POR_M.

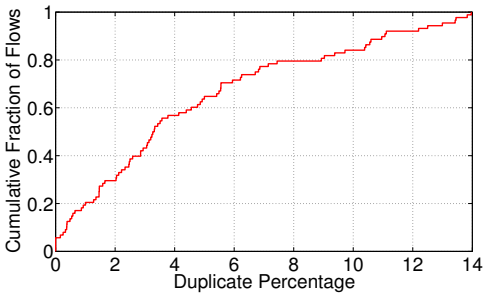


Figure 9: CDF of the duplicate percentage.

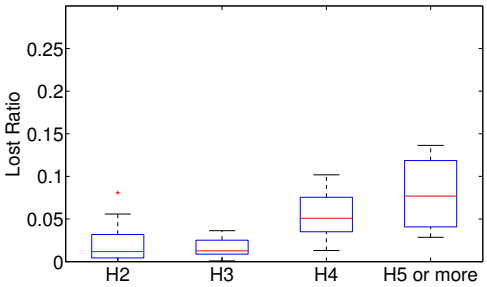


Figure 10: Packet lost ratio.

this is because links have fewer states when they are either very good or very bad; in between, they have more states and POR can better capture the true cost of such links and find better paths.

7.1.4 Various Aspects of POR

We also process the log files to examine various aspects of POR. **Overhead:** POR's overhead include the following. First, the opportunistic routing coordination overhead, which is mainly the feedback frames. Second, the multi-rate probe overhead, which is mainly the probe packets. Third, the partial packet recovery overhead, which includes the headers and checksums in POR data packets. To quantitatively measure the overhead, we define the overhead percentage of a flow as the total air time all nodes transmit the overhead over the total air time all nodes transmit packets. We note that this takes into account the possibility that two nodes may transmit simultaneously due to spatial multiplexing. Fig. 8 shows the CDF of the overhead percentage of POR for each flow, where we can see that the overhead is reasonably low with a median of 10.36%.

Duplicate Percentage: We say the transmission of a data block is a *duplicate* if at least one of the downstream nodes has received the block correctly. To measure the duplicate percentage, in the experiments, each node increments a counter when it receives a block, and increments another counter when the received block is already in the buffer. To avoid a transmission being counted more than once, a node only increments the counters for transmissions from its previous hop node. In the end, we sum up the two counters of all nodes and use the ratio as an approximation of the fraction of the duplicate. Fig. 9 shows the CDF of the duplicate percentage, where we can see that POR achieves a very small duplicate percentage with a median of 3.29%. This further confirms that the feedback mechanism is effective and allows nodes to avoid unnecessary transmissions.

Packet Loss: Fig. 10 shows the boxplot of the end-to-end packet lost ratio of each flow classified according to path lengths. We can see that POR maintains a low end-to-end packet lost ratio: the median of the lost ratio is less than 1% when the number of hops is no more than 3, and is less than 5% when the number of hops is 4. We note that in practical mesh networks, paths of length more than 4 are rare because such paths cannot sustain high throughput. In addition, we find that the longer paths in our experiments usually contain very bad links and loss is in some sense inevitable. Fig. 11 shows the relation between packet lost ratio and path cost when the hop count of path is 4. The results indicate that packet lost ratio increases when the path cost increases, which is because high cost paths may include some very bad links.

Packet Delay: Fig. 12 shows the boxplot of end-to-end packet delays of POR classified according to path lengths. We can see that the packet delays are typically around 100 ms and the variance is reasonably small for short paths.

Queue Length: Fig. 13 shows the CDF of the average node queue length for nodes on the packet forwarding path in each experiment. We can see that in most cases, the buffers are not full and the queue lengths are around 10 packets, which confirms that the congestion control mechanism works reasonably well.

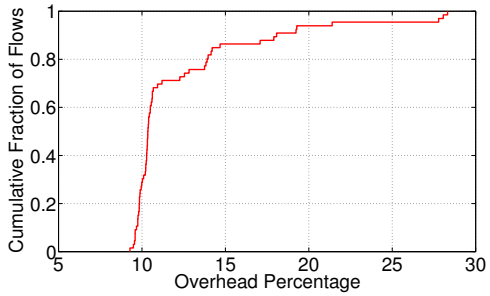


Figure 8: CDF of the overhead percentage.

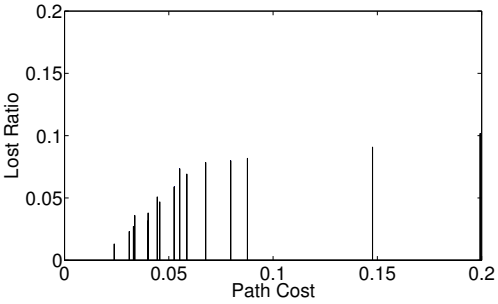


Figure 11: Packet lost ratio and path cost.

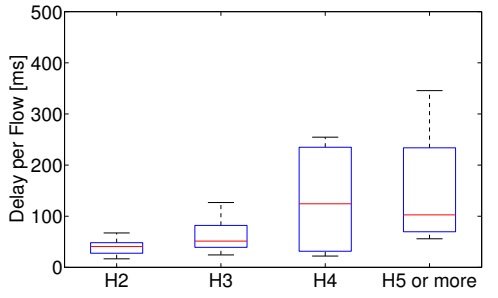


Figure 12: Packet delay.

7.2 POR for Multiple UDP Flows

We also run experiments to demonstrate the capability of POR to support multiple flows in a testbed similar to that for the single-flow UDP experiments. Since the MORE implementation at [24] does not support multiple flows, we compare POR only with SPP. We run 40 experiments by randomly choosing source and destination pairs, varying the number of random active flows in the network from 2 to 4. Fig. 14 show the average per-flow throughput as a function of the number of flows. The throughput of multiple flows depends on many issues out of the scope of this work, such as network-wide load-balancing; still, we can see that POR achieves higher throughput than SPP.

7.3 TCP Performance with POR

We also run TCP experiments on top of POR. To the best of our knowledge, prior to this, no demonstration has been made to run TCP on top of an OR protocol, because many existing OR protocols require packet batching and many will disrupt packet order. In the single UDP flow experiments, we have demonstrated that POR can achieve low end-to-end loss and delay, which should allow POR to support TCP. To verify this, we implement a standard TCP with the

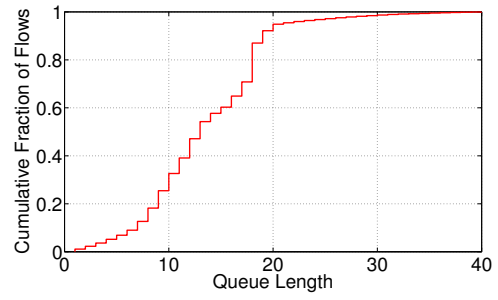


Figure 13: CDF of the average queue length.

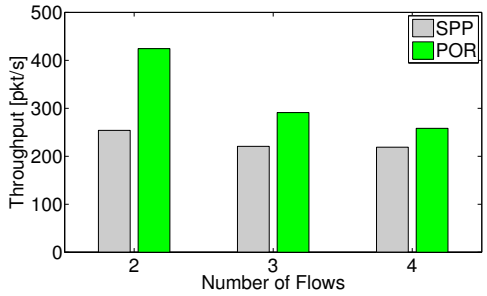


Figure 14: Average flow throughput with multiple flows.

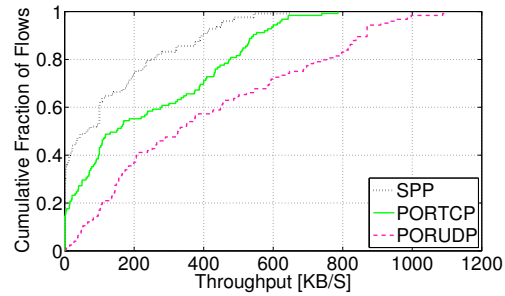
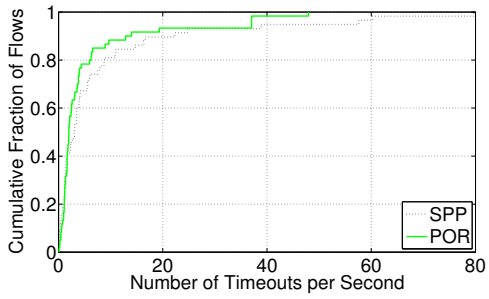


Figure 15: CDF of the TCP throughput on top of SPP and POR and UDP throughput on top of POR.

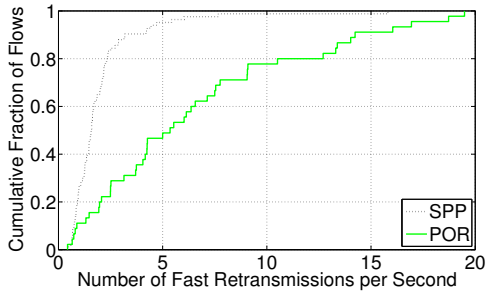
available code from [25] within the Click modular router [23]. We run TCP on top of POR and SPP in a testbed similar to that for the single-flow UDP experiments for 46 flows. We also run UDP on top of POR during the experiments to find the upper bound of the throughput.

Fig. 15 shows the CDF of the TCP throughput on top of SPP and POR, as well as the CDF of UDP throughput on top of POR. We can see that POR achieves a sizable gain over SPP for TCP. We believe *this is the first real-world demonstration of unmodified TCP over an OR protocol with a performance gain*. The TCP throughput of POR is less than UDP, which is partly because TCP has both the forward flow for data and the backward flow for TCP ACKs, partly because TCP responds to packet loss and packet reordering by cutting down the sending rate.

Fig. 16(a) shows the CDF of the average number of TCP timeouts per second in our experiments. We can see that TCP has less timeouts when running on top of POR than on SPP. This is because in SPP, congestion collapse can also trigger the timeout; on the contrary, POR adopts a congestion control mechanism to prevent congestion in the network. Fig. 16(b) shows the CDF of the average number of TCP fast retransmissions per second in our experiments. We can see that TCP tends to have more fast retransmissions on top of POR than on SPP. In POR, packets within the same TCP flow may travel through different nodes toward the destination, resulting in out-of-order reception at the destination. Such out-of-order reception is undesirable because TCP may interpret an out-of-order reception as a lost packet, and invoke fast retransmission to resend the packet as well as reducing the congestion window size. In an OR protocol, out-of-order receptions are inevitable because overheard packets, by their nature, arrive out-of-order. POR controls out-of-order reception by forwarding packets with smaller sequence number first; we can see that the number of actual fast retransmissions is reasonably small with a median around 5. More importantly, POR compensates for the performance loss due to



(a)



(b)

Figure 16: (a). CDF of TCP timeouts per second. (b). CDF of TCP fast retransmissions per second.

out-of-order receptions with other mechanisms and still achieves throughput gain over SPP.

8. CONCLUSIONS

In this paper, we propose POR, a new Opportunistic Routing (OR) protocol for high-speed, multi-rate wireless mesh networks that runs on commodity Wi-Fi interface, supports TCP, has low complexity, supports multiple link layer data rates, and is capable of exploiting partial packets for high efficiency. We believe POR is the first OR protocol that meets all such requirements, and thus significantly advances the practicability of OR protocols. POR adopts a per-packet feedback mechanism in packet forwarding to avoid sending data that has been received by the downstream nodes. POR also incorporates a block-based partial packet recovery scheme, a multi-hop link rate adaptation scheme, and a novel path cost calculation method which enables good path selection. We implement POR within the Click modular router. Our experiments in a 16-node testbed confirm that POR achieves significantly better performance than the compared protocols. We also demonstrate running unmodified TCP on POR, which is the first demonstration of unmodified TCP on an OR protocol with performance gain over traditional routing protocol to the best of our knowledge. Our future work includes extending POR to exploit features in 802.11n and 802.11ac networks.

9. REFERENCES

[1] K. Lin, N. Kushman, and D. Katabi, "ZipTx: Harnessing partial packets in 802.11 networks," in *ACM Mobicom*, 2008.

[2] B. Han, A, Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller, "Maranello: Practical partial packet recovery for 802.11," in *USENIX NSDI, 2010.*

[3] S. Chachulski, M. Jennings, S. Katti and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *ACM Sigcomm*, 2007.

[4] S. Katti, D. Katabi, H. Balakrishnan and M. Medard, "Symbol-level network coding for wireless mesh networks," in *ACM Sigcomm*, 2008.

[5] S. Biswas and R. Morris, "Opportunistic routing in multi-hop wireless networks," in *ACM Sigcomm*, 2005.

[6] E. Rozner, J. Seshadri, Y. A. Mehta, and L. Qiu. "SOAR: Simple opportunistic adaptive routing protocol for wireless mesh networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 12, pp, 1622-1635, 2009.

[7] T. Li, D. Leith, and L. Qiu, "Opportunistic routing for interactive traffic in wireless networks," in *IEEE ICDCS*, 2010.

[8] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," in *ACM Sigcomm*, 2006.

[9] V. Sevani, B. Raman and P. Joshi, "Implementation based evaluation of a full-fledged multi-hop TDMA-MAC for WiFi mesh networks", *IEEE Transactions on Mobile Computing*, to appear.

[10] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Predictable 802.11 packet delivery from wireless channel measurements," in *ACM Sigcomm*, 2010.

[11] G. Judd, X. Wang, and P. Steenkiste, "Efficient channel-aware rate adaptation in dynamic environments," in *ACM MobiSys*, 2008.

[12] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, "AccuRate: Constellation based rate estimation in wireless networks," in *USENIX NSDI*, 2010.

[13] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation," in *ACM Sigcomm*, 2009.

[14] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan, "Robust rate adaptation for 802.11 wireless networks," in *ACM MobiCom*, 2006.

[15] R. Laufer, H. D.-Ferrière, and L. Kleinrock, "Polynomial-time algorithms for multirate anypath routing in wireless multihop networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 3, pp. 743-755, Jun. 2012.

[16] K. Zeng, W. Lou, and H. Zhai, "Capacity of opportunistic routing in multi-rate and multi-hop wireless networks," *IEEE Transactions on Wireless Communications*, vol. 7, no. 12, pp. 5118-5128, Dec., 2008.

[17] P. A.K. Acharya, A. Sharma, E. M. Belding, K. C. Almeroth, and K. Papagiannaki, "Rate adaptation in congested wireless networks through real-time measurements," *IEEE Transactions on Mobile Computing*, vol. 9, no. 11, pp. 1535-1550, Nov., 2010.

[18] E. Ancillotti, R. Bruno, and M. Conti, "Experimentation and performance evaluation of rate adaptation algorithms in wireless mesh networks," in *Proceedings of the 5th ACM symposium on performance evaluation of wireless ad hoc, sensor, and ubiquitous networks (PE-WASUN '08)*, pp. 7-14, 2008.

[19] J. C. Park and S. K. Kasera, "Reduced packet probing multirate adaptation for multi-hop Ad Hoc wireless networks," *IEEE Symposium on World of Wireless, Mobile and Multimedia Networks*, June 2009.

[20] Optimized Link State Routing Protocol (OLSR), *http://www.ietf.org/rfc/rfc3626.txt*.

[21] The MadWifi Project, *http://madwifi-project.org/*.

[22] Cisco Aironet 802.11a/b/g wireless cardbus adapter, http://www.cisco.com/.

[23] The Click Modular Router, *http://read.cs.ucla.edu/click/*.

[24] *http://people.csail.mit.edu/szym/more/README.html*.

[25] D. Levin, H, Schioberg, R. Merz and C. Sengul, "TCPSpeaker: Clean and dirty sides of the same slate," *Mobile Computing and Communications Review*, vol. 14, no. 4, pp. 43-45, 2010.

[26] J. Xie, W. Hu, and Z. Zhang, "Revisiting partial packet recovery in 802.11 wireless LANs," in *ACM Mobisys*, 2011.

[27] Z.Zhang, W. Hu, and J. Xie, "Employing coded relay in multi-hop wireless networks," in *IEEE Globecom*, 2012.

[28] W. Hu, J. Xie, and Z. Zhang, "The complexity of the routing problem in POR," *Technical Report TR-130611*, Computer Science Department, Florida State University, 2013.