Simple Opportunistic Routing Protocol for Wireless Mesh Networks

Eric Rozner, Jayesh Seshadri, Yogita Mehta, and Lili Qiu
{erozner,jayeshs,yamehta,lili}@cs.utexas.edu
The University of Texas at Austin

***Abstract*— Multihop wireless mesh networks are becoming a new attractive communication paradigm. Many cities and public places have deployed or are planning to deploy mesh networks to provide Internet access to residents and local businesses. Routing protocol design is critical to the performance and reliability of wireless mesh networks. Traditional routing protocols send traffic along pre-determined paths and have been shown ineffective in coping with unreliable and unpredictable wireless medium. In this paper, we develop a Simple Opportunistic Adaptive Routing protocol (SOAR) for wireless mesh networks. SOAR maximizes the progress each packet makes by using priority-based timers to ensure that the most preferred node forwards the packet with little coordination overhead. Moreover, SOAR minimizes resource consumption and duplicate transmissions by judiciously selecting forwarding nodes to prevent routes from diverging. To further protect against packet losses, SOAR uses local recovery to retransmit a packet when an ACK is not received within a specified time. SOAR uses a combination of selective ACKs, piggyback ACKs, and ACK compression to protect against ACK loss while minimizing ACK overhead. We evaluate SOAR using NS-2 simulations. Our preliminary results show that SOAR is promising to achieve high efficiency and effectively support multiple simultaneous flows.**

I. Introduction

Wireless mesh networks are becoming a new attractive communication paradigm owing to their low cost and rapid deployment. Routing is critical to the performance and reliability of mesh networks. In this paper, we present a novel routing protocol, called Simple Opportunistic Adaptive Routing (SOAR), for wireless mesh networks.

A. Benefits of Opportunistic Routing

A natural approach to routing traffic in wireless mesh networks is to adopt shortest path routing schemes as in wireline networks. These schemes select a shortest path (according to some metric) for each source-destination pair and send traffic along the pre-determined path. Most of the existing routing protocols, such as DSR [5], AODV [8], DSDV [7], and LQSR [3], fall into this category. They are also referred to as traditional routing.

Recently, researchers have proposed opportunistic routing for mesh networks. Opportunistic routing differs from traditional routing in that it exploits the broadcast nature of wireless medium and defers route selection after packet transmissions. This can cope well with unreliable and unpredictable wireless links. There are two major benefits in opportunistic routing.

First, opportunistic routing can combine multiple weak links into one strong link. For example, consider a source that has 20% delivery rate to each of its five neighbors, and each of these neighbors have 100% delivery rate to the destination. Under a traditional routing protocol, we have to pick one of the five intermediate nodes as the relay node, and cannot take advantage of a transmission that reaches the nodes other than the selected relay node. So altogether we need 5 transmissions on average to send a packet from the source to the relay node, and 1 transmission from the relay node to the destination. In comparison, under opportunistic routing, we can treat the five intermediate nodes as one unit that cooperatively forwards the packet to the destination. The combined link has a success rate of $1-(1-0.2)^5 = 0.672$. Therefore, on average only 1/0.67=1.487 transmissions are required to deliver 1 packet to at least one of the five intermediate nodes, and another transmission is required for an intermediate node to forward. Altogether it takes only 2.487 transmissions to deliver the packet end-to-end, thereby achieving 2.4 times the throughput of traditional routing.

Second, a traditional routing protocol has to trade off between link quality and the amount of progress each transmission makes. For example, consider a linear topology where A sends data to D along the path $A-B-C-D$ and loss rate increases with distance. If B is used as the next hop, then the quality of link $A-B$ is good, and no retransmission is required to deliver the packet to B. But the progress made is small. Alternatively, if C is chosen as the next hop, a large progress is made if the packet reaches C. However since the quality of link $A-C$ is poor, multiple transmissions are required to deliver the packet to C. In comparison, opportunistic routing does not commit to B or C before transmissions. Among the nodes that receive the packet, we choose the one closest to the destination to forward. In this way, we can opportunistically leverage transmissions that are either unexpectedly short or unexpectedly long, thereby achieving high throughput.

B. Challenges for Opportunistic Routing

The major challenge in opportunistic routing is to maximize the progress of each transmission without causing duplicate transmissions or incurring significant coordination overhead. In order to realize the potential benefits of opportunistic transmissions in real networks, a practical opportunistic routing protocol should achieve the following design goals:

- Efficient: It should achieve significant performance improvement over traditional routing.
- Flexible: The protocol should support diverse traffic patterns, including multiple simultaneous flows.

C. Prior Work

ExOR [1] is the seminal opportunistic routing protocol. In ExOR, a sender broadcasts a batch of packets (10-100 packets per batch). Each packet contains a list of nodes that can potentially forward it. In order to maximize the progress each transmission makes, the forwarding nodes relay data packets in the order of their proximity to the destination, as measured using ETX [2]. To minimize redundant transmissions, ExOR uses a batch map that records which packets each node has received; every forwarding node only forwards data that has not been acknowledged by the nodes closer to the destination in their batch maps.

ExOR provides significant throughput improvement over traditional routing and achieves the first design goal. In particular, it offers throughput gains of 35% over one and two hop connections, and a gain of a factor of 2-4 for more distant pairs. On the other hand, ExOR cannot support multiple simultaneous flows, which is common in mesh networks. This limits its practical use in real mesh networks.

D. Our Approach

To this end, we develop a simple opportunistic adaptive routing protocol (SOAR) towards achieving the above two design goals. To take advantage of transmissions that reach nodes other than the next-hop, we introduce a novel mechanism called priority-based forwarding. Priority-based forwarding maximizes the progress each packet makes by choosing the node closest to the destination to forward the packet. Different priorities are realized by using priority-based timers: the node with highest priority performs forwarding first, and other nodes hearing the transmission automatically cancel their transmissions, thereby minimizing the number of duplicate transmissions in a cheap and distributed way. To make priority-based timers work, all the forwarding nodes should hear each other with a high probability. To ensure this condition, we judiciously select forwarding nodes to avoid diverging routes. To further protect against packet losses, SOAR uses local recovery to retransmit a packet when an ACK is not received within a specified time period. SOAR uses selective ACKs to protect against ACK losses and minimize unnecessary retransmissions, and uses piggyback ACKs and ACK compression to reduce ACK overhead.

To demonstrate its effectiveness and feasibility, we implement SOAR in the NS-2 simulator [6]. Our preliminary results show that SOAR is promising to achieve both efficiency and flexibility in wireless mesh networks.

E. Paper Outline

The rest of the paper is organized as follows. We describe SOAR protocol in Section II. We present our preliminary evaluation of SOAR using NS-2 simulations in Section III. We conclude in Section IV.

II. Simple Opportunistic Adaptive Routing Protocol (SOAR)

In this section, we first give an overview of SOAR and then describe the protocol details.

A. Overview

The major challenge in opportunistic routing is to maximize the progress of each transmission while minimizing duplicate transmissions and coordination overhead. To achieve this goal, a sender in SOAR selects a shortest path to forward the packet towards the destination. As described in Section I-A, routing strictly along the shortest path is not efficient under unreliable and unpredictable wireless links. Therefore, we introduce a novel mechanism called priority-based forwarding to take advantage of path diversity.

Specifically, in SOAR a sender broadcasts a data packet that includes an ordered list of forwarding nodes. Upon hearing the transmission, the nodes not on the forwarding list simply discard the packet. Nodes on the forwarding list store the packet and set timers based on their priorities. A node with higher priority uses a smaller timer so that its timer expires earlier; once the timer expires, the node forwards the packet. Other nodes, upon hearing the transmission, will remove the corresponding packet from their queues to avoid duplicate transmissions.

SOAR relaxes the actual route that data traverses to be along or near the shortest path. Different from traditional routing, SOAR leverages path diversity by using more flexible routes: nodes other than the next hop can forward the data. Different from ExOR, in SOAR the nodes involved in routing a packet are constrained to be near the shortest path, as shown in Figure 1. This prevents routes from diverging and minimizes duplicate transmissions. Moreover, this also simplifies coordination since all the nodes involved are close to nodes on the shortest path and can hear each other with a reasonably high probability. Therefore, we can use overheard transmissions to coordinate with each other in a cheap and distributed way.

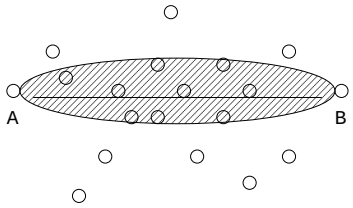


Fig. 1. The actual route in SOAR involves nodes on or near the shortest path. In the figure, the nodes in the shaded region participate in forwarding packets from A to B.

B. Shortest Path Selection

A sender S selects a shortest path to route towards destination D. There are several routing metrics that we could use to select the shortest path. We use the ETX metric, a state-of-the-art routing metric proposed by De Couto *et al.* [2]. A link's ETX metric measures the expected number of transmissions (including retransmissions) required to send a unicast packet across the link. Let p_f and p_r denote the loss probability of the link in the forward and reverse directions, respectively. Each node measures loss rate of its links to and from its neighbors (*i.e.*, p_f and p_r) by broadcasting one probe packet every second and counting the number of probes received in the last 10 seconds. Then, the link's ETX metric is calculated

as $\frac{1}{(1-p_f)\times(1-p_r)}$, assuming independent packet losses. Each node maintains an exponentially weighted moving average of ETX samples. The shortest path from S to D is the path with smallest ETX.

C. Priority-based Forwarding

We use the following priority-based forwarding to maximize the progress each transmission makes. A sender transmits a packet, which specifies a list of forwarding nodes in an increasing order of ETX towards destination. Each node hearing the packet first checks if it is included in the forwarding list. If not, it discards the packet. Otherwise, it sets forwarding timer proportional to its position in the forwarding list. So the node with lower ETX towards the destination forwards the packet earlier, and other nodes hearing its forwarding will cancel their forwarding timer and remove the packet from their queues, thereby avoiding duplicate forwarding.

Our priority-based forwarding has some similarity with the MAC-layer anycast mechanism proposed in [4]. In [4], the sender sends a RTS, and multiple receivers respond to the RTS in the order of their proximity to the destination. Among those that receive the RTS, the one closest to the destination sends the CTS first, and becomes selected to receive the subsequent data packet. Different from [4], we directly maximize the progress of data packets by only selecting the node that receives the data packet to forward, whereas in [4] the reception of RTS does not guarantee reception of the subsequent data packet. Moreover, SOAR is a protocol at the network-layer, which is very different from MAC-layer anycast in [4].

Next we address the following important issues in priority-based forwarding: (i) how to select forwarding nodes? and (ii) when to forward?

```
forwardList = ();
if i is on the shortest path to the destination
  for each node j in the topology
    if (ETX(j, dest) < ETX(i, dest) and ETX'(i, j) < threshold)
      add j to forwardList
    end
  end
  // further prune the forwarding list to ensure their ETX' to each
  // other are within threshold
  forwardList = Prune(forwardList, threshold);
else // i is not on shortest path
  find j such that its ETX' to i is smallest among all nodes in shortest path
  if ETX(j, dest) < ETX(i, dest)
    add j to forwardList
  end
  foreach node k in j's forwardList
    if (ETX(k, dest) < ETX(i, dest) and ETX'(i, k) < threshold)
      add k to forwardList
    end
  end
end
```

Fig. 2. Selection of forwarding nodes at node i, where $ETX'(i,j)$ is the ETX metric on link i-j, and $ETX(i,j)$ is the shortest path from i to j in terms of ETX metric.

1) Forwarding node selection: Figure 2 shows the pseudo-code we use to select forwarding nodes. As it shows, if a node i is on the shortest path, i selects the forwarding nodes that satisfy the following three conditions: (i) the forwarding node's ETX to the destination is lower than i's ETX to the destination, and (ii) the forwarding node's ETX' to i is within a threshold, and (iii) the ETX' between any pair of forwarding nodes is within a threshold. The first constraint ensures the packet makes progress, and the second and third constraints ensure that the link quality between i and its forwarding node j or between any two forwarding nodes is reasonably good so that they can hear each other's forwarding and avoid duplicate transmissions.

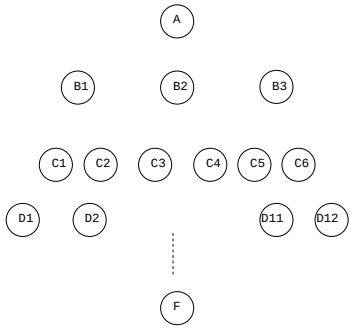


Fig. 3. Careful forwarding node selection is necessary to prevent routes from diverging.

SOAR also allows nodes not on the shortest path to forward packets. How should these nodes select forwarding nodes? One possibility is to select the forwarding nodes in the same way as the nodes on the shortest path (described above). However this is not desirable since this could result in diverging forwarding paths and duplicate transmissions. For example, in Figure 3 node A wants to send traffic to node F. A selects $B1$, $B2$, and $B3$ as forwarding nodes; $B1$ selects $C1$, $C2$, $C3$, $C4$ as forwarding nodes, while $B3$ selects $C3$, $C4$, $C5$, $C6$ as forwarding nodes. Then if $B1$ and $B3$ do not hear each other's forwarding of a packet (due to some packet losses between $B1$ and $B3$), they each forward a copy of packet, and thus the packet may be forwarded on to $C1$, $C2$, $C3$, $C4$, $C5$, and $C6$. Since $C1$ is far away from $C6$ and do not hear each other, these two nodes further perform duplicate forwarding and the paths will further diverge and yield many redundant transmissions. Therefore selecting forwarding nodes just to ensure the packet makes progress is not sufficient and we should also prevent routes from diverging to minimize redundant transmissions. Preventing diverging routes is especially important for supporting multiple simultaneous flows.

To avoid diverging paths, the nodes not on the shortest path use the following method to select their forwarding list. Let i be the node not on the shortest path. Among all nodes on the shortest path, i finds the one that has smallest ETX' to i, and denotes this node as j. i first adds j to its candidate forwarding list if j is closer to the destination than itself. In addition, i adds a node from j's forwarding list (denoted as k) as its own candidate forwarding node if k satisfies the following two conditions: (i) k is closer to the destination than i (i.e., $ETX(i,dest) > ETX(k,dest)$), and (ii) k has good connectivity with i (i.e., $ETX'(i,k) < threshold$). Applying this idea to the example in Figure 3, we observe that even when $B1$ and $B3$ perform duplicate forwarding, since their forwarding lists only have $C3$ and $C4$, the routes do not further diverge, thereby minimizing duplicated forwarding.

2) Forwarding time: A sender sorts the forwarding nodes in an increasing order of their ETX towards the destination. The i-th forwarding node on the list sets its forwarding timer to

$(i-1)*\delta$, where i starts from 1, δ is the time it takes to wait and transmit the data (including the waiting time for all the packets queued before it in the wireless card to finish transmission). We set $\delta = 45\ ms$. It is easy to see the maximum time it takes for the forwarding is $maxForwardTime = \delta * (numForward - 1)$. To limit the delay variance and reduce overhead, we limit the maximum number of forwarding nodes to 5.

D. Local Recovery

When a node transmits a packet, if at least one of the forwarding nodes specified in the header receives the packet, the packet makes progress towards the destination. If none of the forwarding nodes receive the packet, the packet should be retransmitted. SOAR uses hop-by-hop network-layer ACKs to provide reliability. When an ACK is not received within a retransmission timeout, the packet is assumed to be lost and should be retransmitted. Each node retransmits a packet up to $maxRetries$, which is set to 3 in our evaluation. So essentially each sender (including an intermediate forwarder) tries to ensure its transmission is received by at least one node in its forwarding list and retransmits the packet if necessary up to $maxRetries$. So SOAR provides best effort reliability (similar to IEEE 802.11 link-layer retransmissions though at the network layer), and applications that demand end-to-end full reliability can further use upper-layer protocols (e.g., TCP or application-layer support) to provide a reliability guarantee.

SOAR uses selective ACKs to protect against ACK losses; meanwhile it also uses piggyback ACKs and ACK compression to reduce ACK overhead. Below we elaborate each of these schemes.

1) Using selective ACKs to protect ACK losses: Loss of ACKs is costly, since it results in unnecessary retransmissions. To minimize the effect of an ACK losses, we use *selective ACKs* to acknowledge all recently received packets. Using selective ACKs reduces the effect of ACK losses, since even if an ACK for packet i is lost, the subsequent ACKs still convey that packet i is received so that the packet will not be retransmitted unnecessarily.

The selective ACK contains two fields: (i) the starting sequence number of the out of order ACKs ($start$), and (ii) a bit-map of out of order ACKs (Our implementation uses a fixed length bit-map with 256 bits, i.e., 32 bytes). Figure 4 shows how we update the fields. All the packets up to $start$ are assumed to be received, and i-th position in the bitmap is 1 if and only if $start + i$-th packet is received.

One caveat is that it is possible that the difference between the largest and smallest sequence numbers of the lost packets is above 256. One way to handle is to use a variable-size bit-map and let the bit-map grow as large as necessary to accommodate this range of sequence number difference. This is not desirable since it may incur significant overhead due to large ACK size and it is even possible to grow the packet over MTU. Our implementation always limits the size of the bit-map to be within 256 bits as follows. We update $start$ so that the largest received packet is no more than $start + 256$. This implies that we may not have received all packets up to $start$ even though we assume so. The likelihood of such occurrence is low since 256 is quite a large range. Moreover, SOAR is designed to provide best-effort reliability and leaves the upper-layer to ensure full reliability if needed.

```
if(recvPktSeq ≤ start)
  no op;
else if (recvPktSeq = start + 1)
  start = start + 1;
  shift bitmap left by 1;
else if (recvPktSeq < start + 256)
  set bitmap(recvPktSeq-start) to 1;
else
  shift bitmap left by (recvPktSeq − start − 256);
  start = recvPktSeq − 256;
end
```

Fig. 4. Update selective ACK.

2) Using piggyback ACKs and ACK compression to reduce ACK overhead: SOAR uses piggyback ACKs and ACK compression to reduce the overhead of acknowledgements.

More specifically, SOAR piggybacks ACKs to data packets so that each data packet carries information about which set of packets have been recently received. We generate a piggybacked ACK just before the data packet is sent to the wireless card, thereby allowing ACKs to carry the latest information about which packets have been received. Each node that receives the packet checks if any of the packets in its queue have been ACKed by a higher-priority node. If so, it removes the packet from its queue and cancels its forwarding timer for the packet.

Piggyback ACKs are especially effective when there are enough data packets to transmit. When a node does not have much data to send, it should also send stand-alone ACKs to provide timely feedback. Stand-alone ACKs have higher priority than data packets, and are inserted in the front of queue before data packets. Our evaluation shows that sending one ACK for every packet yields significant overhead, and may reduce throughput on a reliable link. To reduce such overhead, we use the following *ACK compression*. We schedule an ACK either when K new data packets have been received or an ACK timer expires. As a further optimization, before the ACK is sent to the wireless card, if another data packet coming from the same flow arrives and is within K packets away from the ACK, we cancel the stand-alone ACK and piggyback the ACK to that data packet. We use ACK timer of 30 ms and $K = 2$ in our evaluation. Our results show that the combination of piggyback ACKs and ACK compression is effective in reducing overhead.

E. Rate Control

To improve network utilization, SOAR uses a sliding-window protocol to allow multiple packets outstanding. The difference in delay between two forwarding nodes with consecutive priorities, denoted as δ, depends on the number of outstanding packets. This is because δ should be set large enough so that the packet transmitted by a higher-priority node still proceeds the transmission from a lower priority node even if the packet from the higher-priority node is at the end of the current sliding window while the packet from the lower-priority node is at the beginning of current sliding

window (*e.g.*, when lower-priority node has low load). In order to achieve low delay, we limit the maximum number of outstanding data packets to 3 in our evaluation. This limit also helps to make data packets carry up-to-date piggyback ACKs. Note that ETX probe packets are not subject to this limit to ensure timely delivery of the control packets.

III. Performance Evaluation

In this section, we present our evaluation methodology and performance results.

A. Evaluation Methodology

Our evaluation is based on 802.11a and 6 Mbps medium bit rate with RTS/CTS disabled. Disabled RTS/CTS is the default setting in real networks. We generate CBR traffic with 6 Mbps data rates, which is high enough to saturate the wireless links. We use the end-to-end goodput (i.e., total number of non-duplicate received bits per second) over all flows as the performance metric.

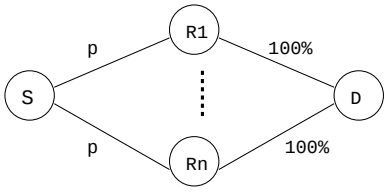


Fig. 5. Diamond topology.

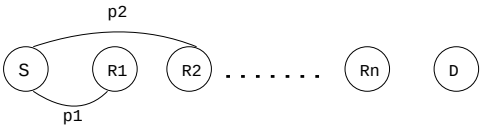


Fig. 6. Linear chain topology.

1) Network topologies: Our evaluation uses various network topologies. We use diamond topologies, as shown in Figure 5, to evaluate the capability of SOAR in combining multiple weak links into a stronger link. We use linear-chain topologies, as shown in Figure 6, to evaluate the effectiveness of SOAR in opportunistically taking advantage of lucky long transmissions. Finally we use grid topologies to evaluate the performance of SOAR in more general topologies. We add a packet dropping module at the MAC layer to introduce controllable packet losses. The final link loss rate includes both injected packet losses and packet collisions.

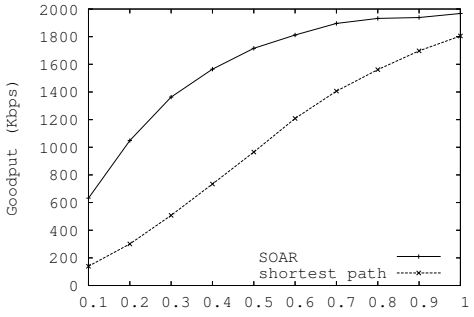
2) Baseline comparison: We use the *ETX-based shortest-path routing protocol* as a baseline comparison. We extend DSDV in NS-2 (version 2.29) to support the ETX routing metric as follows. Each node sends one broadcast probe per second, and propagates the updated loss rates on all its links to the rest of the network. Based on this information, each node computes a shortest path to the destination using $\frac{1}{(1-p_f)(1-p_r)}$ as a link weight, where p_f and p_r are forward and reverse link loss rates.

B. Evaluation Results

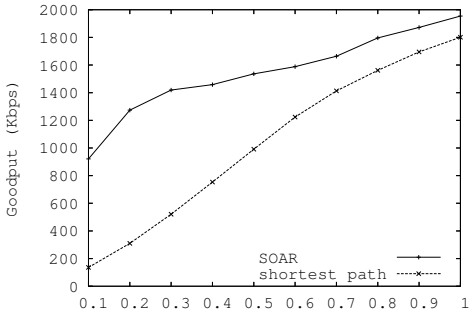
First we present performance results of a single flow over diamond, linear-chain, and grid topologies. Then we evaluate the performance of multiple flows.

1) Diamond topologies: Figure 7 compares the goodput of SOAR and shortest path routing using diamond topologies, as shown in Figure 5. The delivery rate from the source to each intermediate node (denoted as p) is varied from 0.1 to 1, and all the other links (including the links from the intermediate nodes to the source) have delivery rate of 1.

We make the following observation. First, in all cases SOAR outperforms the shortest path routing. The goodput improvement ranges from 10.2% to 366.9% for 2 intermediate nodes and ranges from 8% to 578.7% for 5 intermediate nodes. Second, the percentage-wise improvement is largest when p is small. This is because the effect of combining multiple weak links into a strong link is larger for weak links.



(a) 2 intermediate nodes



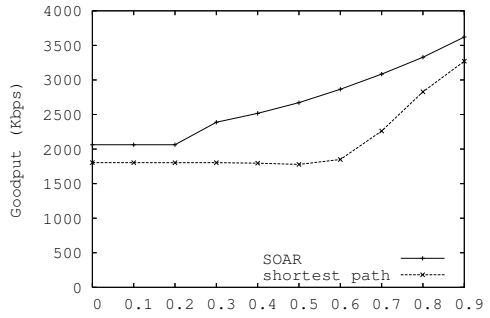
(b) 5 intermediate nodes

Fig. 7. Diamond topologies: vary the delivery rate from the source to each intermediate node from 0 to 1 and fix the delivery rate of all other links to 1.

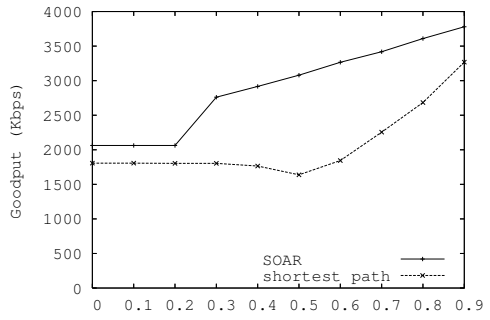
2) Linear-chain topologies: Next we use linear-chain topologies to evaluate the effectiveness of SOAR in leveraging lucky long transmissions. Let $p1$ denote the delivery rate of one-hop links, and $p2$ denote the delivery rate of two-hop links. In the case of asymmetric loss rates, $p1$ and $p2$ are both the delivery rates of links in the forward direction, and the delivery rates of the reverse direction are 1. In the case of symmetric loss rates, the delivery rates in forward and reverse directions are both $\sqrt{p1}$ for one-hop links, and are both $\sqrt{p2}$ for two-hop links.

Figure 8 compares SOAR with the shortest path routing in two-hop linear chain topologies, where $p1$ is 1 and $p2$ varies from 0 to 0.9. We observe SOAR significantly out-performs the shortest path routing. The improvement is largest when $p2$ has moderate delivery rate (around 0.5). This is because when $p2$ is too low, there are few lucky long transmissions for SOAR to take advantage of, and when $p2$ is too high, the shortest

path routing also utilizes these long transmissions since the ETX value over a direct two-hop link is lower than the sum of ETX values over two one-hop links (i.e., $1/p2 < 2/p1$). In comparison, for a moderate $p2$, there are significant lucky long transmissions, and the shortest path routing does not take advantage of them.



(a) Asymmetric losses



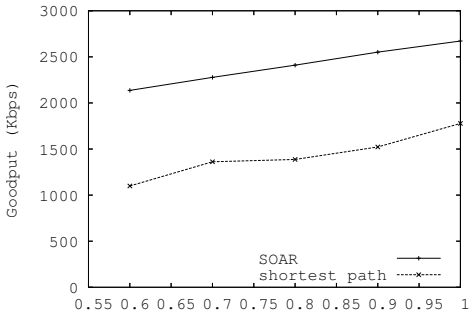
(b) Symmetric losses

Fig. 8. Two-hop linear chain topologies: $p1$ is 1 and $p2$ varies from 0 to 0.9.

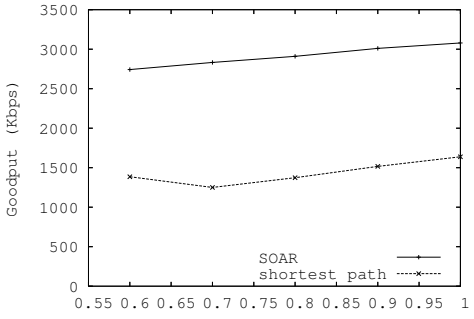
Figure 9 evaluates SOAR using two-hop linear chain topologies, where $p1$ varies from 0.6 to 1 and $p2$ is 0.5. As it shows, SOAR improves goodput by 50.3% - 94.4% under asymmetric losses, and by 88.0% - 126.6% under symmetric losses. The larger improvement in symmetric losses is because $p2$ is the product of forward and reverse delivery rates, and for the same value of $p2$, the delivery rate in the forward direction is higher under symmetric losses than asymmetric losses, thereby providing more opportunities for packets to make progress beyond one hop.

Figure 10 compares SOAR with the shortest path routing under a varying number of hops between the source and destination. Again SOAR consistently out-performs shortest path routing.

3) Grid topologies: Now we compare SOAR with the shortest-path routing using grid topologies. Figure 11 shows a 3*3 grid, where the delivery rate of solid links (i.e., links over one vertical/horizontal hop) is $p1$ and delivery rate of dashed links (i.e., links over two vertical/horizontal hops or one diagonal hop) is $p2$. Figure 12 shows the goodput of a flow from node 1 to node 9 in Figure 11 with varying $p2$, and Figure 13 shows the goodput of the same flow in 3*3 grid topologies with varying $p1$. We make the following observations. First, SOAR can yield significant improvement. The improvement is largest when $p2$ is around 0.5, since SOAR can leverage a

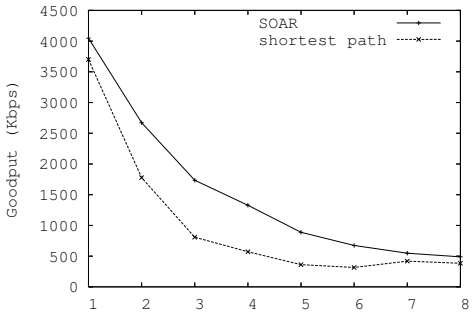


(a) Asymmetric losses

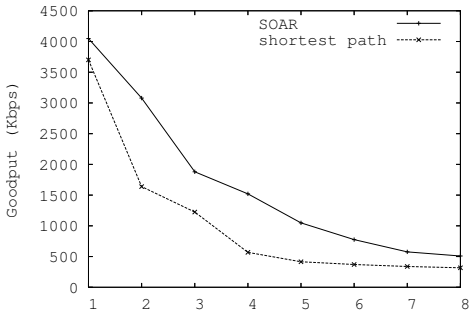


(b) Symmetric losses

Fig. 9. Two-hop linear chain topologies: $p2$ is 0.5, and $p1$ varies from 0.6 to 1.



(a) Asymmetric losses



(b) Symmetric losses

Fig. 10. Linear chain topologies: $p1 = 1$, $p2 = 0.5$, and the number of hops between the source and destination varies from 1 to 8.

significant number of lucky transmissions, which the shortest path routing cannot. Second, we observe the shortest path routing has a dip in its goodput. A closer look reveals that it is because when there are multiple paths of similar ETX, there is route flapping (at some time, one path is slightly better than the other; at another time, the other path is slightly better). In comparison, SOAR simultaneously utilizes multiple paths and its performance is more stable.

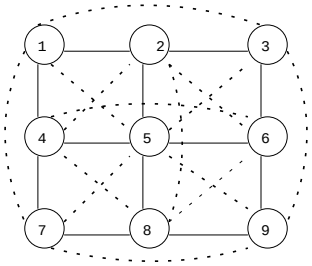


Fig. 11. 3*3 grid topologies

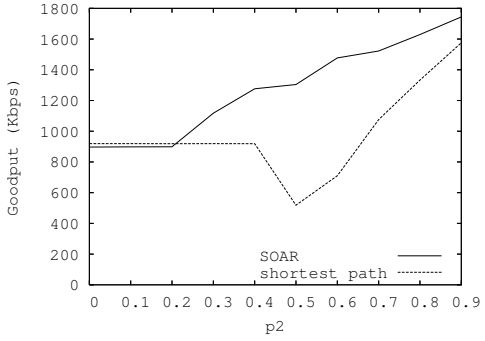


Fig. 12. 3*3 grid topologies: $p1$ is 1, and $p2$ varies from 0 to 0.9.

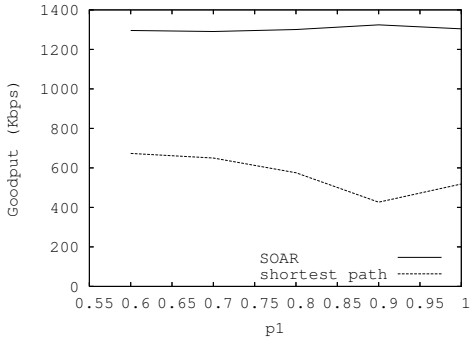
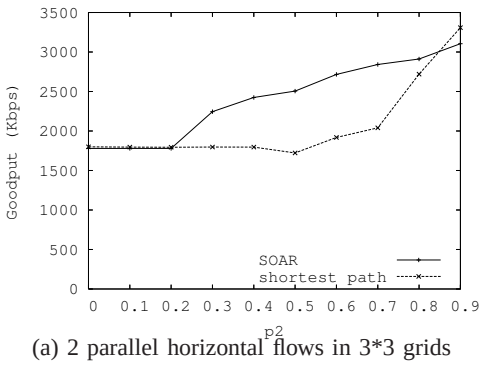


Fig. 13. 3*3 grid topologies: $p1$ varies from 0.6 to 1, and $p2$ is 0.5.

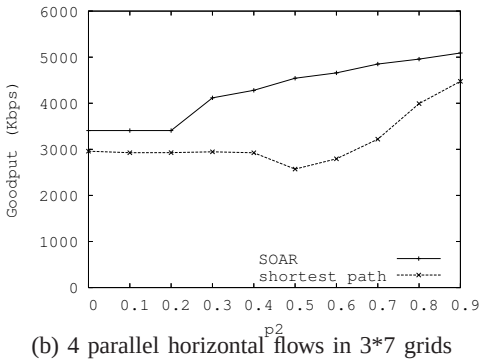
4) Multiple flows: So far we consider the performance of a single-flow. Next we evaluate the performance of multiple flows. Figure 14 shows the goodput of 2 and 4 parallel flows in the grid topologies. We observe that SOAR significantly out-performs shortest path routing. The improvement is largest when $p2$ is around 0.5 since SOAR leverages lucky long hops, whereas the shortest path routing does not take advantage of.

IV. Conclusion

In this paper, we develop a novel opportunistic routing protocol, called SOAR. It uses priority-based forwarding and



(a) 2 parallel horizontal flows in 3*3 grids



(b) 4 parallel horizontal flows in 3*7 grids

Fig. 14. Grid topologies: total goodput of parallel flows, where $p1$ is 1 and $p2$ varies from 0 to 0.9.

judicious forwarding node selection to maximize the progress of each transmission with little coordination among the nodes. It protects against packet losses using local recovery based on selective ACKs, which are sent using either piggyback or ACK compression. Our preliminary results show that SOAR achieves significant improvement over traditional routing and supports multiple simultaneous flows. To the best of our knowledge, it is the first opportunistic routing protocol that supports multiple flows. We are currently implementing SOAR in our wireless testbed so that we can realize the potential of opportunistic routing in real networks.

References

[1] S. Biswas and R. Morris. ExOR: Opportunistic multi-hop routing for wireless networks. In *Proc. of ACM SIGCOMM*, Aug. 2005.

[2] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MOBICOM*, Sept. 2003.

[3] R. Draves, J. Padhye, and B. Zill. Comparison of routing metrics for multi-hop wireless networks. In *Proc. of ACM SIGCOMM*, Aug. 2004.

[4] S. Jain and S. Das. Exploiting path diversity in the link layer in wireless ad hoc networks. In *Proc. of the 6th IEEE WoWMoM Symposium*, Jun. 2005.

[5] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, 2001.

[6] The network simulator – ns-2. http://www.isi.edu/nsnam/ns/.

[7] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of ACM SIGCOMM*, 1994.

[8] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.