SOAR: Simple Opportunistic Adaptive Routing Protocol for Wireless Mesh Networks

Eric Rozner
University of Texas at Austin
erozner@cs.utexas.edu

Jayesh Seshadri
VMware
jseshadr@vmware.com

Yogita Ashok Mehta
Google
yogitamehta@gmail.com

Lili Qiu
University of Texas at Austin
lili@cs.utexas.edu

Abstract—Multihop wireless mesh networks are becoming a new attractive communication paradigm owing to their low cost and ease of deployment. Routing protocols are critical to the performance and reliability of wireless mesh networks. Traditional routing protocols send traffic along predetermined paths and face difficulties in coping with unreliable and unpredictable wireless medium. In this paper, we propose a Simple Opportunistic Adaptive Routing protocol (SOAR) to explicitly support multiple simultaneous flows in wireless mesh networks. SOAR incorporates the following four major components to achieve high throughput and fairness: (i) adaptive forwarding path selection to leverage path diversity while minimizing duplicate transmissions, (ii) priority timer-based forwarding to let only the best forwarding node forward the packet, (iii) local loss recovery to efficiently detect and retransmit lost packets, and (iv) adaptive rate control to determine an appropriate sending rate according to the current network conditions. We implement SOAR in both NS-2 simulation and an 18-node wireless mesh testbed. Our extensive evaluation shows that SOAR significantly outperforms traditional routing and a seminal opportunistic routing protocol, ExOR, under a wide range of scenarios.

Index Terms—C.2.1.k [Communication/Networking and Information Technology]: Network Architecture and Design – Wireless communication; C.2.2.d [Communication/Networking and Information Technology]: Network Protocols – routing protocols.

1 Introduction

Multihop wireless mesh networks are becoming a new attractive communication paradigm. Many cities across the world have deployed or are planning to deploy them [34], [33], [30] to provide Internet access to residents and local businesses. Routing protocol design is critical to the performance and reliability of wireless mesh networks.

A natural approach to routing traffic in wireless mesh networks is to adopt techniques similar to those in wire-line networks, which select a best path for each source-destination pair (according to some metric) and send traffic along the pre-determined path. Most of the existing routing protocols, such as DSR [16], AODV [26], DSDV [25], and LQSR [8], fall into this category, which we refer to as traditional routing.

This research is supported in part by by National Science Foundation grants CNS-0546755 and CNS-0627020. We thank Szymon Chachulski, Michael Jennings, Sachin Katti and Dina Katabi for providing ExOR source code. Jayesh Seshadri and Yogita Ashok Mehta worked on this project while they were students at University of Texas at Austin.

Recent studies [2], [3], [35] show that traditional routing faces difficulties in coping with unreliable and unpredictable wireless medium. Motivated by these observations, researchers [2], [3], [35] developed opportunistic routing protocols for wireless mesh networks. Opportunistic routing exploits the broadcast nature of the wireless medium and does not commit to a particular route before data transmission. Instead, the sender broadcasts its data; among the nodes that hear the transmission, the one closest to the destination is selected to forward the data. In this way, opportunistic routing can effectively combine multiple weak links into a strong link and take advantage of transmissions that reach unexpectedly near or unexpectedly far.

In this paper, we present the design and implementation of a Simple Opportunistic Adaptive Routing protocol (SOAR), a new addition to the opportunistic routing protocol design space. Different from the existing opportunistic routing protocols, SOAR explicitly supports multiple simultaneous flows by strategically selecting forwarding nodes and employing adaptive rate control. To demonstrate its effectiveness and feasibility, we implement SOAR in both the NS-2 simulator [23] and an 18-node wireless testbed. Using extensive evaluation, we show that SOAR can significantly out-perform traditional routing and a seminal opportunistic routing protocol, ExOR [2], under a wide range of scenarios.

The rest of the paper is organized as follows. In Section 2, we survey related work and motivate opportunistic routing. In Section 3, we describe design challenges of opportunistic routing and present the SOAR routing protocol. We evaluate SOAR using NS-2 simulations in Section 4. We present testbed implementation and evaluation in Section 5. Finally, we conclude in Section 6.

2 Background

In this section, we first review traditional routing protocols for wireless networks. Then, we explain the potential benefits of opportunistic routing and introduce existing opportunistic routing protocols.

2.1 Traditional Routing Protocols

Routing has been an active area in wireless networking research. Most of the original work in this area targeted high-mobility scenarios such as battlefield networks. Therefore, the focus was on establishing and maintaining routes under frequent and unpredictable changes in network connectivity. A number of on-demand routing protocols have been proposed for this purpose, as exemplified by DSR [16] and AODV [26], where packets are routed along paths with the shortest hop count.

Recently, wireless mesh networks [17], [28], [30] have emerged as a new dominant application of multihop wireless networks. Nodes in such networks have little or no mobility and often are not constrained by short battery-life or limited computational power. Therefore, improving network performance becomes the primary focus. Researchers have found that the hop-count metric, as used in DSR and AODV, does not provide good performance since not all hops are equal. To address this issue, various link-quality metrics have been proposed [1], [7], [9], [10], [12], [13]. These metrics quantify the quality of links using link loss rate, packet transmission time, or signal-to-noise ratios. These works are complementary to opportunistic routing by offering metrics for comparing different routes. As with other opportunistic routing protocols, SOAR uses ETX [7] as the underlying routing metric, but it is easy for SOAR to support any alternative routing metric.

In addition to routing metric design, there is another thread of works that optimize routing in wireless mesh networks by casting it as a linear program (*e.g.*, [14], [31]) or non-linear program (*e.g.*, [21]).

2.2 Benefits of Opportunistic Routing

More 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, it can combine multiple weak links into one strong link. Second, it takes advantage of unexpectedly short or unexpectedly long transmissions. We further illustrate the benefits using the following two examples.

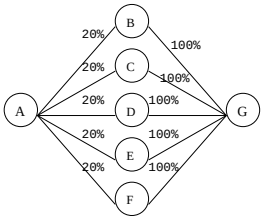


Fig. 1. Opportunistic routing can take advantage of multiple weak links.

As shown in Figure 1, a source has weak wireless connectivity to each of the five intermediate nodes, with a delivery rate of 20%. For ease of illustration, assume independent loss rates on each link (This assumption is not required for our protocol). All the intermediate nodes 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. Therefore, on average, a packet needs to be transmitted five times in order to reach the next hop. Then, the packet needs to be forwarded once to reach the destination. Altogether, six transmissions are required to deliver the packet end-to-end. In comparison, in 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 reach at least one of the five intermediate nodes, and another transmission is required for an intermediate node to forward. Therefore, opportunistic routing achieves 2.5 times the throughput of traditional routing.

Fig. 2. Opportunistic routing can maximize the progress each transmission makes.

Second, a traditional routing protocol has to trade off between link quality and the amount of progress each transmission makes. For example, consider the network shown in Figure 2, A sends data to D along the path $A-B-C-D$. If B is used as the next hop and the quality of link $A-B$ is good, then no retransmissions are 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 if 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.

2.3 Prior Opportunistic Routing Protocols

ExOR [2] is a seminal opportunistic routing protocol. In ExOR, senders broadcast 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 of each transmission, the forwarding nodes relay data packets in the order of their proximity to the destination (Throughout the rest of this paper, proximity is measured by the ETX metric [7], not in terms of physical distance). To minimize redundant transmissions, ExOR uses a batch map, which records the list of packets each node has received; every forwarding node only forwards data that has not been acknowledged by the nodes closer to the destination. ExOR imposes strict timing constraints among the forwarders to facilitate

coordination in the relay process. Only one forwarder can be active at a time, and spatial reuse of the wireless spectrum is reduced. Furthermore, ExOR's forwarding paths can easily diverge, *i.e.*, nodes on the different forwarding paths may not hear from each other and cause duplicate forwarding. These difficulties make it unclear how well ExOR supports multiple simultaneous flows.

MORE [3] applies network coding to opportunistic routing in a clever way. Since random coding can effectively generate linearly independent coded packets with a high probability, the forwarding nodes in MORE do not need to coordinate which packets are forwarded by which nodes. However, MORE selects forwarding nodes in a similar way as ExOR and does not prevent diverging paths, which can lead to waste of resource usage. Furthermore, it does not rate-limit the initial transmission of packet batches, and may cause degradation in aggregate network performance and fairness. We believe that the prevention of diverging paths and rate-limiting are necessary for efficiently supporting multiple flows under opportunistic routing protocols.

ROMER [35], another opportunistic routing protocol, tries to forward the packets simultaneously along multiple paths. It incorporates a credit based scheme to limit the number of transmissions that a packet is allowed before reaching the destination. Even with the credit-based scheme, there is still significant overhead since a packet is allowed to be forwarded by multiple nodes at each hop. Also, setting the credit is non-trivial and static credit has difficulties in coping with different topologies. As a result, ROMER only supports a single flow.

Zhong *et al.* [36] show that the routing metric used to select and prioritize forwarding nodes is important. They develop a new routing metric, called EAX, to account for inter-candidate communication in opportunistic routing.

In addition to opportunistic routing protocols for mesh networks, researchers have also designed opportunistic routing protocols for ad-hoc and sensor networks. For example, [5] and [32] both dynamically select forwarding nodes based on recent link quality. However, in both protocols, only one forwarding node is selected before transmissions, and they cannot take advantage of transmissions reaching nodes other than the previously selected forwarder. [4] balances the energy consumption rates of different nodes in a sensor network by opportunistically incorporating forwarders' energy consumption.

Our previous workshop paper [29] presents an initial design of an opportunistic routing protocol and preliminary simulation results in some toy topologies. This paper is built on our previous work, but differs in the following important ways. First, we make several significant new optimizations, which include (i) developing a rate control scheme for opportunistic routing, which adapts the sending rate according to link loss rate; (ii) adapting the number of forwarding nodes according to link quality, (iii) adapting retransmission timeout according to network delay, and (iv) cross-flow ACKs to increase the efficiency of receiver feedback in the presence of multiple flows. These optimizations are critical to ensure SOAR performs well in a wide range of scenarios. Second, we extensively evaluate the performance of SOAR by simulating in a large range of synthetic topologies and traffic demands. Third, we implement SOAR in a wireless testbed, and demonstrate its feasibility and effectiveness in a real network. In doing so, we add comparisons against ExOR, a seminal opportunistic routing protocol.

3 The SOAR Routing Protocol

In this section, we first describe design challenges of opportunistic routing protocols. Then we present an overview and the protocol details of SOAR.

3.1 Design Challenges

The goal of opportunistic routing is to maximize the progress each transmission makes without causing duplicate (re)transmissions or incurring significant coordination overhead. In order to achieve this goal, several important design issues should be addressed:

Forwarding node selection: While opportunistic routing defers the final route selection after data transmissions, the candidate forwarding nodes should still be selected in advance. This is necessary because the number of duplicate transmissions and coordination overhead tend to increase with the number of forwarding nodes. Without judicious forwarding node selection, the overhead of opportunistic routing might offset its benefits.

Avoid duplicate transmissions: When multiple nodes overhear a transmission, we want to ensure that only the node closest to the destination forwards it. The best forwarding node should be selected in a cheap and distributed way.

Loss recovery: In opportunistic routing, each node broadcasts data packets, and broadcast packets are vulnerable to packet losses and corruption since the MAC layer offers no reliability support for broadcast. Therefore it is important for opportunistic routing protocols to efficiently detect and recover packet losses.

Rate control: Determining an appropriate sending rate is important for opportunistic routing. Without rate control, a flow may send too many packets on the first few hops which cannot be forwarded on the subsequent hops. Due to wireless interference, such transmissions take away available bandwidth from the subsequent hops and significantly degrade performance [20].

To address the above challenges, SOAR consists of the following four major components: (i) adaptive forwarding path selection to leverage path diversity while avoiding diverging paths, (ii) priority timer-based forwarding to allow only the best forwarding node to forward the packet, (iii) local loss recovery to efficiently detect and retransmit lost packets, and (iv) adaptive rate control to determine an appropriate sending rate according to the current network condition.

3.2 Overview

As ExOR and MORE, SOAR is a proactive link state routing protocol. Every node periodically measures and disseminates link quality in terms of ETX. Based on this information, a sender selects the default path and a list of (next-hop) forwarding nodes that are eligible for forwarding the data. It then broadcasts a data packet including this information. 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 forwarding timers based on their proximity to the destination. A node closer to the destination uses a smaller timer and forwards the packet earlier. Upon hearing this transmission, other nodes will remove the corresponding packet from their queues to avoid duplicate transmissions. Like all the existing opportunistic routing protocols, SOAR broadcasts data packets at a fixed PHY data rate. How to perform automatic rate adaptation in opportunistic routing is an interesting topic by itself, and we plan to investigate as part of our future work.

3.3 Adaptive Forwarding Path Selection

To support opportunistic routing, each node maintains a routing table of the following format: (destination, default path, forwardList), where the default path is the shortest path from the current node to the destination and the forwarding list includes a list of next-hop nodes that are eligible to forward the transmission.

3.3.1 Default Path Selection

In order to compute the default path and forwarding lists, every node measures and maintains the network topology, as in existing wireless mesh routing protocols (*e.g.*, Srcr [7], LQSR [8], and ExOR [2]). Several routing metrics have been proposed in the literature to assign link weights based on link quality. We use the ETX metric, a state-of-art routing metric proposed by De Couto *et al.* [7]. A link's ETX metric measures the expected number of transmissions (including retransmissions) required to reliably send a packet across the link. Let p_f and p_r denote the loss probabilities of the link in the forward and reverse directions, respectively. Each node measures the 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 default path is the shortest path between the source and destination in terms of ETX. The ETX routing metric has been shown effective in selecting good quality routes [7], [8].

3.3.2 Forwarding Node Selection

Forwarding node selection is critical to the performance of SOAR. In order to leverage path diversity while avoiding duplicate transmissions, SOAR relaxes the actual route that data traverses to be along or near the default path. Different from traditional routing, SOAR leverages path diversity by using more flexible routes: nodes other than the next hop can forward the data. Differing from existing opportunistic routing protocols, SOAR constrains the nodes involved in routing a packet to be near the default path, as shown in Figure 3. This prevents routes from diverging and minimizes duplicate transmissions. Moreover, this forwarding node selection also simplifies coordination since all the nodes involved are close to nodes on the default path and can hear each other with a reasonably high probability. Therefore, we can use overheard transmissions to coordinate between forwarding nodes in a cheap and distributed way.

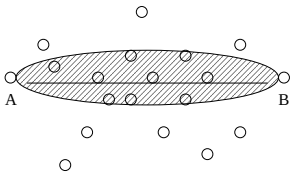


Fig. 3. The actual route in SOAR involves nodes on or near the default path. In the figure, the nodes in the shaded region participate in forwarding packets from A to B.

Our forwarding list selection algorithm consists of two steps. First, a sender selects an initial forwarding list based on the default path. Then it further limits the number of forwarding nodes to minimize duplicate transmissions. These steps are taken by a sender on each packet, allowing for the forwarding list to quickly adapt to network conditions. Also, a sender in this case refers to either a flow source or a forwarder within the flow (*i.e.*, the forwarding list is updated at each network hop).

Selecting initial forwarding lists: When node i is on the default path, i selects the forwarding nodes that satisfy the following conditions:

(C1) The forwarding node's ETX to the destination is lower than i's ETX to the destination.

(C2) The forwarding node's ETX to i is within a threshold.

The first constraint ensures that the packet makes progress. The second constraint ensures that i hears the forwarding node's transmissions with a high probability to avoid duplicate retransmissions.

Selecting the correct threshold is nontrivial and using a constant threshold can be ineffective. Consider a 4-node network, where there is a flow from node A to D with a default path $A-B-D$. Whether node C should be included as a forwarding node not only depends on the absolute delivery rates of links $A-C$ and $C-D$, but also depends on how they compare with the delivery rate of links on the default paths. For example, suppose $A-B$, $B-D$, and $C-D$ all have 50% delivery rate. If $A-C$ has delivery rate of 5%, it is not helpful to include C as a forwarding node since in most cases when a packet does not reach B it does not reach C either. If the delivery rate of $A-B$ changes to 5%, now it makes sense to

include C as a forwarding node. This example suggests that we should adapt the threshold according to the link to the default next hop. So we set the threshold to $\gamma \times ETX(i, nexthop)$ and use $\gamma = 4.0$ in our evaluation. The intuition is that the links among the forwarding nodes and between the forwarding nodes and i should not be too much worse than the link between i and $nexthop$ (We also try other values of γ between 2 and 6, and they do not cause significant performance difference).

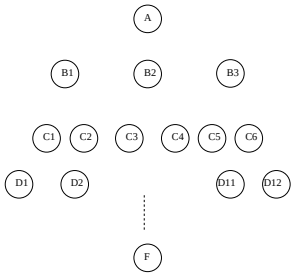


Fig. 4. Careful forwarding node selection is necessary to prevent routes from diverging.

SOAR also allows nodes not on the default path to forward packets. These nodes could select their forwarding nodes in the same way as the nodes on the default path. However, this could potentially result in diverging forwarding paths and duplicate transmissions. For example, in Figure 4 node A wants to send traffic to node F. A selects $B1$, $B2$, and $B3$; $B1$ picks $C1$ and $C2$, while $B3$ selects $C5$ and $C6$ as forwarding nodes. Then if $B1$ and $B3$ do not hear each other's forwarding (since the link between them has non-zero loss rate), a copy of the packet is forwarded to $C1$, $C2$, $C5$, and $C6$. Since $C1$ and $C6$ cannot hear each other, the packet will further diverge and yield redundant transmissions. Therefore not only should we ensure that forwarding nodes make progress and have sufficiently good link quality from node i, but also we want the selected forwarding nodes to be adjacent to the default path and every pair of forwarding nodes has sufficiently good link quality between them to avoid diverging paths. This leads to the following two additional constraints in selecting forwarding nodes:

(C3) Each forwarding node is close to at least one node on the default path (*i.e.*, with ETX below a threshold).

(C4) The ETX of a link between any pair of forwarding nodes is within a threshold.

These constraints ensure forwarding nodes have good connectivity amongst themselves and to nodes on the default path. To achieve this, we filter out the nodes that do not satisfy **(C3)**, and then successively add the remaining nodes to the forwarding list in an increasing order of ETX to the destination, continuously ensuring that **(C4)** is satisfied. In Section 5.4.1, we found that adjacent forwarders in ExOR do not always satisfy **(C4)**, which can lead to a partition amongst the forwarding nodes.

Further pruning forwarding nodes: To further reduce coordination overhead and duplicate traffic, we bound the maximum number of forwarding nodes within a specified threshold (Our evaluation uses 5 as the upper bound). Meanwhile we may not need to include all the nodes if the nodes closest to the destination are sufficiently reliable. For example, if the links from A to all five neighbors are 100% in Figure 1, we only need to use one of the neighbors as the forwarding node. This observation suggests that we should only include enough forwarding nodes (in an increasing order of their ETX to the destination) to bound the loss rate of the $virtual\ link$, which consists of the physical links between the sender and all the forwarding nodes selected so far.

Therefore we start from the forwarding node closest to the destination and iteratively add a node to the final forwarding list until either the loss of a virtual link is below a threshold or the specified maximum number of forwarding nodes is reached. We add forwarding nodes in an increasing order of their ETX to the destination because only when the nodes closest to the destination have a sufficiently reliable virtual link can we prune the remaining forwarding nodes. In the other order, even when the forwarding nodes included so far have sufficiently reliable links, there is still a benefit of including additional forwarding nodes if they are closer to the destination than those included so far. Finally, if the loss rate of the virtual link is higher than the loss threshold, we replace the last forwarding node added so far with the forwarding node that has lowest ETX to the current node. This reduces the virtual link loss rate and increases the chance that the packet makes progress beyond the current node.

3.4 Priority-based Forwarding

We use priority-based forwarding to maximize the progress each transmission makes. A sender transmits a packet, which specifies a list of forwarding nodes sorted in a non-decreasing order of ETX towards the destination. The forwarding list and its priority are specified on a per packet basis. This makes it easy to adapt to topology changes – upon topology changes, a sender can simply specify a different set of forwarding nodes in a different order, and this change takes effect immediately.

Each node hearing the packet first checks if it is included in the forwarding list. If not, it discards the packet. Otherwise, it sets its forwarding timer as follows. The i-th forwarding node on the list sets its forwarding timer to $(i-1) * \delta$, where i starts from 1. In this way, 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 queue, thereby avoiding duplicate forwarding.

In order for priority-based forwarding to work well, δ should be large enough to ensure that the transmission from a higher priority node precedes that from a lower priority node even under variable queueing and contention delay. To achieve this, we limit the maximum

number of packets queued on the wireless card to 3. We use $\delta = 45ms$, which is appropriate for bulk transfer, targeted by all opportunistic routing protocols, including SOAR.

3.5 Local Recovery

SOAR uses per-hop network-layer ACKs and retransmissions to provide best effort reliability. Each node on the forwarding path will keep retransmitting a packet until either the packet is acknowledged by a node closer to the destination or the maximum retry count is reached. As with any ACK-retransmission scheme, we need to address the following issues in our design: (i) when to send ACKs, (ii) what information is contained in ACKs, and (iii) when to retransmit. We will describe our approach to each of these issues.

3.5.1 When to send ACKs?

Sending one ACK for every packet yields significant overhead, and may reduce throughput over a reliable link. To reduce the overhead of ACKs, SOAR uses a combination of *piggyback ACKs* and *ACK compression* as follows.

A node piggybacks its ACK to a data packet whenever possible. Moreover, an ACK piggyback occurs just before the data packet is sent to the wireless card to allow the ACK to carry the latest information about which packets have been received. A node considers a packet as received if it receives the packet itself or it hears an ACK for this packet from a node closer to the destination than itself.

Piggyback ACKs are 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. To reduce the overhead of ACKs, SOAR uses ACK compression, where an ACK is scheduled either when K data packets have been received or an ACK timer expires. As a further optimization, before a stand-alone ACK is sent to the wireless card, if another data packet coming from the same flow arrives and is within P packets away from the ACK, we cancel the stand-alone ACK since the data packet can carry the ACK. Our evaluation uses an ACK timer of 30 ms, $K = 10$, and $P = 2$. Our results show that the combination of piggyback ACKs and ACK compression is effective in reducing feedback overhead.

3.5.2 What to send in ACKs?

SOAR uses cumulative/selective ACKs to minimize the impact of ACK loss. The ACKs for each flow contain two fields: (i) the starting sequence number of the out of order ACKs ($start$) and (ii) a bit-map of out of order ACKs. (We use a fixed length bit-map with 256 bits). 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. We update $start$ so that the largest received packet is no more than $start + 256$. This implies that it is possible that a node may not have received all packets up to $start$ even though it assumes so. The likelihood of such occurrence is low since the bit-map size of 256 bits is generally large enough. Moreover, SOAR is designed to provide best-effort reliability and leaves the upper-layer to ensure full reliability if needed.

When there are multiple flows, a simple approach is to ACK each flow separately. However this would have higher ACK delay and potentially result in unnecessary retransmissions. To address this issue, SOAR uses *cross-flow ACKs*. Whenever a node sends an ACK (regardless of whether it is a piggyback ACK or stand-alone ACK), it includes not only the ACK for the current flow but also ACKs for some other flows.

Now the questions are how many flows to ACK each time and which flows to include in the ACK. We use the following constraints to limit the size of cross-flow ACKs. (i) The final packet size (including payload) is within MTU. (ii) The maximum number of flows to be ACKed is no more than a specified number of flows. The limit is set to 4, including the current flow, in our evaluation. (iii) Senders only include an ACK for another flow when that flow has new packets to be acknowledged since the last transmission of an ACK for that flow. The last constraint not only reduces cross-flow ACK overhead in the normal situation, but also automatically times out flows that become inactive or change route (We use the last constraint to conservatively send cross-flow ACKs. When the previous ACK for flow i is lost, the cross-flow ACK does not include an ACK for flow i, since flow i will generate its own ACKs anyway). When picking which flows to ACK, SOAR favors the flows that have the largest number of new packets to acknowledge since their last ACKs.

3.5.3 When to retransmit data?

In SOAR, each node estimates its retransmission timeout (RTO) in a similar way as done in TCP. Specifically, for every packet that has not been retransmitted, a node measures the time difference between when the packet is transmitted and when the corresponding ACK (in SOAR) is received. Let T denote the measured round-trip time of the current packet over a virtual link. Then the node updates its RTO as shown in Figure 5. RTO is initialized to 30 ms. Our evaluation uses $K = 4$, $\alpha = 1/8$, and $\beta = 1/4$ as in TCP [24].

```
if (T is the first RTT measurement)
  SRTT = T;
  RTTVAR = T/2;
  RTO = SRTT + K*RTTVAR;
else
  RTTVAR = (1 − β) × RTTVAR + β × |SRTT − T|;
  SRTT = (1 − α) × SRTT + α × T;
  RTO = SRTT + K ∗ RTTVAR;
end
```

Fig. 5. Estimation of RTO.

As in TCP, after each retransmission, the timeout of the packet is increased exponentially. We update RTO as $RTO(i) = RTO(i-1) * 1.5$ where $RTO(i)$ is the timeout

of i-th retransmission. We use 1.5 instead of a commonly used 2 to reduce the retransmission time, while quickly adapting to the current network condition.

3.6 Rate Control

Another important design issue is how fast a flow source should transmit data. For both a single flow and multiple simultaneous flows, not controlling the senders' data rates can severely degrade network throughput when they send more than what the path can support. This occurs because any additional traffic reduces the capacity of the bottleneck links due to wireless interference. For multiple simultaneous flows, rate control is also useful to improve fairness and avoid starvation, as shown in [11], [27], [19]. While traditionally rate control is typically considered the responsibility of a transport layer protocol, previous works [11], [27], [19] have shown that the existing congestion control at the transport layer is not effective for multihop wireless networks. A joint optimization of rate limiting and routing is more promising. This is also the approach taken by SOAR. In SOAR each flow uses end-to-end ACKs from its destination node to control its source's sending rate.

Sending end-to-end ACKs: A destination sends an end-to-end ACK after receiving every K unique packets or when a timer expires. The format of an end-to-end ACK is similar to a per-hop ACK in Section 3.5.2. The only difference is an end-to-end ACK contains an additional $totalReceived$ field, indicating the cumulative number of unique packets the destination has received so far. This explicit counter helps the source to determine an appropriate sending rate and is more accurate than using an ACK bit-map.

Different from per-hop ACKs, end-to-end ACKs are forwarded beyond the immediate neighbors. Since end-to-end ACKs can traverse multiple hops, they are sent less frequently than per-hop ACKs. Our evaluation uses $K = 20$ and a timeout of 100ms. To reduce the delay in transmitting an end-to-end ACK, the flow destination transmits end-to-end ACKs along the shortest path via MAC-layer unicast to the flow source.

Updating sending rate: In SOAR, a flow source has a limit on the maximum number of packets sent during the current time interval, denoted as $Window$. During transfers, the sender uses the $totalReceived$ field in the end-to-end ACKs from its destination to update $Window$ at the beginning of each time interval. As shown in Figure 6, when the number of packets acknowledged in the previous interval ($ACKed(i-1)$) is smaller than the minimum window (Min), $Window$ is set to Min. Otherwise, $Window$ is set to $ACKed(i-1)$ plus a small increment ($Incr$). The rationale is that the network can support at least $ACKed(i-1)$ since this is the number of packets successfully delivered in the previous interval, and we increment it by $Incr$ to allow the rate to grow when the network condition improves. Finally, when a sender is unable to send as many packets as were acknowledged in the previous interval (due to MAC scheduling), a $Credit$ is accrued to $Window$ in the successive interval. Figure 6 shows the pseudo-code. While many rate-control solutions are possible, we choose this for its simplicity and effectiveness.

```
In Interval i:
if (ACKed(i − 1) ≤ Min)
  Window(i) = Min;
else
  Window(i) = ACKed(i − 1) + Incr ;

if (Sent(i − 1) < ACKed(i − 2))
  Credit(i − 1) = ACKed(i − 2) - Sent(i − 1);
else
  Credit(i − 1) = 0;

Window(i) = Window(i) + Credit(i − 1);
```

Fig. 6. SOAR rate control algorithm. (Our evaluation uses 200 ms intervals, $Min = 1$, and $Incr = 1$.)

4 Simulation Results

We evaluate SOAR using NS-2 simulations and testbed experiments. The former lets us evaluate under a broader range of scenarios, while the latter provides a more realistic setting. In this section, we use NS-2 simulations to compare SOAR with traditional shortest path routing. In the next section, we will further compare SOAR with traditional shortest path routing and ExOR using testbed experiments. Below, we first describe our evaluation methodology and then present performance results. Our results show that SOAR significantly improves goodput and fairness of bulk transfer over traditional routing.

4.1 Evaluation Methodology

We implement SOAR in NS-2 (version 2.29) [23]. We use an *ETX-based shortest-path routing protocol* as a baseline comparison. In our ETX implementation, each node sends one broadcast probe per second, and its neighbors measure link loss rates by counting the number of probes received over a 10-second time window.

To capture wireless medium losses in real networks, we augment NS-2 to generate packet losses by dropping packets received at the MAC layer according to a Bernoulli distribution.

Evaluation scenarios: Our evaluation uses 802.11a and disables RTS/CTS, which is the default setting in most wireless networks. As in ExOR, we disable auto-rate and set the data rate to 6 Mbps. We generate 6 Mbps CBR traffic and use a 1000-byte packet size. We use a warm-up period of 500 seconds to let the ETX metrics converge, and then measure network performance over the next 110 seconds.

Performance metrics: We quantify the performance using two metrics: (i) *the average end-to-end goodput of all flows*, where all flows are routed using either shortest path or SOAR, and (ii) fairness using the traditional *Jain's fairness index* [15]. The *goodput* (*i.e.*, total number of non-duplicate received bits per second) is measured over a

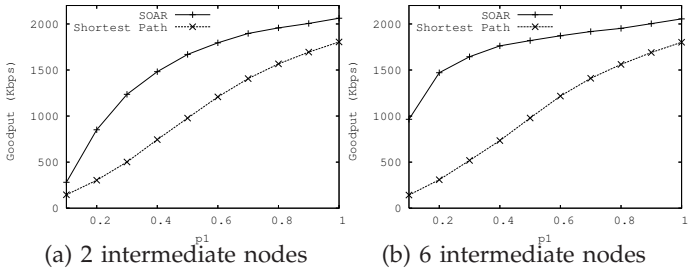
110-second transfer. Jain's fairness index is defined as $(\sum x_i)^2/(n * \sum {x_i}^2)$, where x_i is the goodput of flow i and n is the total number of flows in the network. For both metrics, higher values indicate better performance and fairness.

4.2 Evaluation Results

We evaluate SOAR using a range of network topologies and traffic demands. First, we study the performance of a single flow over simple canonical topologies, and then evaluate the performance of multiple flows using grid and random network topologies.

4.2.1 Single Flow

First, we use diamond and linear chain topologies to evaluate the performance under a single flow.

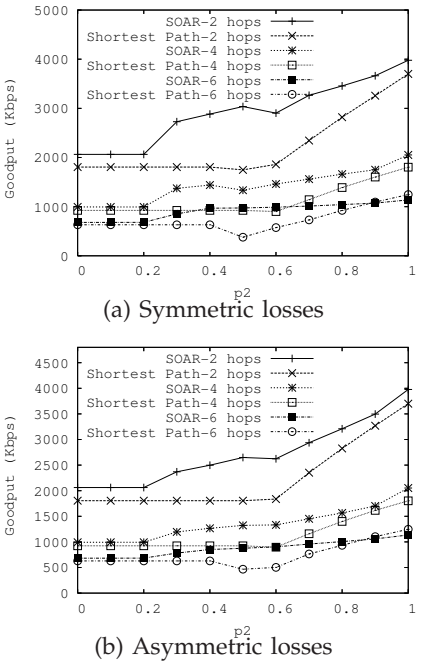


(a) 2 intermediate nodes (b) 6 intermediate nodes

Fig. 7. Diamond Topology.

Diamond topologies: In the diamond topologies (similar to Figure 1), the delivery rate from the source to each intermediate node (denoted as $p1$) 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 (i.e., no loss). Note that there are contention losses in addition to the above inherent wireless link losses. This is true for all the other evaluation as well. Figure 7 compares the goodput of SOAR with the shortest path routing for 2 and 6 intermediate nodes. SOAR performs significantly better for all the values of loss rates. Improvement ranges from 18.37% to 578.62%. For smaller values of $p1$, having more intermediate nodes gives more opportunism for forwarding packets and results in higher goodput improvement. For larger values of $p1$, we restrict the number of forwarding nodes selected (as discussed in Section 3.3.2), so we see similar improvement for different numbers of intermediate nodes. When $p1 = 1$, we see SOAR slightly out-performs shortest path routing. The performance gain comes from two factors: (i) even when $p1 = 1$, there are still collision losses, which SOAR can help to recover via opportunistic routing, and (ii) SOAR uses broadcast transmissions and has no ACK overhead, while shortest path routing uses unicast transmissions and incurs ACK overhead.

Linear chain topologies: Next we use linear chain topologies to evaluate the effectiveness of SOAR in leveraging unexpectedly long transmissions. The one-hop links are perfect, i.e. the delivery rate of the one-hop link in forward and reverse direction is 1. The two-hop link has $p2$ delivery rate. In the case of asymmetric



(a) Symmetric losses

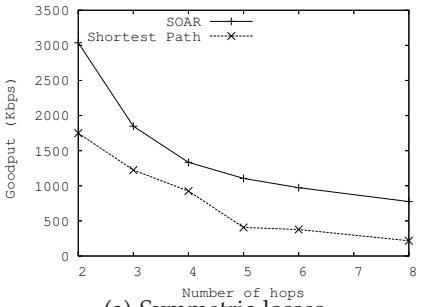
(b) Asymmetric losses

Fig. 8. Linear Topology - goodput for 2-hop, 4-hop, and 6-hop linear chains.

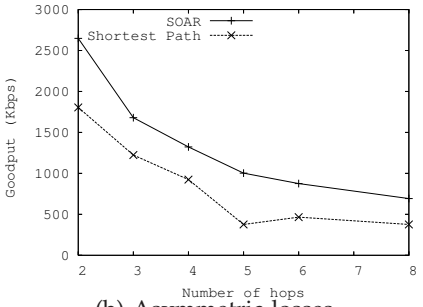
loss rates, $p2$ is the delivery rate of the two-hop links in the forward direction (d_f), and the delivery rate of the reverse direction (d_r) is 1. In the case of symmetric loss rates, $p2$=d_f^2=d_r^2 to ensure the bi-directional delivery rate of $p2$. Figure 8 compares SOAR with shortest path routing for linear chain topologies with symmetric and asymmetric loss rates. We vary $p2$ from 0.0 to 1.0. The improvement is largest (as high as 157.1%) when $p2$ has a moderate delivery rate (around 0.5). This is because when $p2$ is low, there are few long transmissions for SOAR to take advantage of, and when $p2$ is too high, the shortest path routing already directly uses the reliable "two-hop" path, which is close to optimal in this case. Note that when $p2 = 1$, SOAR slightly out-performs shortest path routing since SOAR helps to recover collision losses and has no ACK overhead.

We see larger improvement under symmetric losses than asymmetric losses. This is because for a fixed $p2$, d_f is higher under symmetric losses than asymmetric losses, thereby providing more opportunities for packets to make progress. Furthermore, d_r does not affect SOAR significantly because cumulative/selective ACKs mitigate the effect of ACK losses.

Figure 9 compares SOAR with shortest path routing by varying the number of hops between the source and destination with $p2$ = 0.5. Compared to shortest path routing, SOAR yields an improvement of 37.2% to as high as 256.9%. A 3.5-fold increase occurs in the 8-hop topology, where SOAR has more p2 links to take advantage of. We again observe a higher improvement under symmetric losses for the similar reason as above.

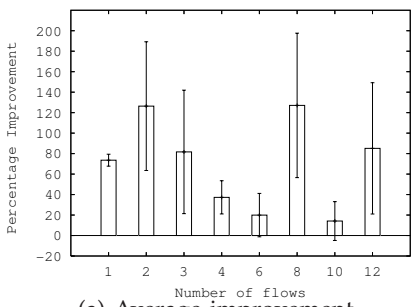


(a) Symmetric losses

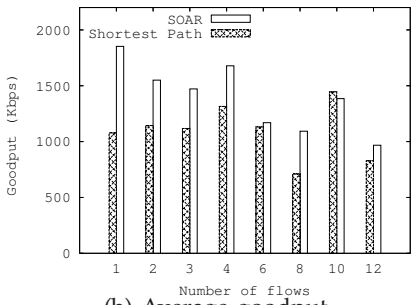


(b) Asymmetric losses

Fig. 9. Linear Topology - goodput comparison across different numbers of hops under p2=0.5.

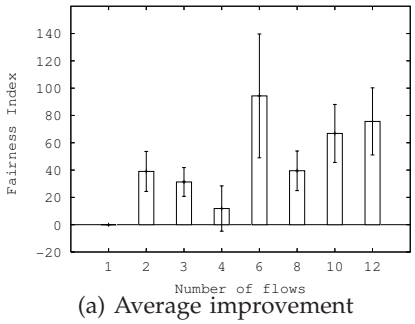


(a) Average improvement

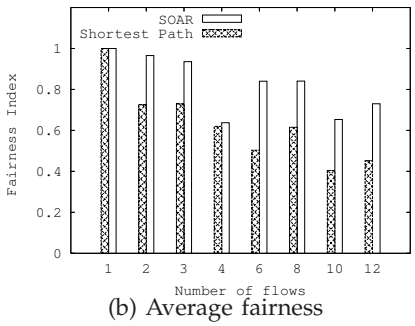


(b) Average goodput

Fig. 10. Goodput results for random flows in a 5x5 grid.



(a) Average improvement



(b) Average fairness

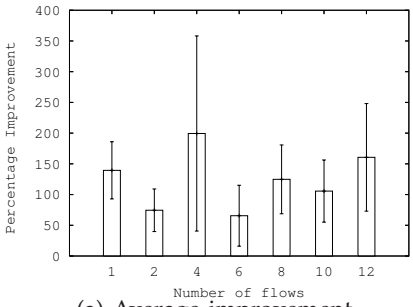
Fig. 11. Fairness results for random flows in a 5x5 grid.

4.2.2 Multiple Flows

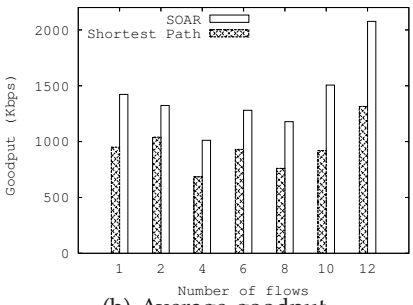
Next we evaluate the performance of multiple flows by randomly choosing source and destination pairs in either grid or random topologies. For each scenario, we report average goodput and average fairness over five different runs. In addition, we also report the average improvement in goodput and Jain's fairness index, which are calculated as: $mean(\frac{Goodput_{SOAR}}{Goodput_{shortest}} - 1)$ and $mean(\frac{Fairness_{SOAR}}{Fairness_{shortest}} - 1)$, respectively. In all the figures, the length of the error bars in the graphs show the standard deviation. The improvement in (absolute) average goodput and fairness differs from average improvement in goodput and fairness in that the former is dominated by the runs with large absolute values whereas the latter weighs all the runs equally. Reporting both metrics allow us to get a more complete picture on where the improvement comes from. As a concrete example, consider SOAR's throughput are 4Mbps in the first run and 2Mbps in the second run, whereas the corresponding values for shortest path are 4Mbps and 1Mbps. The average goodput for SOAR is 6Mbps and for shortest path is 5Mbps, which gives 20% improvement in (absolute) average goodput. In comparison, the average improvement over the two runs are 50%, since 0% improvement for the first run and 100% improvement for the second run.

Grid topology: Figure 10 shows the goodput results for different numbers of random flows in a 5x5 grid topology, where delivery rates of one-hop and two-hop links, denoted as $p1$ and $p2$, are 1 and 0.5, respectively. SOAR out-performs shortest path routing – its average improvement in goodput ranges from 19.9% to 127%. The only exception occurs in 10 simultaneous flows, where the average goodput of shortest path routing is slightly more than SOAR. However SOAR has 14.4% average improvement for 10 flows. This is because among the five runs, one run has several source/destination pairs that are one hop away. Shortest path routing favors these flows and starves the other flows, hence it has higher average goodput. However, SOAR uses rate-limiting to give fair share across the flows. This fact is evident in Figure 11, which shows that SOAR consistently has a significantly higher fairness index.

Random topology: Next we evaluate random flows using 25 nodes in 400m x 400m random topologies. To

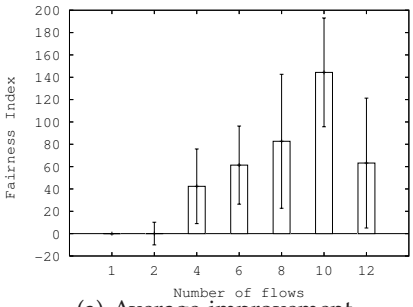


(a) Average improvement

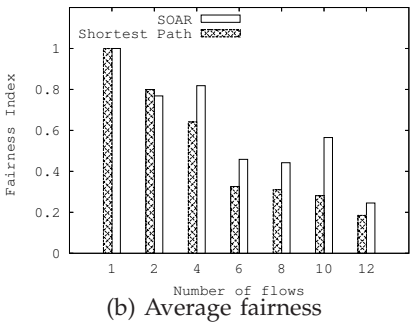


(b) Average goodput

Fig. 12. Goodput results for random 25 nodes placed in a 400 m x 400 m grid.



(a) Average improvement



(b) Average fairness

Fig. 13. Fairness results for random 25 nodes placed in a 400 m x 400 m grid.

create each topology, we first randomly place a node and then iteratively add nodes as long as they're in transmission range of an existing node. Transmission range is 50 m and the interference range is 100 m. The loss rates on the links vary from 0% to 80%. The average neighbors per node is 6.76, which is consistent with previous evaluations [18]. Figures 12 and 13 show the average goodput and fairness indices of 10 random runs as the number of flows varies. SOAR out-performs shortest path routing in both metrics. The average improvement in goodput ranges from 65.6% to 199.4%, and the average improvement in Jain's fairness index is as high as 144.4%.

4.3 Summary

Our evaluation shows that SOAR yields significant improvement over shortest path routing. Furthermore, SOAR effectively supports multiple simultaneous flows by improving both goodput and fairness. The benefit of SOAR is highest under networks with medium link loss rates.

5 Testbed Evaluation

We implement SOAR in a wireless testbed, and compare it with a traditional shortest-path routing protocol (either LQSR or SPP, see below) and a seminal opportunistic routing protocol (ExOR). Furthermore, we disable the rate control in SOAR and compare it against each of the above protocols. In this section, we first present our implementation, and then describe evaluation methodology and performance results.

Note that MORE [3] is a new opportunistic routing protocol proposed recently. As shown in [3], its performance under multiple flows is very close to that of ExOR. Given MORE's similar performance to ExOR, we expect its performance difference from SOAR under multiple flows to be similar to the difference between SOAR and ExOR.

5.1 Implementation

We implement SOAR in Microsoft Windows XP Professional. Both SOAR and LQSR are implemented in the Microsoft Research Mesh Connectivity Layer (MCL) [22], which is a 2.5 layer device that resides as a driver for the wireless card. We configure both LQSR and SOAR to use the ETX metric in the testbed. We use the default MCL ETX implementation to periodically obtain topology and link quality information. Probe packets are sent once per second, and nodes broadcast their link-state once every ten seconds. We set the initial loss rate on a newly discovered link to 80%, and provide sufficient warmup period for the metrics to converge.

We use the ExOR and SPP implementation in [3]. Both are implemented in the Click modular router [6], which runs in Linux. To ensure that the performance of ExOR in Linux and SOAR in Windows is comparable, we compare their performance over a set of controlled topologies, including 1-hop, 2-hop, and 3-hop linear topologies. As shown in Table 1, the performance difference due to different implementation platforms is negligible. We also compare the performance of LQSR and SPP over all the runs. We find that the two shortest path protocols perform within 10% of each other, further validating that the difference in operating systems does not account for the difference observed between SOAR, ExOR and shortest path. Therefore the difference in the following performance numbers of SOAR versus ExOR is due to efficiency of these protocols rather than different implementation platforms.

Hops	p1	p2	SOAR	ExOR
1	1	0	4.093	4.138
2	1	0	2.035	2.029
3	1	0.5	1.894	1.885

TABLE 1
Average goodput (Mbps) of one flow in controlled linear topologies, where $p1$ and $p2$ denote the delivery rates of one-hop and two-hop links, respectively.

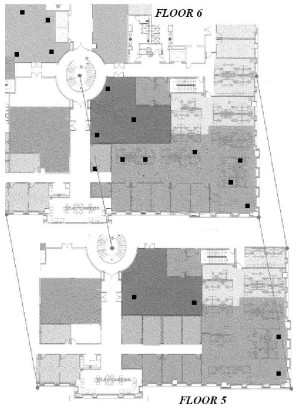


Fig. 14. Node placement in the testbed topology.

5.2 Testbed Description

Our testbed consists of 18 DELL dimensions 1100 PCs, located on two adjacent floors of an office building (see Figure 14). Each machine has a 2.66 GHz Intel Celeron D Processor 330 with 512 MB of memory and is equipped with a 802.11 a/b/g NetGear WAG511 wireless card. To avoid interference with the campus 802.11b/g network, we set the nodes to use 802.11a. Our building has no other 802.11a users. The RTS/CTS handshake is disabled, which is the default setting of the drivers. We disable auto-rate and set the data rate to 6Mbps. Each node uses the lowest transmission power so that we can get up to 6 hops in the network. We keep other parameters, such as the number of unicast retransmission attempts, to their default values.

5.3 Evaluation Methodology

We conduct experiments in our testbed as follows. Each of our experiments consists of a warm-up period and a data transfer period. During the warm-up period, we upload the routing protocol's driver to all nodes (either SOAR, LQSR, SPP or ExOR) and let the routing metrics converge. We use 10 minutes as the warm up period. During the data transfer period, we evaluate the performance of each protocol with a varying number of network flows. As in the evaluation of previous routing protocols for mesh networks (*e.g.*, ETX [7], ExOR [2], LQSR [8]), the source and destination nodes of each flow are randomly selected. Transfers start in parallel and we run a 1-minute transfer for each pair. We augment the ttcp tool included with MCL to generate CBR traffic at 6 Mbps and measure the goodput of each flow. The ExOR and SPP implementations also generate CBR traffic and measure goodput via Click. All measurements and transfers are run during the nights and weekends in order to control the topology as much as possible.

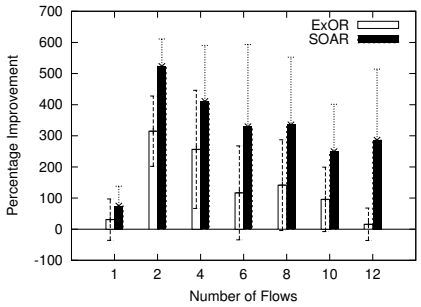
As previously mentioned, we find the performance of the two shortest path protocols to be within 10% of each other. Therefore, when we present the performance of the shortest path, we present the maximum between LQSR and SPP. We take great care to ensure that consistent topologies are used by the Windows protocols and the Linux protocols. The consistency between the protocols within a given operating system is easy to achieve because they both run on the same operating system and temporal variation in topologies is small and easily smoothed out over multiple random runs. Ensuring consistency across operation systems is harder to guarantee since SOAR and LQSR run on Windows and ExOR and SPP run on Linux. Even though on both platforms we use an equivalent transmission power level, which results in similar received signal strength, we still observe substantial difference in the delivery rates of the links across the platforms. To ensure consistent topologies, before experiments we measure the delivery ratio of every link in the network. Whenever the links in Windows have higher delivery rates than those in Linux by 0.4, we use MCL's $artificialdrop$ functionality [22] to generate additional artificial loss to compensate for the difference in their delivery rates (We do not impose artificial losses to the links under Linux even if they are significantly better to favor ExOR). With the imposed artificial losses, the delivery rates of links in Windows are similar to those in Linux – the average difference of all entries in the connectivity matrices under Windows and Linux is 0.0206, where an entry in the connectivity matrix (i, j) denotes the delivery rate from node i to node j.

5.4 Evaluation Results

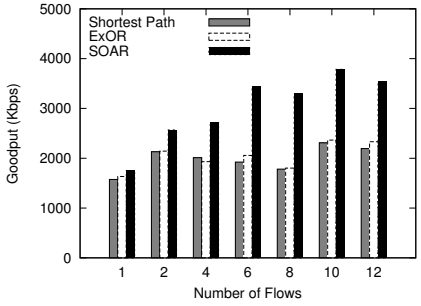
We compare the performance of SOAR, ExOR and shortest path. In all cases, we vary the number of simultaneously active flows. For each scenario, we conduct 30 random runs, where each run consists of a different set of source/destination pairs. We report the average goodput and average fairness index, along with average improvement in goodput and fairness index. The length of error bars in the graphs corresponds to standard deviation.

5.4.1 Protocol Results

Figure 15(a) shows the average goodput improvement of SOAR and ExOR over shortest path (*i.e.*, averaging the percentage of goodput improvement across 30 random pairs), and Figure 15(b) shows the average goodput. As we can see, in all cases SOAR out-performs shortest path and ExOR. The improvement of SOAR over shortest path ranges from 11.2% (1 flow) to 85.3% (8 flows). The improvement of SOAR over ExOR ranges from 7.1% (1 flow) to 83.1% (8 flows). Generally, SOAR has a larger difference in the average goodput for larger



(a) Average goodput improvement over shortest path



(b) Average goodput

Fig. 15. Protocol goodput results in the testbed.

sets of flows. This is because SOAR judiciously selects forwarding nodes to limit resource consumption and rate limits the source; these features are especially important under a larger number of flows. We see the average goodput of ExOR is comparable to that of shortest path in our testbed. We find in a very small number of instances ExOR yields close to 0 goodput. After taking a closer look, we find that ExOR selects nodes as forwarders if they forward at least 10% of the total transmissions in simulation but it does not guarantee the selected forwarding nodes are connected. Disconnection among forwarding nodes can result in serious performance degradation because the data packets cannot make progress across the partition. Even though the occurrence of such disconnection is very low in a dense network, including our testbed, the forwarding list selection in ExOR is inadequate for general topologies. SOAR explicitly ensures forwarding nodes are connected and achieves reasonable goodput in all cases. We also find that the strict timing constraints imposed on ExOR's forwarders inhibits spatial reuse on multi-hop flows, which can limit the effectiveness of opportunistic transmissions. Figure 15(a) shows the improvement of SOAR over shortest path is consistently high. While this graph shows ExOR can provide improvement over shortest path, this improvement is still less than SOAR's.

To further understand the performance benefit of SOAR across different numbers of flows, we study the link loss characteristics in our testbed. We classify a link as a *binary* link if its loss rate is $\leq 20\%$ or $\geq 80\%$, and classify a link as a non-binary link if its loss rate is between 20% and 80%. The number of non-binary links in the network directly affects the performance improvement

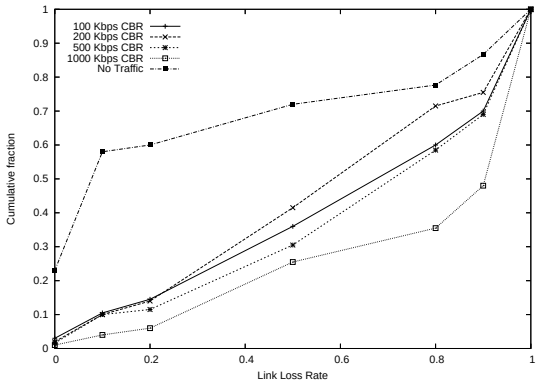
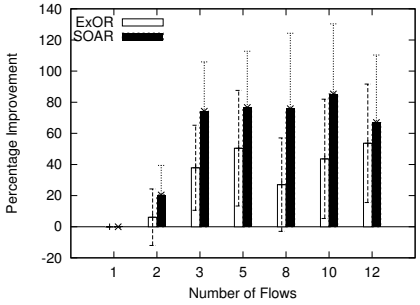
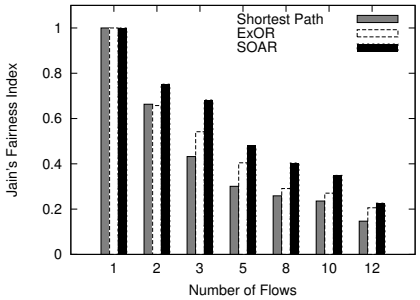


Fig. 16. CDF of link loss rates with and without traffic in our testbed using 802.11a.

of SOAR. Figure 16 shows the cumulative distribution (CDF) of loss rates across different links in our testbed as we vary the amount of traffic. As it shows, there are only 15% non-binary links without traffic. We then measure link loss rates when there are 10 simultaneous flows in the network and each flow generates CBR traffic varying from 100 Kbps to 1000 Kbps. We observe that injecting traffic turns many reliable links into lossy links, thereby creating more non-binary links. For example, when there are 10 random flows, each with 200 Kbps, almost 55% links are non-binary. The larger number of non-binary links creates more opportunism and enlarges the performance benefits of SOAR. ExOR has a tougher time realizing these benefits because the fine-grained timing constraints are hard to enforce on forwarders over many flows. This matches the performance results in Figure 15.



(a) Average improvement over shortest path in fairness index

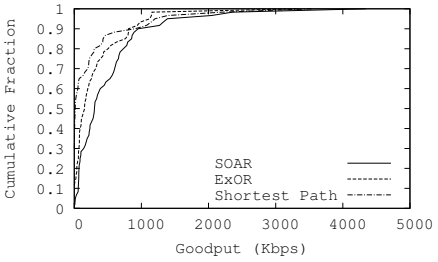


(b) Average fairness index

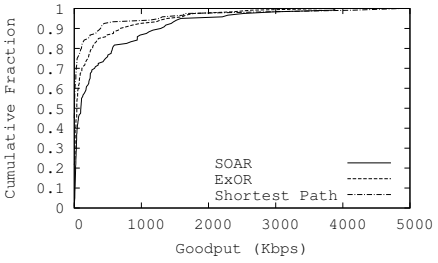
Fig. 17. Protocol fairness results in the testbed.

Next we study the fairness index. Figure 17(a) shows

the improvement of Jain's Fairness Index as we vary the number of simultaneous flows in the network, and Figure 17(b) further shows the average Jain's Fairness Index for each number of flows. We see that SOAR generally provides better fairness than ExOR and shortest path. For instance, the average fairness of SOAR in the eight flow cases is about 60% higher than the average fairness of shortest path and about 50% higher than the average fairness of ExOR. The rate control in SOAR often prevents the starvation of flows and provides better opportunities for equal medium access across different flows.



(a) 4 flows



(b) 8 flows

Fig. 18. CDFs of protocol goodput over all flows under 4 or 8 simultaneous flows in the testbed.

Figures 18(a) and (b) show CDFs of the goodput for all the flows included over 30 random runs under 4 and 8 simultaneous flows, respectively. In all the scenarios, SOAR out-performs ExOR and shortest path routing. In the 4-flow case, the median per-flow goodput of SOAR is 72% higher than the median goodput of ExOR (300 Kbps vs. 174 Kbps). The 90th percentile goodput improves by 17% (951 Kbps vs. 807 Kbps). In the 8-flow case, the median goodput of SOAR is 2.63 times higher than the median goodput for ExOR (100 Kbps vs. 38 Kbps). The 90th percentile goodput of SOAR again nearly doubles that of ExOR (1339 Kbps vs. 681 Kbps). In addition, we observe that 25% of the flows in ExOR get starved (*i.e.*, throughput $\leq$ 10 Kbps), and SOAR reduces the number of starved flows to 10%.

Next, Figure 19 plots goodput as we vary the number of nodes while fixing the number of flows to 4. As we can see, increasing the number of nodes significantly increases SOAR's throughput, since it has more forwarders to take advantage of path diversity. In comparison, the performance improvement for shortest path and ExOR is much smaller.

Finally, Figure 20 further compares the performance of ExOR and shortest path with two versions of SOAR: one

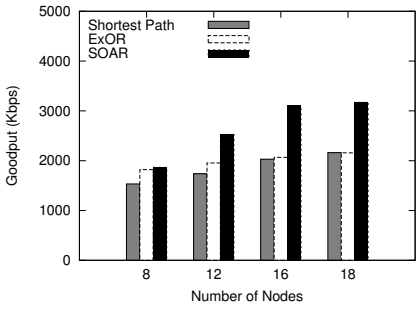


Fig. 19. Goodput under a varying number of nodes when the number of flows is fixed to 4 in the testbed.

enabling the rate limiting in Figure 6 and one disabling the rate limiting (SOAR-nrl). Since the comparison was conducted at a later time, the topology in use was different from the one used in Figure 15. As before, SOAR out-performs ExOR, which in turn out-performs shortest path. Comparing SOAR with SOAR-nrl, we observe that SOAR-nrl performs slightly better under the 1-flow case, since SOAR's rate limiting is a little conservative. However, SOAR out-performs SOAR-nrl under more flows as the importance of rate limiting increases with more flows. In addition, SOAR-nrl out-performs ExOR and shortest path due to better selection of forwarding paths and avoiding strict timing constraints on the forwarders. SOAR-nrl improves ExOR by up to 13%, and improves shortest path by 9-27%. As we would expect, SOAR yields better fairness than all the other schemes, while SOAR-nrl provides similar fairness as ExOR and shortest path. We omit the figure in the interest of brevity. These results suggest that both routing and rate limiting design in SOAR are useful and they each help to contribute to the final performance gain.

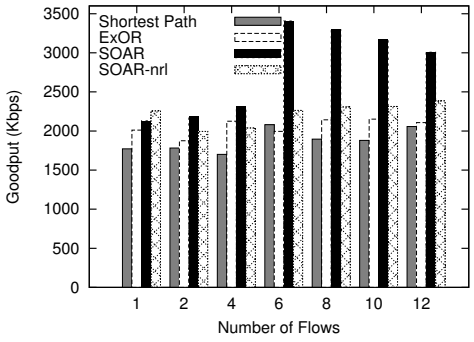


Fig. 20. Comparing SOAR without rate limit in the testbed.

6 Conclusion

In this paper, we develop SOAR, a novel opportunistic routing protocol. SOAR effectively realizes opportunistic forwarding by judiciously selecting forwarding nodes and employing priority-based timers. It further incorporates adaptive rate control to dynamically adjust sending rates according to network conditions and recovers lost packets using efficient local feedback and recovery.

The combination of these techniques enables SOAR to achieve high efficiency and effectively support multiple bulk transfer flows. Our simulations and testbed experiments demonstrate the effectiveness of SOAR.

Through the design and implementation of SOAR, we gain the following insights. First, judiciously selecting forwarding nodes and avoiding diverging paths are important especially for supporting multiple simultaneous flows. Second, rate control is important to the performance of a routing protocol. It improves both aggregate goodput and fairness among multiple competing flows. The joint design of routing and rate limiting as in SOAR is useful, and may be useful to the design of other opportunistic routing protocols. Finally, the default path selection algorithm is important to the performance of opportunistic routing. In this paper, we use the ETX metric to select the default path, but SOAR can easily accommodate other default path selection metrics. The performance of SOAR could further improve with enhanced default path selection, which we plan to investigate as part of our future work.

REFERENCES

[1] B. Awerbuch, D. Holmer, and H. Rubens. Provably secure competitive routing against proactive Byzantine adversaries via reinforcement learning. In *JHU Tech Report Version 1*, May 2003.

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

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

[4] C. Chen, D. Aksoy, and T. Demir. Processed data collection using opportunistic routing in location aware wireless sensor networks. In *Proc. of International Conference on Mobile Data Management*, May 2006.

[5] R. Choudhury and N. Vaidya. MAC-layer anycasting in wireless ad hoc networks. In *Proc. of Hot Topics in Networks (HotNets)*, Nov. 2003.

[6] Click. http://pdos.csail.mit.edu/click/.

[7] 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.

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

[9] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proc. of ACM MOBICOM*, Sept. - Oct. 2004.

[10] R. Dube, C. Rais, K.-E. Wang, and S. Tripathi. Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. In *IEEE Personal Comm*, Feb. 1997.

[11] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla. The impact of of multihop wireless channel on TCP performance. *IEEE Trans. on Mobile Computing*, Mar. 2005.

[12] T. Goff, N. Abu-Aahazaleh, D. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In *Proc. of ACM MOBICOM*, Jul. 2001.

[13] Y. C. Hu and D. B. Johnson. Design and demonstration of live audio and video over multi-hop wireless networks. In *Proc. of MILCOM*, Oct. 2002.

[14] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *Proc. ACM MOBICOM '2003*, Sept. 2003.

[15] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report TR-301*, Sep 1984.

[16] 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.

[17] R. Karrer, A. Sabharwal, and E. Knightly. Enabling large-scale wireless broadband: The case for TAPs. In *Proc. of HotNets*, Nov. 2003.

[18] L. Kleinrock and J. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. In *NTC*, Dec. 1978.

[19] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, and E. Rozner. Predictable performance optimization for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 38(4):413–426, 2008.

[20] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, Z. Zhong, G. Deshpande, and E. Rozner. Effects of interference on throughput of wireless mesh networks: Pathologies and a preliminary solution. In *Proc. of HotNets-VI*, Nov. 2007.

[21] R. Madan, S. Cui, S. Lall, and A. Goldsmith. Cross-layer design for lifetime maximization in interference-limited wireless ad hoc networks. *IEEE trans. Wireless Communications*, 2006.

[22] Mesh connectivity layer. http://research.microsoft.com/mesh/#software.

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

[24] V. Paxson and M. Allman. Computing tcp's retransmission timer. *IETF Internet DRAFT*, 2000. http://www3.ietf.org/proceedings/00jul/I-D/paxson-tcp-rto-01.txt.

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

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

[27] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. In *Proc. of ACM SIGCOMM*, Sept. 2006.

[28] MIT Roofnet. http://www.pdos.lcs.mit.edu/roofnet/.

[29] E. Rozner, J. Seshadri, Y. Mehta, and L. Qiu. Simple opportunistic routing for wireless mesh networks. In *Second IEEE Workshop on Wireless Mesh Networks*, Sept. 2006.

[30] Seattle wireless. http://www.seattlewireless.net.

[31] W. Wang, X. Liu, and D. Krishnaswamy. Robust routing and scheduling in wireless mesh networks. In *Proc. of IEEE SECON*, 2007.

[32] C. Westphal. Opportunistic routing in dynamic ad hoc networks: The oprah protocol. In *Proc. of IEEE MASS*, Oct. 2006.

[33] City-wide Wi-Fi rolls out in UK. http://news.bbc.co.uk/2/hi/technology/4578114.stm.

[34] Cities unleash free Wi-Fi. http://www.wired.com/gadgets/wireless/news/2005/10/68999.

[35] Y. Yuan, H. Yuan, S. H. Wong, S. Lu, and W. Arbaugh. ROMER: resilient opportunistic mesh routing for wireless mesh networks. In *Proc. of IEEE WiMESH*, Sept. 2005.

[36] Z. Zhong and S. Nelakuditi. On the efficacy of opportunistic routing. In *Proc. of IEEE SECON*, Jun. 2007.

Eric Rozner is a Ph.D. candidate in the Department of Computer Sciences at the University of Texas at Austin. He received B.S. degree from University of Wisconsin at Madison in 2005. His research interests are Internet and wireless networking.

Jayesh Seshadri received his M.A. in Computer Sciences from the University of Texas at Austin in 2007. He is a Software Engineer at VMware, Inc, Palo Alto, CA, with a specific focus on distributed systems management. He also has a B.E. in Computer Science and Engineering from Anna University, India.

Yogita Ashok Mehta received her Masters degree from University of Texas at Austin in 2007. She received her Bachelor's in Engineering from University of Mumbai in 2005. She is currently working as Software Engineer at Google Inc.

Lili Qiu received her Ph.D. from Cornell University in 2001. She is an Assistant Professor in the Department of Computer Sciences at the University of Texas at Austin. Her research area is computer networks, with special focuses on wireless network management, content distribution, Internet measurement, and overlay networks. She is a recipient of NSF career award (2006), and a senior IEEE member. She was a researcher at Microsoft Research during 2001-2004.