Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2014, Article ID 637278, 12 pages
http://dx.doi.org/10.1155/2014/637278

Research Article

SCAR: A Coding-Aware Routing Protocol with Self-Recommendation in Static Wireless Ad Hoc Networks

Jin Wang,[1] Cenzhe Zhu,[1] Teck Yoong Chai,[2] and Wai-Choong Wong[1]

[1] *National University of Singapore, Singapore 119613*
[2] *Institute for Infocomm Research, Singapore 138632*

Correspondence should be addressed to Jin Wang; g0900220@nus.edu.sg

Received 9 February 2014; Accepted 10 June 2014; Published 25 June 2014

Academic Editor: Liansheng Tan

There have been researchers working on the design of coding-aware routing protocols to exploit the power of network coding in static wireless ad hoc networks. However, most of them have overlooked the fact that routing decisions for multiple flows need to be coordinated in order to achieve maximum throughput. A mechanism to alter prior routing decisions is mandatory for this purpose. In this paper, we propose Self-recommendation coding-aware routing (SCAR), to provide such a mechanism. With intermediate nodes sensing the arrival of new flows, they can send self-recommendations to source nodes, triggering a route-change procedure. In the route-change procedure, adjustments are applied to the route metrics so that the newly recommended route can be fairly weighed without bias. A thorough analysis of coding structures is carried out, and a series of indicators are devised to predict how much throughput benefits we can gain from the mere knowledge of the topology. The rationale behind the protocol design and the effectiveness of the indicators are further justified by a series of simulations. Results show that SCAR can exploit coding opportunities better and provide higher throughput than other coding-aware routing protocols. It is also shown that the opportunity of throughput gain is ubiquitous and it is tested on many typical topologies.

1. Introduction

Network coding is a technique that exploits the broadcast nature in wireless networks to provide throughput improvements. The idea is first proposed in [1] and later developed as a practical network protocol in COPE [2]. The latter describes an "X" scenario, and it is mostly used to illustrate how network coding can reduce the number of transmissions for a given task. As shown in Figure 1, a packet from A to D and a packet from C to B can be transmitted with a total of 3 transmissions using network coding technique. They are as follows: one transmission from A to O, one transmission from C to O, and a broadcast of the *XOR*ed packet by O. This "X" structure is the only applicable network structure for the COPE protocol, which is seen today as the major limitation of COPE.

To relax the restriction on coding-possible structures in static wireless ad hoc networks, distributed coding-aware routing (DCAR) [3] proposes a routing protocol that is applicable in all network topologies. With the general coding conditions defined, this protocol can discover coding-possible routes in various kinds of network structures. In addition, [3] proposes a coding-aware routing metric (CRM) to quantify and compare the merits between coding-possible and coding-impossible routes.

Although current coding-aware routing protocols, with the representative of DCAR, are able to discover coding-possible routes, they lack a mechanism to conduct coordination among different flows. In this paper, the term "flow" is referred to a possible path between a source node and a destination node. If the route is selected with the mere information collected from a single flow, this routing decision essentially achieves only a local optimum. If we alter routing decisions for other flows, we might end up with a higher overall throughput. Our work provides us with a means for intermediate nodes to sense the arrival of the new flow and to participate in the route-maintenance phase. This is called *self-recommendation* (SR). SR enables a reassessment of prior routing decisions and provides a procedure to update it. Consider the scenario shown in Figure 2 where each node can

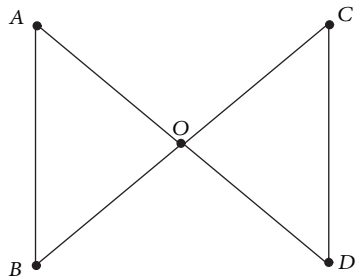


Figure 1: In "X" scenario, a packet from A to D and a packet from C to B can be transmitted with a total of 3 transmissions using network coding technique.

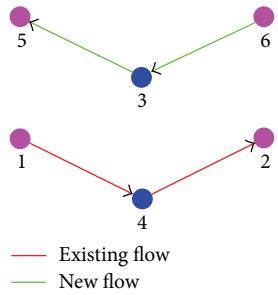


Figure 2: A simple test scenario for SCAR.

only reach its nearest neighbor. Suppose there is an existing flow 1 → 4 → 2. With the arrival of a new flow 6 → 3 → 5, new coding opportunities are introduced at node 3. Node 3 can sense this new opportunity as an intermediate node. Without the mechanism of self-recommendation, node 1 will not be aware of the flow arrival, not to mention changing the route to utilize this coding opportunity. Our protocol specifies when intermediate nodes will send self-recommendations and how source nodes will compare the available routes. In our protocol, node 1 will reroute via 1 → 3 → 2, fully utilizing the coding opportunity.

In order to design a protocol that is able to handle the case we described, we need to solve several problems. The first is how to discover the arrival of new flows and thus new coding opportunities. Secondly, we need to know how to evaluate these opportunities and decide whether to change existing routes. To complete the work, we also need a third step to test whether the newly introduced complexity is well compensated with higher performances. Here we first briefly answer these questions. The network dynamics are monitored by intermediate nodes. Like most routing protocols, our protocol also employs a RREQ-RREP procedure to discover routes. The benchmark protocol is dynamic source routing (DSR) [4] protocol and parts of technique of DCAR [3] protocol are also included. In an on-demand routing protocol, RREQ/RREP packets suggest new incoming flows. This discovered information is encoded in a self-recommendation (SR) packet and is sent to the source node, triggering a route-change (RC) procedure to handle it. The route-change procedure employs a new unbiased routing metric to evaluate all possible routes and makes updates to the routing table. On the other hand, we have carefully analyzed the causes for the throughput gain and devised a series of indicators to predict the gain. Through extensive simulations on various topologies, we conclude empirically that throughput gains are observed whenever the indicators imply so, and the presence of throughput gain indicators is ubiquitous.

The contributions of this work are as follows.

(1) We have proposed a practical coding-aware routing protocol that enables coordination among flows. This coordination is done with the help of self-recommendations (SRs) from intermediate nodes. The route-change (RC) procedure is devised to synchronize different nodes.

(2) We have studied the coding-aware routing metric (CRM) for quantifying the merits of candidate routes. The concepts of biased and unbiased CRM, as well as the methods to convert biased CRM to unbiased CRM, are introduced. We design packet formats and route-maintenance procedure to gather necessary information to make the conversion possible.

(3) We have analyzed various coding structures and propose a series of indicators. These indicators can be used to estimate how much throughput gain is achievable under SCAR. They can also be used as a guide for modifying network topologies to improve throughput.

Compared to previous coding-aware routing protocols [5–8], the performance evaluation of our protocol shows that our protocol can significantly improve throughput in many network topologies. It is also shown that our proposed protocol is robust against different traffic patterns. The order of arrivals of flows is generally indifferent for our protocol.

The rest of the paper is organized as such. In Section 2 we introduce more related works as well as some basic building blocks in coding-aware routing protocols. Section 3 gives an overview of our SCAR protocol, with Section 4 detailing the route-change procedure in our protocol. Section 5 analyzes coding structures and proposes a series of indicators to estimate throughput gain. The performance evaluation is given in Section 6 and we conclude in Section 7.

2. Related Works and Preliminaries

Network coding is a research area that may have interesting applications in practical networking systems. It can improve throughput when two wireless nodes communicate via a common intermediate relay node. Works on network coding started with a pioneering paper by Ahlswede et al. [1], who showed that with network coding, as symbol size approaches infinity, a source can multicast information at a theoretically maximum rate. This rate can be calculated by modelling the network as a max-flow min-cut problem in graph theory. The studies in [9, 10] focus on the construction of efficiency of network coding. A theoretical study on the efficiency of network coding in multihop wireless networks is given in [11, 12]. Recently, there have been several works [2, 13–17] on

multiple unicast sessions by applying wireless network coding. These approaches are categorized as interflow network coding, which encodes multiple packets destined to different next hops and broadcasts them together.

The one-hop coding opportunities were first studied and exploited by the COPE protocol in [2]. The simple one-hop nature and the empirical success of COPE have since motivated numerous subsequent works. Some examples include [18], which finds the energy-efficient scheduling with opportunistic coding [17], which calculates the maximum number of overhearing opportunities under practical wireless settings. References [3, 19, 20] develop techniques to select routes that create more coding opportunities, [21, 22] jointly optimize network coding and scheduling, [23] picks the modulation rate that takes into account both coding gain and data rate, and [24] proposes a technique to XOR packets that use different modulation schemes. Some recent efforts considered cross-layer approaches in the context of coding-aware routing [19]. Chaporkar and Proutiere [21] also studied the issue of joint scheduling and COPE-like coding, focusing on characterizing the capacity region of a simplified version of COPE combined with scheduling according to backpressure. In [3], Le et al. proposed an on-demand distributed coding-aware unicast routing protocol (DCAR) for selecting a high throughput path with more potential coding opportunities by introducing a coding-aware routing metric called CRM, which jointly considers coding opportunities, congestion levels, and related factors for comparing coding-possible and coding-impossible routes when multiple routes are available. In addition, DCAR can detect coding opportunities on an entire path, thus eliminating the two-hop coding limitation in COPE. However, both of these existing coding-aware routing protocols do not consider the dynamic nature of data flows in a network.

Since the metric used in DCAR (CRM) is of great importance to the formulation of this paper, we will briefly incorporate part of the derivation of CRM here. The CRM of a route is the sum of the CRMs through all intermediate nodes. When comparing multiple possible routes at the source node, the route that has the smallest CRM value will be picked. The calculation of CRM at a single node is nevertheless nontrivial, and the calculation process reflects the evolution of the design of this metric. The CRM of a node starts from the average queue length of a sliding window (Q). A smaller value suggests less traffic, thus shorter delays. Coding opportunities are then considered to transform Q to MQ (modified queue length). MQ is calculated as follows. First the queue lengths of all transmission queues inside the specified node are summarized. With each transmission queue denoted as a vertex and each coding-possible queue pair denoted as an edge, we form a "coding graph." The queue length of the queues is attached to the vertexes in the graph. After solving a max-clique problem on the graph, MQ is calculated as

$$\mathrm{MQ} = \sum_{C_i \in \mathbf{C}} \max_{j \in C_i} \left\{ Q_j \right\}, \tag{1}$$

where $\mathbf{C}$ is the set of cliques that solves max-clique problem and Q_j is the average queue length of queue j. This MQ measures one node's ability to transmit packets promptly while taking into account the effect of network coding. MQ is further extended to MIQ when considering the effect of neighboring interference. Consider the following:

$$\mathrm{MIQ}(n) = \mathrm{MQ}(n) + \sum_{i \in I(n)} \mathrm{MQ}(i), \tag{2}$$

where n is the specified node and $I(n)$ is the set of neighbors of node n. Lastly, CRM is derived by further taking into account packet losses:

$$\mathrm{CRM} = \frac{1 + \mathrm{MIQ}}{1 - P_l}, \tag{3}$$

where P_l is the probability of packet loss.

3. Protocol Overview

In the following subsection we explain the major components of the protocol.

3.1. Two Tables to Store Flow Information. The route-discovery procedure in DCAR [3] has already collected enough information for future routing coordination, but DCAR simply dumps it after the routes are constructed. In the SCAR protocol, each node maintains two tables: potential-flow table and relayed-flow table.

When an RREP passes by, the node will store the route information in the potential-flow-table. A potential-flow table entry can be moved to the relayed-flow table when data packets are received from this route. It will be moved back if the node fails to receive data packets from this route for a period of time. In a traditional coding-aware routing protocol, a potential-flow-table entry is only used for calculating the coding-aware metric when forwarding RREP back to the source. In our protocol, this potential-flow information is stored for a longer time, so that this intermediate node can possibly recommend the source node to use this route in the future. Obviously, a route can be either in the potential- or the relayed-flow table but could not be in both simultaneously. Self-recommendations would only be done for potential flows (because relayed flows are already chosen by their sources).

3.2. Self-Recommendation Sent to the Source. With two tables storing the necessary information for making self-recommendation, the next question is when to initiate the SR procedure. The SR procedure should be triggered when a new flow joins the network and a potential coding opportunity is found. In addition, when an existing flow is terminated, the source should also be notified of the change and routes should be updated given the new flow pattern.

In the first case, upon receiving an RREP of flow f_n, an intermediate node will check whether this new flow can be coded with any of the potential flows it has stored. If a potential flow f_p can be coded with f_n, this node can send a self-recommendation to the source node "later," indicating a potential coding opportunity. It is sent "later" because

we need to wait for this new flow f_n to stabilize. If after a delay, this new flow f_n is moved to the relayed-flow table, we can safely recommend f_p to the source of f_n. Otherwise f_n remains in the potential-flow table, which means that the source of f_n has found a better route than the one via this intermediate node. To utilize this coding opportunity we will have to send out two recommendations, which will very likely cause turbulence to the network. For now, our protocol will send self-recommendation only when f_n is later moved to the relayed-flow table in order to reduce overhead.

In the second case, when an intermediate node detects the queue length for a certain relayed flow, f_1 is drastically decreasing, and this decrease has continued for some time, it then checks whether there is any other existing flow f_2 being coded with f_1. If so, the source of f_2 will be notified of the change by an SR packet, and the route should be updated.

3.3. Handling Self-Recommendations. Not all recommendations should be accepted. The source retains its autonomy in determining which route to use. In order to compare this recommended route with other existing routes, the source node will send modified RREQs to update the CRM of all existing routes as well as the recommended route. Unlike RREQ, modified RREQs are unicast packets that use source routing. This modified RREQ will request the destination and relay nodes to append the CRM value in reply packets (modified RREPs).

3.4. Decision Making. Upon receiving the modified RREP packets, the source node updates the CRM value of all routes. Ideally the source node can compare among these routes and choose the route with the minimum CRM. However, the CRM values received are biased. They will favor the currently in-use route most of the time. In the next section we will discuss how to remove this bias and improve the effectiveness of self-recommendations. Altering the routing decision is not the end of the story. The routing decision we have just made can be wrong sometimes because of two reasons.

(1) The CRM metric proposed in [3] is only a heuristic indicator for the suitability of a route being used in the context of network coding. There is no guarantee that using a route with higher CRM will result in a lower throughput.

(2) The CRM values calculated can be imprecise. The max-clique problem inherent in CRM calculation is NP complete, so approximation algorithms are usually used to calculate the CRM.

In order to discover and recover from these errors, each recommended route is given a probation period. After its probation, the source node will refresh its routing table through the modified RREQ/RREP procedure again. Monitoring this routing table refresh can give us hints whether recommending that route is a right decision.

3.5. Controlling the Frequency of Self-Recommendations. For a particular flow, there might be several intermediate nodes that will send SRs to the source node. An intermediate node may also send multiple SRs for the same potential flow. Thus the source will be constantly flooded with requests to update. To prevent this from happening, a way to control the frequency of SR is much needed. This control is done in two parts.

(1) After an intermediate node sends an SR for a certain path, it is blocked from sending this specific SR for a period of time.

(2) Whenever a source receives a new SR for a particular destination, the route-change (RC) procedure is initiated. Subsequent SRs for the same destination will be taken care of but will not trigger sending of modified RREQ again. Details for the RC procedure are explained in the next section.

4. Route-Change Procedure

This section describes how self-recommendation packets can alter existing routes. The route changes happen in a distributed but ordered manner.

4.1. Procedure Timeline. Why is a timeline important? In the simple scenario shown in Figure 2, synchronization is not a must. However, as the topology incorporates more nodes and the network accommodates more flows, modified RREPs may not immediately follow their initiating modified RREQs. They can be severely delayed or simply get lost. We need to clarify how to handle these delayed packets. Moreover, intermediate nodes can receive more than one self-recommendation in the same time frame, therefore, triggering even more modified RREQ/RREP packets. We need a synchronization method to coordinate all the self-recommendation, modified RREQ, and modified RREP packets so that they do not mess with each other. This synchronization method is a common timeline.

The route-change procedure starts with the arrival of an SR packet. SR packets can either initiate a route-change procedure, or they can join an existing procedure. For a specific source node, there can be several parallel RC processing instances going on, each for a different destination. Modified RREQ packets are sent at certain time points in the procedure and the modified RREP packets are handled depending on the time they were received. Figure 3 shows the timeline for one of such RC instances.

An RC procedure consists of three periods, and we discuss them separately as follows.

(1) Waiting period: upon receiving SR0 for destination D, if there has not been any RC instance for D, a new RC instance is initiated. This node then sends several modified RREQs (short for MQ in Figure 3), each for a known route to D. During the waiting period, any SR packets and any modified RREP (short for MP) packets for this destination will be temporarily stored.

(2) Probation period: at T_1, the node examines the SR/MP packets received so far, compares the previous in-use route with the recommended routes, and decides which route is going to be adopted.

Data packets are then sent through the newly adopted route. SR and MP packets received during this period are simply discarded.

(3) Confirming period: at the end of probation period, this node will again send modified RREQs for each route and gathers modified RREPs from these routes. This confirming period provides the chance to revert previous route changes if the recommended route has higher CRM than the previously-in-use route.

4.2. The Metric: Unbiased CRM. In the last section we mentioned that the CRM values in modified RREP packets received are biased. They will favor the currently in-use route. We justify this statement here.

The CRM values returned for the previously-in-use route $r_{\text{in-use}}$ and the recommended route r_{sr} are unbiased only if there is no data packet sent through $r_{\text{in-use}}$. The unbiased CRMs will weigh two routes equally. Consider a scenario as in Figure 2. The flow ($1 \rightarrow 4 \rightarrow 2$) has existed for some time. As the new flow ($6 \rightarrow 3 \rightarrow 5$) arrives, node 3 will send a self-recommendation to node 1. The previously in-use route is $r_{\text{in-use}}$ ($1 \rightarrow 4 \rightarrow 2$ in Figure 2), and the recommended route is r_{sr} ($1 \rightarrow 3 \rightarrow 2$). In Figure 2, unfortunately, there have been data packets sent through ($1 \rightarrow 4 \rightarrow 2$) before the self-recommendation is received by node 1. This traffic will increase the static modified queue length (this is a component of the CRM metric, refer to [3] for more details) of node 4 from 0 to some value *diff*. This modified queue length is transmitted in node 4's hello message, so node 3 will account for this *diff* when calculating the CRM for ($1 \rightarrow 3 \rightarrow 2$). Ironically, flow ($1 \rightarrow 4 \rightarrow 2$) will not account for *diff* in calculating its own CRM. As a result, node 1 receives a higher CRM for flow ($1 \rightarrow 3 \rightarrow 2$), which is unfair. Such comparison between two flows makes no sense and also leads to a wrong routing decision.

The remedy for the problem is a more sophisticated modified RREQ-RREP procedure. In this scheme, though the CRM values are calculated as usual, node 1 will preprocess the values and make sure that the final comparison is unbiased and effective.

The packet formats of the modified RREQ/RREP are shown in Figure 4 where *Path* denotes the route of interest, and *InUsePath* the previously in-use route. In addition to the normal CRM calculation, when the modified RREP is transmitted back towards the source node, the intermediate nodes update the *Count* of the packet. If a node in *InUsePath* happens to be its neighbor, the corresponding count is incremented by 1. Again using the simple scenario in Figure 2, the modified RREP for flow ($1 \rightarrow 3 \rightarrow 2$) will have a count of 1. This means node 4, which is in the *InUsePath*, is node 3's neighbor. Moreover, the modified RREP packet has an extra optional field. This field will be present only when *Path* is exactly the same as *InUsePath*. This field contains the difference between the static modified queue lengths of the two cases: calculated with the in-use route and without the in-use route.

When the source node receives the modified RREPs for both $r_{\text{in-use}}$ and r_{sr}, we have enough information to adjust the CRM values to the unbiased values. The modified RREP from $r_{\text{in-use}}$ has the optional field, and the modified RREP from r_{sr} has the count field. For each intermediate node in the previously in-use route, subtract *count* × *difference* from the CRM value of r_{sr}. Therefore, the CRM values of both flows are the net effect of the previously in-use route traffic, and they can be compared directly to decide which route should be chosen.

5. Topology Analysis

The performance of a coding-aware routing protocol is highly dependent on the network topology, or rather "coding structures." To better understand the importance of routing coordination, we have studied a series of indicators to quantify how much a topology is susceptible to previous routing decisions, and in that case, what benefits could routing coordinations bring. Given the indicators, the performance of the routing protocol can be estimated even before simulations or deployment. Furthermore, these indicators shed some light on coding structures, so that topology modification can be done in a guided direction seeking throughput gain.

Before describing the indicators, we first define some terms used.

Link: undirected multihop path connecting one node to another.

Route: directed multihop path connecting a source node to a destination node.

Flow or S-D pair: an ordered pair of source and destination nodes, without specifying the route chosen.

Route coding-possible: this is the legacy definition. For multiple routes, if each of their destinations can decode the coded packets, these routes are coding-possible.

Flow coding-possible: this is a new definition used in this paper. If n flows are said to be coding-possible, there exist n coding-possible routes for the n respective flows.

The series of indicators are named as CN with N starting from 2. For *two* 2-hop flows that are coding-possible, say, (S_1-D_1, S_2-D_2), we assume the number of possible routes for S_1-D_1 is larger than or equal to that for S_2-D_2. If we choose a route R_1 from all the possible routes for S_1-D_1 randomly, the probability that there exists a route R_2 for S_2-D_2 that is coding-possible with R_1 is defined as $C2$ for S_1-D_1, S_2-D_2. For comparison purpose, this $C2$ value is then adjusted as $C2_{\text{adj}} = 2 \times |C2 - 0.5|$. The rationale of doing this adjustment is discussed later in this section. As an illustration, we describe the calculation of $C2$ for 1-2, 6-5 in the topology shown in Figure 2. There are two possible routes for flow (1-2) and only one possible route for (6-5). When choosing the route (1-4-2) for flow (1-2), there is no possible route for flow (6-5) to code, but if we choose the route (1-3-2) for flow (1-2), there is possible route. So the $C2$ value for these two flow is 0.5, and after adjustment, 0. The $C2$ value of the topology is simply an average of all the adjusted $C2$ values

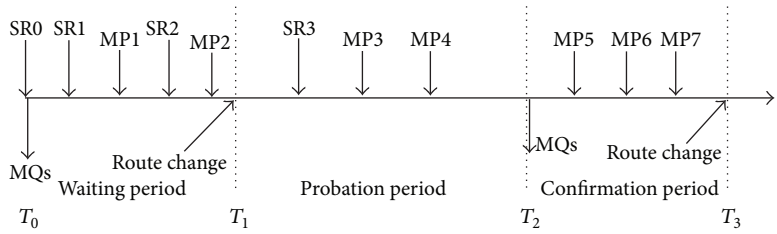
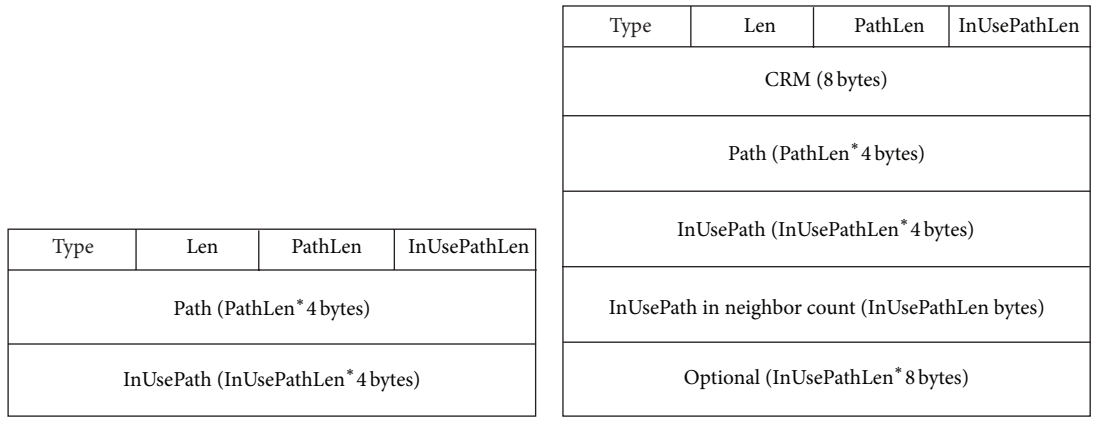


Figure 3: Route-change procedure timeline (SR: self-recommendation, MQ: modified RREQ packet, MP: modified RREP packet).



(a) Modified RREQ (b) Modified RREP

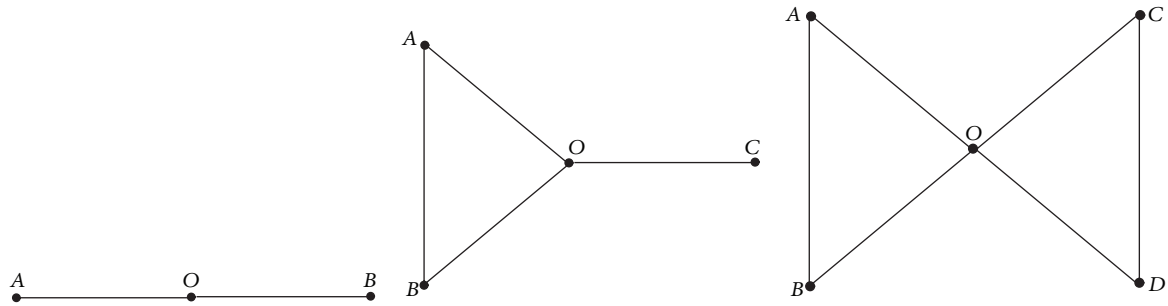
Figure 4: Modified RREQ/RREP packet formats.

of every pair of coding-possible flows. $C3$ is defined similarly with the only exception that we consider *three* 2-hop flows, while still randomly selecting one route for the first flow. The adjustment of $C3$ is the same, $C3_{adj} = 2 \times |C3 - 0.5|$. Note that we consider only 2-hop flows. This is because, in our simulation, coding opportunities are mostly confined in 2-hop scenarios. Though we had also formulated the indicators for multihop routes, simulation results did not agree with the indicators. Multihop transmissions are prone to packet loss and they usually have outrageous delays compared to 2-hop transmissions. Given the definition of CN, if we can find a systematic way of listing all coding-possible route pairs/triples, $C2$ and $C3$ can be easily calculated. We then provide a graph-based algorithm to enumerate all possible coding-possible route pairs/triples.

Consider 3 possible node placements as shown in Figure 5. The three subfigures considered how many end nodes these two routes have in common. Figure 5(a) shows the case when two routes have their both end nodes in common; that is, any 2-hop route can be coded with its reverse route. In Figure 5(b), route *A*-*O*-*C* can be coded with route *C*-*O*-*B*. Similarly, route *B*-*O*-*C* can be coded with *C*-*O*-*A*. In this topology, two route coding pairs are found. Note that we do not count *A*-*O*-*C* and *C*-*O*-*A* in this figure, because this pair is already counted in Figure 5(a). In Figure 5(c), where two routes share no common end node, four route coding pairs are found. They are *A*-*O*-*C* with *D*-*O*-*B*, *A*-*O*-*D* with *C*-*O*-*B*, *C*-*O*-*A* with *B*-*O*-*D*, and *C*-*O*-*B* with *A*-*O*-*D*. Similarly, the pair *A*-*O*-*C* and *C*-*O*-*B* is not counted here as it has been counted in the second case. In summary, our problem is reduced to a problem of listing these structures, and then list the route pairs as above as long as the respective sources and destinations of these routes are not neighbors.

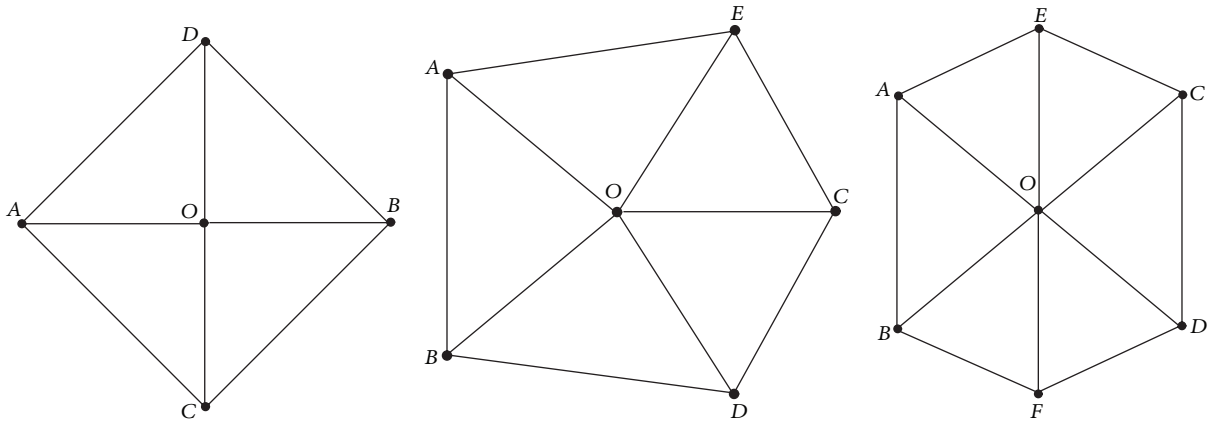
So far, the work seems trivial. But as we generalize to coding-possible route triples, we begin to find some patterns. A coding-possible route triple can be seen as a coding-possible route pair with an additional route. The additional route should meet certain criteria, that is, the destination/source of the additional route should be neighbor to the source/destination of either of the other two routes (consider a sharing node as a special case of neighborhood). So we derive the three possible topologies for coding-possible route triples from Figure 5, yielding Figure 6.

Extending Figure 5(a), the additional route cannot share the end node with the previous routes, yielding a 4-node structure with 4 triangles (Figure 6(a)). With the initial two routes *A*-*O*-*B* and *B*-*O*-*A*, the new route can be either *C*-*O*-*D* or *D*-*O*-*C*. In addition, this topology can be rotated 90 degree, with *C*-*O*-*D* and *D*-*O*-*C* chosen to be the initial routes. So we can find $2 \times 2 = 4$ route triples. Extending Figure 5(b), the additional route again cannot share end nodes with the previous two, introducing the additional nodes *D* and *E* in Figure 6(b). With the link *O*-*C* shared by the initial two routes, there are two coding triples in this topology: (*A*-*O*-*C*, *C*-*O*-*B*, *D*-*O*-*E*) and (*B*-*O*-*C*, *C*-*O*-*A*, *E*-*O*-*D*). Again, due to the symmetry of the topology, the number of route triples in this topology should be multiplied by 5, giving $2 \times 5 = 10$. Extending Figure 5(c) is the most complex one.



(a) A route coded with its reverse (b) Routes share an end node (c) Routes with distinct end nodes

Figure 5: Possible topologies for 2 coding-possible route pairs.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 6: Possible topologies for 3 coding-possible route triples.

The additional route can potentially share end nodes with initial routes. However, further investigation reveals that sharing end nodes will reduce the situation back to the one we have covered in Figures 6(a) and 6(b). The only new situation we should consider is shown in Figure 6(c), where altogether 6 end nodes are considered. There are two route triples in this topology: (*A*-*O*-*D*, *C*-*O*-*B*, *F*-*O*-*E*) and (*B*-*O*-*C*, *D*-*O*-*A*, *E*-*O*-*F*). Rotating the topology will not give new route triples.

Now we can see the importance of triangles in these topologies. As long as we list all triangles in a topology, by examining the adjacency of these triangles, we can easily list all route pairs and route triples. This deduction can be carried forward for 4, 5, or even more routes. By the definition, $C2$ and $C3$ can be easily calculated. We then discuss what are the implications of these indicators.

The improvement of SCAR over DCAR lies in the route-maintenance phase. SCAR provides a mechanism to revise already-established routes. Therefore, if prior routes are poorly selected, they can be changed to consider any new flows in the network. In terms of $C2$ values, any value between 0 and 1 leaves the possibility of choosing the "wrong" route initially. DCAR is more likely to choose the "right" route if the $C2$ value is closer to 1, and it is less likely to choose right route if the $C2$ value is closer to 0. On the other hand, as the $C2$ value increases from 0.5 to 1, the throughput gain SCAR can provide over DCAR decreases as a result of better DCAR performance. As the $C2$ value decreases from 0.5 to 0, theoretically SCAR could still find the "right" route after a route-change procedure and thus should provide a higher throughput gain. However in practice, SCAR's performance also suffers. A decreasing $C2$ value usually means more potential routes and less "correct" routes. With more routes in the network, the modified RREQ-RREP packets are prone to get lost, and yielding incorrect route change decisions. So the intuition is $C2$ value being too big or too small can limit SCAR's performance over DCAR. So an adjusted $C2$ is proposed as $C2_{\text{adj}} = 2 \times |C2 - 0.5|$. $C2_{\text{adj}}$ is still in the scale of 0 to 1. Smaller values imply higher throughput gain.

We argue the following.

(1) $C2$ represents the level of confidence that DCAR will find a coding structure in a given topology. $C3$ and higher rank indicators denote the likelihood for DCAR to find multipacket coding opportunities.

(2) Adjusted $C2$, or $C2_{\text{adj}}$, represents the extent of throughput gain we can expect from choosing to use SCAR over DCAR as the routing protocol. Smaller $C2_{\text{adj}}$ implies higher throughput gain.

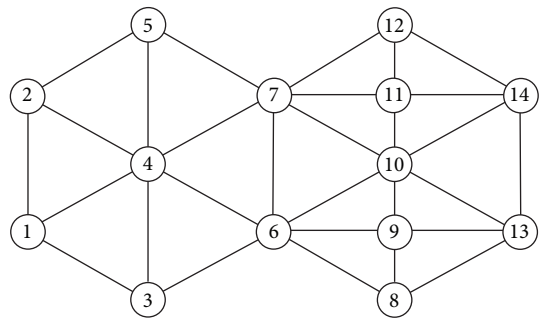


Figure 7: An artificially designed topology to measure the effectiveness of *CN* indicators.

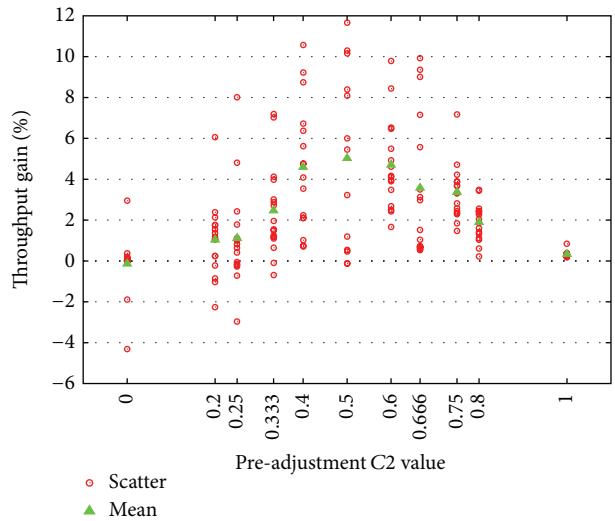


Figure 8: Scatter plot of throughput gain with different *C*2 values.

(3) Adjusted *C*3 or higher-rank indicators offer possibility of higher performance as well, but the volatility of throughput gain is higher.

We validate our argument by simulations. The objective is to test and justify the effectiveness of using *CN* indicators to predict SCAR's throughput gain against DCAR. The approach we take is, first, to prove that the *C*2 can indeed reflect SCAR's ability to improve throughput and then to verify that *C*3 and higher-rank indicators for flows can reflect SCAR's ability to discover multipacket coding opportunities. These two justifications are based on simulation results from specially designed topologies. We can assume that *CN* for the whole topology, which is an average of adjusted *CN* values over all flows, can reflect SCAR's capability to improve throughput. This assumption is justified by simulation results in the next section.

5.1. Simulation 1. With an artificially designed topology where flows with different *CN* values can be found (See Figure 7), we choose 16 flow pairs for each of the *C*2 values to be examined, namely, 0, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, and 1. For each flow pair, we measure the network throughput of both routing protocols under 6 different offered loads and 10 random seeds. The throughput when the network becomes saturated is averaged through 10 random seeds. Figure 8 is the scatter plot of throughput gain for each flow pair against the value of *C*2. The mean throughput gain for each *C*2 value is denoted as a triangle symbol in the figure. The throughput gains for both pre-adjusted C2 and adjusted C2 are averaged through the data points to yield Figure 9. From these results, we observe that a *C*2 value between 0 and 1 implies a throughput gain significantly different from zero, while a *C*2 value of zero or one implies a throughput gain near zero. It is worth noting that *C*2 being 0.5 gives the highest throughput gain. This justifies the adjustment on *C*2. The adjusted C2 values are readily comparable and thus can be averaged to get the *C*2 for the whole topology. Though the correlation between throughput gain and adjusted C2 values is not strictly linear, it is safe to assume a linear relationship in the granularity we are discussing.

5.2. Simulation 2. Topologies with a specified *C*3 value are difficult to construct. In fact, the topology shown in Figure 7 gives 30 nonzero *C*3 values, but 26 of them are 1/3. In order to test with other *C*3 values, we have to construct a series of new topologies, which are omitted here due to limited space. The throughput gain with different *C*3 values

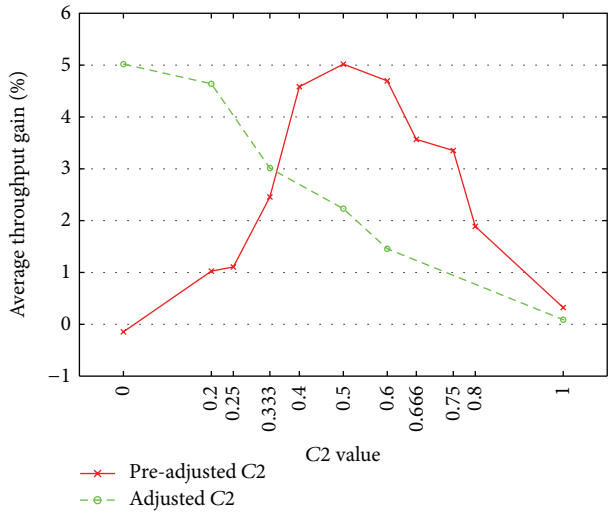


Figure 9: Average throughput gain is nearly linear to adjusted $C2$ values.

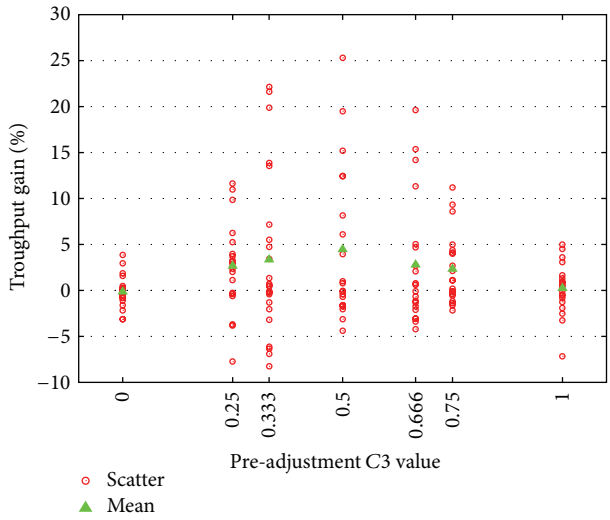


Figure 10: Scatter plot of throughput gain with different $C3$ values.

is plotted in Figure 10, with the average throughput gain shown in Figure 11. We can find some patterns within the scatter plot. The scatter plot for each nonzero and nonone $C3$ value in Figure 10 can be basically divided into two clusters. One in the ±10% throughput gain range, and the other cluster above 10%. The first cluster is roughly symmetric about the zero point and we can explain it as a normal deviation. In these cases, SCAR simply brings in a dynamic scheme for the sources to reconsider those already established routes, adding some deviations to the mean throughput. The second cluster, however, is the cases where SCAR managed to find multipacket coding possibilities. In fact, after thorough examination of the simulation log file for data points in this cluster, SCAR tends to find 3-packet coding opportunities while DCAR only finds 2-packet coding opportunities or no coding opportunities at all due to the order of flow arrivals. As a conclusion, a $C3$ value between 0 and 1 offers multipacket coding opportunities that might be discovered by SCAR.

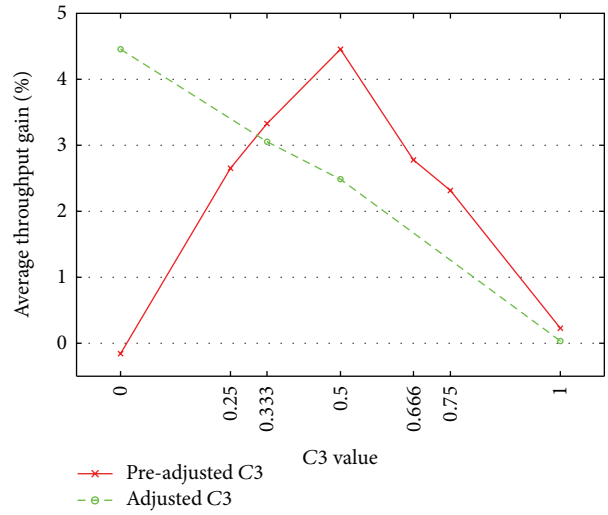


Figure 11: Average throughput gain for different $C3$ values.

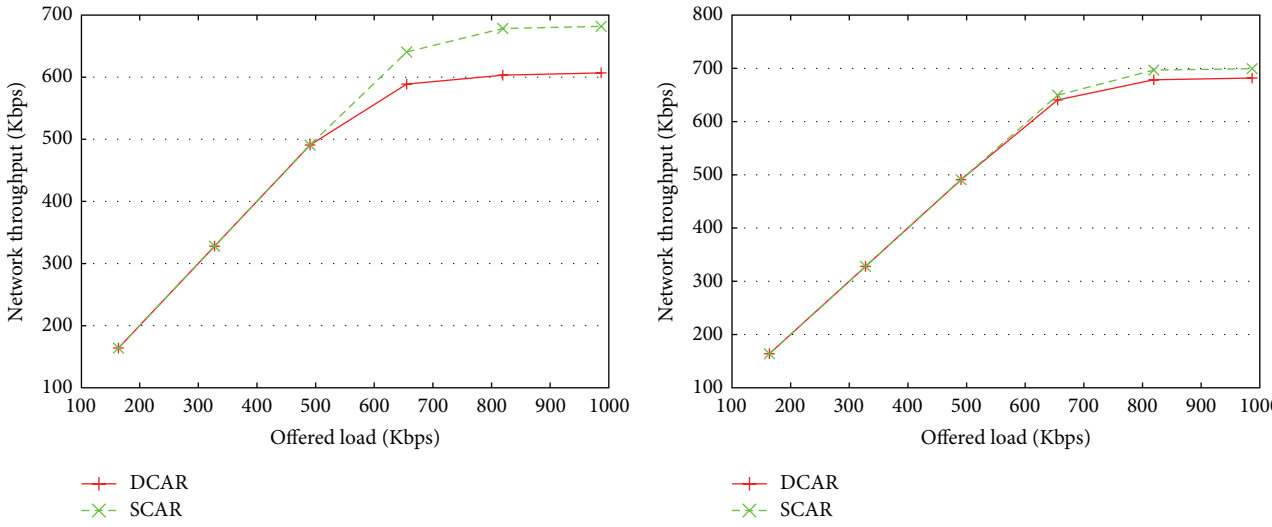
However, SCAR is not guaranteed to find them and the performance can fluctuate within a relatively large variation depending on the traffic condition. Average throughput gain under different $C3$ values is shown in Figure 11.

6. Evaluation

Three sets of simulations are done on the Qualnet (http://web.scalable-networks.com/content/qualnet/) simulator, revealing different aspects of features of our proposed protocol SCAR.

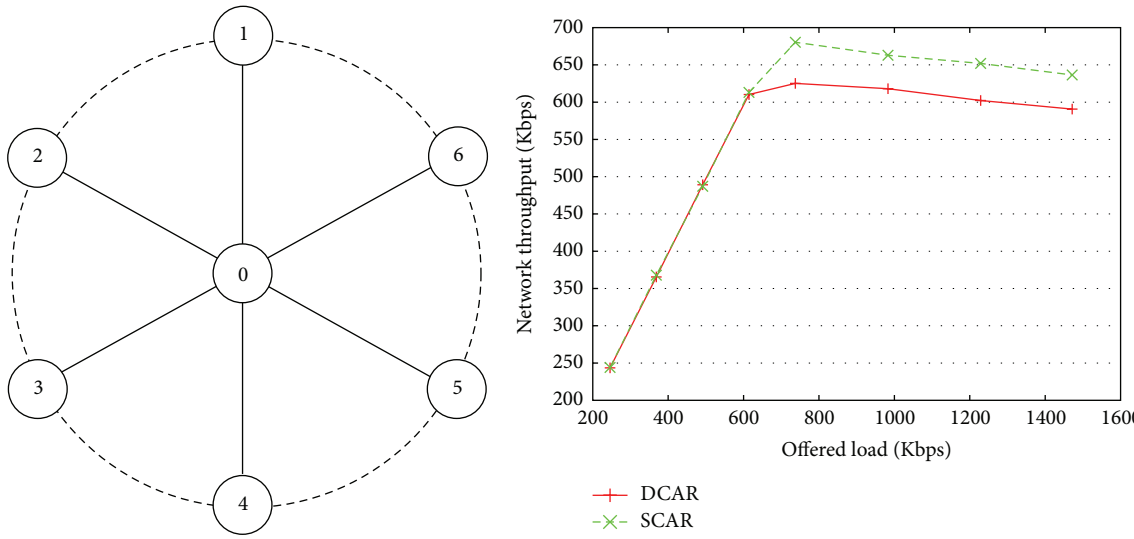
6.1. Simulation 1: Simple Topology. Using the simple scenario in Figure 2, we illustrate how DCAR is vulnerable to flow-arrival times and how our proposed protocol SCAR can withstand this variability. We start with a flow from node 1 to node 2 and, then, add the new flow from node 6 to 5. This is defined as starting order 1. The starting order is then reversed and is defined as starting order 2. The flows are given the same traffic load. For starting order 1, whether there is a coding structure depends on the route selection for flows 1-2. The paths 1-3-2 and 1-4-2 are of the same quality at the beginning. Under DCAR, if the node 1 randomly chooses the path 1-4-2, there will be no coding structure formed in the network. For starting order 2, DCAR can perfectly choose the right routes to use. In contrast, for SCAR, no matter which flow node 1 chooses at the beginning, it is able to adjust itself to form a coding structure and improves the throughput of the network. In terms of CN indicators, the flow pair (1-2) and (6-5) has a $C2$ value of 0.5, which equals an adjusted $C2$ value of 0. This implies that it is very likely to have a throughput gain with these flow pairs. The whole topology, when considering all possible flow pairs, has an adjusted $C2$ value of 0.46. There is no multipacket coding opportunity in this topology. Anyway, this topology is simply to serve as a demonstration, so we do not test with more flow pairs.

We vary the offered load and plot the end-to-end throughput in Figure 12. The two starting orders mentioned



(a) Starting order 1 (b) Starting order 2

Figure 12: Simulation results for the simple test topology.

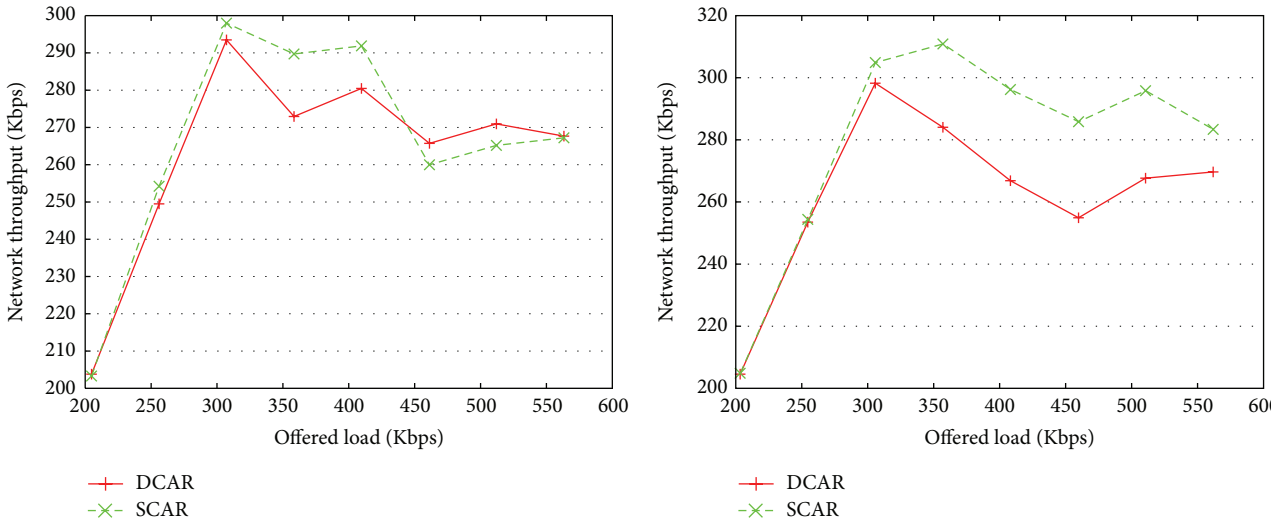


(a) "Wheel" topology (b) Average throughput of the network

Figure 13: Simulation results for "wheel" topology.

above are considered for DCAR and SCAR. Each data point in the figure is an average throughput over 10 different random seeds. We observe that SCAR could always choose the path 1-3-2 after a period of time and forms a coding structure in the network, while DCAR could only choose between the paths 1-3-2 and 1-4-2 randomly under starting order 1. The throughput gain at the highest offered load averages to 12.3%. However, under starting order 2, the performances of DCAR and SCAR are quite similar. The confidence interval, or the minimum/maximum value in the simulation, is not plotted in the figure for clearance. In fact, the minimum and maximum values are quite similar and DCAR's and SCAR's confidence intervals severely overlap. The reason why SCAR's average performance is better than DCAR's is that SCAR has higher probability of reaching the maximum value.

6.2. Simulation 2: "Wheel" Topology. The "wheel" topology as shown in Figure 13(a) is an interesting and efficient topology to study the performance of coding-aware routing protocol. A central node (0) is surrounded by six nodes (1–6) evenly distributed along the cycle. Each node can reach all the other nodes except for the node on the opposite end of the diameter (e.g., node 1 can reach all nodes except node 4 and vice versa). Each node tries to send data to the node at the opposite end of the diameter. There are many coding opportunities in this scenario as studied in [3]. In our topology analysis, this topology has an adjusted $C2$ value of 0.36 and an adjusted $C3$ value



(a) Average throughput of grid topology

(b) Average throughput of modified grid topology

FIGURE 14: Simulation results for grid topology.

of 0.33. This suggests that it is likely to see a throughput gain by exploiting the $C2$ value, while the throughput gain may not be consistent because the $C3$ value brings in more uncertainty as well as an expected throughput gain.

The average throughput of the network under varying traffic loads is plotted in Figure 13(b). When the network is saturated, the throughput gain averages to 8.02%. Further inspection into the simulation log reveals that SCAR tends to find 4-packet coding opportunities when DCAR never found one. We have also tested different starting orders and find that this does not have any influence on the performance of DCAR and SCAR. The reason is that this is a totally symmetric scenario with symmetric topology and flows.

6.3. Simulation 3: Grid Topology. Next we construct the same 4×4 topology as mentioned in [3] where each node can only reach its northern, southern, eastern, and western nodes. The simulation is run for 10 rounds and at each round, five flows (each with 2–5 hops) are randomly added into the network. The adjusted $C2$ and $C3$ values of this topology are 1, suggesting that there would be no throughput gain from switching the routing protocol. This is because, in the deduction of CN indicators, we limit the coding structure to 2-hop ones. But in this grid topology, only multihop coding structures exist. The average end-to-end throughput of the network achieved by DCAR and SCAR is shown in Figure 14(a). In this experiment, SCAR is only slightly better than DCAR because the multihop coding opportunities discovered by SCAR are prone to errors; therefore, the throughput gain is compromised. When the offer load continues to increase, the throughput gain can be negative sometimes. The average throughput gain in saturated network is 1.15%.

However, by modifying the topology only a little bit, we can achieve totally different results. By increasing the transmission range of the nodes to reach all its 8 neighboring nodes, the CN indicators change rapidly. After the modification, the adjusted $C2$ value is 0.38 suggesting high probability of throughput gain, and the adjusted $C3$ value is 0.48, suggesting an expected throughput gain with variation. The simulation results are shown in Figure 14(b) and the throughput gain is obvious. The average throughput gain reaches 9.63% without changing any flow settings.

The comparison between these two grid topology settings further justifies the effectiveness of CN indicators. $C2$ can be used as a reliable predictor for the expected throughput gain. Though $C3$ is not as reliable as $C2$ for a single run of simulation, it still identifies multipacket coding opportunities and provides an expected throughput gain across a large number of runs of simulations. Additionally, one can leverage the knowledge of CN to modify the network topology, hoping to drastically improve the overall throughput.

7. Conclusion and Future Works

In this paper, we have introduced SCAR, a coding-ware routing protocol with self-recommendation. The mechanism of self-recommendation enables the protocol to discover hidden coding opportunities that have been overlooked by other routing protocols. Whenever a new flow joins the network with new coding opportunity, prior routing decisions are revised through a route-change procedure. The route-change procedure triggered by the self-recommendation is able to adjust the biased routing metrics. Moreover, a series of indicators are introduced. These indicators are calculated using graph theory, and they reflect how much throughput gain can be expected when using our protocol under a specified topology. Testing these indicators on multiple topologies reveals that the throughput gain is not a coincidence and many practical topologies come with positive indicators. Further simulations are done in Qualnet to evaluate our protocol. It is shown that our protocol can remarkably outperform other coding-aware routing protocols and the predictions given by the indicators are generally correct. The dynamic nature of our protocol results in a higher invariability against changes in the order of flow arrivals.

There are several problems exposed in the simulations that require further investigation. For example, when two nodes are placed at a distance such that they can hear each other sometimes but cannot in other occasions, the performance of DCAR and SCAR can be severely degraded. This is because utilizing these unreliable coding opportunities can even harm other normal flows if they are coded together. In order to tackle this problem, a more sophisticated routing protocol that handles packet loss probability more than simply normalizing the routing metric should be employed. Moreover, we can also improve the protocol when transmission links suffer from abrupt but temporary changes. It is not uncommon that nodes in the network stop receiving packets for several seconds and resumes afterwards when the traffic load is high. Currently our implementation simply does not drop any links. But when to drop the link and how to drop the link can be important in some scenarios.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: practical wireless network coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, 2008.

[3] J. Le, J. C. S. Lui, and D. Chiu, "DCAR: distributed coding-aware routing in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 4, pp. 596–608, 2010.

[4] D. Johnson, D. Maltz, J. Broch et al., "DSR: the dynamic source routing protocol for multi-hop wireless ad hoc networks," in *Ad Hoc Networking*, C. E. Perkins, Ed., chapter 5, pp. 139–172, Addison-Wesley Longman, Boston, Mass, USA, 2001.

[5] Y. Yan, Z. Zhao, B. Zhang, H. T. Mouftah, and J. Ma, "Rate-adaptive coding-aware multiple path routing for wireless mesh networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '08)*, pp. 1–5, New Orleans, La, USA, November-December 2008.

[6] J. Zhang and Q. Zhang, "Cooperative network coding-aware routing for multi-rate wireless networks," in *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM '09)*, pp. 181–189, Rio de Janeiro, Brazil, April 2009.

[7] Y. Yan, B. Zhang, J. Zheng, and J. Ma, "CORE: a coding-aware opportunistic routing mechanism for wireless mesh networks," *IEEE Wireless Communications*, vol. 17, no. 3, pp. 96–103, 2010.

[8] S. Sengupta, S. Rayanchu, and S. Banerjee, "Network coding-aware routing in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1158–1170, 2010.

[9] S. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[10] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of Allerton Confer ence on Communication, Control, and Computing*, pp. 40–49, 2003.

[11] D. Lun, M. Médard, and R. Koetter, "Efficient operation of wireless packet networks using net work coding," in *Proceedings of the International Workshop on Convergent Technologies*, 2005.

[12] A. Ramamoorthy, J. Shi, and R. D. Wesel, "On the capacity of network coding for random networks," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2878–2885, 2005.

[13] Z. Li and B. Li, "Network coding: the case of multiple unicast sessions," in *Proceedings of Allerton Conference on Communications*, 2004.

[14] T. Ho and R. Koetter, "Online incremental network coding for multiple unicasts," in *Proceedings of DIMACS Working Group on Network Coding*, 2005.

[15] T. Ho, Y. Chang, and K. Han, "On constructive network coding for multiple unicasts," in *Proceedings of Allerton Conference on Communication, Control and Computing*, 2006.

[16] S. Rayanchu, S. Sen, J. Wu, S. Banerjee, and S. Sengupta, "Loss-aware network coding for unicast wireless sessions: design, implementation, and performance evaluation," in *Proceedings of ACM SIGMETRICS Performance Evaluation Review*, pp. 85–96, 2008.

[17] J. Le, J. Lui, and D. Chiu, "How many packets can we encode?—an analysis of practical wireless network coding," in *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM '08)*, pp. 371–375, 2008.

[18] T. Cui, L. Chen, and T. Ho, "Energy efficient opportunistic network coding for wireless networks," in *Proceedings of the 27the IEEE International Conference on Computer Communications (INFOCOM' 08)*, pp. 361–365, 2008.

[19] S. Sengupta, S. Rayanchu, and S. Banerjee, "An analysis of wireless network coding for unicast sessions: The case for coding-aware routing," in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM '07)*, pp. 1028–1036, Anchorage, Alaska,, May 2007.

[20] S. Das, Y. Wu, R. Chandra, and Y. Hu, "Context-based routing: technique, applications, and experience," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*, pp. 379–392, 2008.

[21] P. Chaporkar and A. Proutiere, "Adaptive network coding and scheduling for maximizing throughput in wireless networks," in *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '07)*, pp. 135–146, Montreal, Canada, September 2007.

[22] B. Scheuermann, W. Hu, and J. Crowcroft, "Near-optimal coordinated coding in wireless multihop networks," in *Proceedings of the 3rd International Conference on Emerging Networking Experiments and Technologies (CONEXT '07)*, New York, NY, USA, December 2007.

[23] T. Kim, S. Vural, I. Broustis, D. Syrivelis, S. V. Krishnamurthy, and T. F. La Porta, "A framework for joint network coding and transmission rate control in wireless networks," in *Proceedings of the 29th IEEE Inter national Conference on Computer Communications (INFOCOM '10)*, pp. 1–9, San Diego, Calif, USA, March 2010.

[24] Y. Yan, B. Zhang, H. T. Mouftah, and J. Ma, "Practical coding-aware mechanism for opportunistic routing in wireless mesh networks," in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM '07)*, pp. 2871–2876, Anchorage, Alaska, USA, May 2008.