Minimum Cost Blocking Problem in Multi-path Wireless Routing Protocols

Qi Duan, Mohit Virendra, Shambhu Upadhyaya, *Senior Member, IEEE*, Ameya Sanzgiri

Abstract—We present a class of Minimum Cost Blocking (MCB) problems in Wireless Mesh Networks (WMNs) with multi-path wireless routing protocols. We establish the provable superiority of multi-path routing protocols over conventional protocols against blocking, node-isolation and network-partitioning type attacks. In our attack model, an adversary is considered successful if he is able to capture/isolate a subset of nodes such that no more than a certain amount of traffic from source nodes reaches the gateways. Two scenarios, viz. (a) low mobility for network nodes, and (b) high degree of node mobility, are evaluated. Scenario (a) is proven to be NP-hard and scenario (b) is proven to be #P-hard for the adversary to realize the goal. Further, several approximation algorithms are presented which show that even in the best case scenario it is at least exponentially hard for the adversary to optimally succeed in such blocking-type attacks. These results are verified through simulations which demonstrate the robustness of multi-path routing protocols against such attacks. To the best of our knowledge, this is the first work that theoretically evaluates the attack-resiliency and performance of multi-path protocols with network node mobility.

Index Terms—Attacks, Blocking, Multi-path routing, Max SNP problems (MAXSNP), Wireless networks, #P-Hardness

1 Introduction

MULTI-PATH traffic scheduling and routing protocols in wired networks are deemed superior over conventional single path protocols in terms of both enhanced throughput and robustness. In wireless networks, even though the dynamic nature of networks and resource constraints entail additional overhead in maintaining and reconfiguring multiple routes, which could offset the benefits seen in wired networks, research has proven that multi-path routing provides better Quality of Service (QoS) guarantees. This paper adopts a unique approach to further assay their utility by investigating the security and robustness offered by such protocols. Specifically, we study the feasibility and impact of blocking type attacks on these protocols. In our study, Wireless Mesh Networks (WMNs) [1] are considered as the underlying representative network model. WMNs have a unique system architecture where they have nodes communicating wirelessly over multiple hops to a backbone network through multiple available network gateways. Primary traffic in WMNs is between the backbone network and stationary/mobile nodes. This architecture has led to WMNs emerging as a key component in the networking and communications domain due to their design which allow numerous diverse commercial and military applications [2], [3], [4], [5]. This uniqueness of WMNs has resulted in significant research effort being placed on designing various protocols for it. The main focus, however, is on multi-path routing schemes since efficient multi-path traffic scheduling schemes can split a node's traffic into multiple flows along several accessible gateways and eventually reassemble this traffic at the backbone network at low costs. This makes WMNs ideal candidates for applying the full scope of any wireless multi-path protocols and study the impact of these attack scenarios. Though the underlying representative network model considered for this study is WMN, the attack scenarios and results in this paper are fully portable to other types of wireless data networks which use multi-path routing protocols [6], [7], [8].

• *Authors Q. Duan, M. Virendra, were with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, 14260 at the time of doing this research and authors S. Upadhyaya and A. Sanzgiri are currently with the University at Buffalo.*
E-mail: {qiduan, virendra, shambhu, ams76}buffalo.edu

1.1 Scope, Impact and Relevance

The scope of this paper is the dependability of interconnection networks, their performance, and fault tolerance under various attack scenarios. The research reported here is largely theoretical[1] and establishes the superiority of multi-path routing protocols in the face of malicious attacks. The impact and relevance pertain to building confidence on existing schemes which primarily rely on the robustness of multi-path protocols. The impacted areas would include load balancing [9], network coding [10], [11], [12] and threshold cryptography [13], [14], in the wireless domain.

(a) Active Attack Scenarios for Recovery and Resiliency: This work is highly relevant for scenarios where it may be easier (or harder) for the adversary to compromise some nodes in the network, as compared to compromising the rest of the nodes. For example, it would usually be more difficult (in terms of cost) to block nodes closer to the gateways or Base Stations (BS) due to reasons of physical proximity (physically better guarded), or

1. A wireless mesh network is represented by an undirected graph and all analyses are based on graph theory.

signal strength (nodes closer to BS may have better received signal strength). Similarly, it is highly desirable for protocols to continue to execute correctly without information compromise, even in the presence of a few malicious nodes. Currently, most security protocols do not address recovery from malicious behavior. Protocols simply abort execution and restart if any malicious behavior is detected. This is detrimental especially in applications where real-time response and high level security are important as information may have already been lost in the partial execution and frequent restart of the protocols.

(b) Relevance and Impact on Existing Protocols: Multi-path routing protocols can naturally extend threshold cryptography concepts to the wireless domain. Demonstrated robustness of multi-path protocols against such blocking-type attacks would increase confidence in utilizing threshold cryptography schemes [15], [16]. In threshold cryptography, a node splits a secret into several shares, routes them along independent paths, and a threshold number of shares have to be compromised (at least) for an adversary to recover the secret. Our results imply that it would be at least exponentially hard for an adversary to optimally compromise or block certain threshold number of shares such that either the adversary recovers the secret, or equivalently, the secret is not recovered properly at the destination. Network coding, where nodes intelligently send redundant information along multiple paths to ensure security and reliability and to detect any problems with a route would also benefit from such demonstrated robustness of multi-path routing. Again, it would be at least exponentially hard for the adversary to optimally compromise more than a threshold number of these paths to render such network coding schemes ineffective.

1.2 Contributions

While there has been some work on integrating the benefits provided by multi-path routing protocols with security mechanisms [17], [18], [19], there exists a gap in analyzing multi-path routing attacks. Specifically two areas that need to be analyzed are: (a) The performance in terms of security and resiliency of *mobile* wireless networks multi-path protocols under different attack scenarios, and (b) Comparison with traditional single-path protocols under such circumstances. This paper attempts to achieve the above two desirable goals. To the best of our knowledge, this is the first paper to theoretically evaluate the performance of wireless network multi-path protocols considering node mobility under attack scenarios. The technical contributions of this paper are:

- The identification of the Minimum Cost Blocking (MCB) problem. Though we consider MCB in the WMN setting, the problem is applicable to other wireless or wired networks.
- Evaluating the hardness of the problem. MCB is NP-hard for the low/no node mobility scenario and $\#P$-hard for networks with *patterned* node mobility. The reduction for no-mobility is derived from the basic Set Cover problem [20] and for mobility scenario, from the $3SAT$ [21] and $\#SAT$ [22] problems.
- Development of approximation algorithms for the best case scenario and the performance testing of these algorithms in different settings through random graphs based experiments.
- Laying direction for future research to evaluate the performance of multi-path protocols against sophisticated attacks in mobile wireless networks.

1.3 Paper Organization

The rest of this paper is organized as follows. Section 2 discusses motivation and related work. Section 3 illustrates our attack model and the MCB problem. Section 4 introduces a particular case of multi-node MCB problem and analyzes its complexity. Section 5 presents the main results on the MCB problem by proving its NP-hardness. Section 6 provides two approximation algorithms for the MCB problem. Section 7 presents simulation results for the algorithms for stationary MCB. Section 8 introduces the $\#P$-hard Blocking problem for WMNs with patterned node mobility. Section 9 concludes the paper with a discussion on future research directions.

2 Motivation and Related work

Multi-path routing protocols unlike standard routing protocols intend to discover multiple paths between a source and a destination node. Their utility lies in compensating for the dynamic and unpredictable nature of networks. Specifically, the multiple paths provide load balancing, fault tolerance and higher aggregate bandwidth. It has been proven that using multi-path routing in dense networks enhances performance and result in better throughput than unipath routing [23]. Traditionally, multi-path routing has been in the context of WMNs. But recently, there has been progress in adapting these protocols to other types of networks such as WSNs (Wireless Sensor Networks) [24], [25], [26], [8].

The two main components of multi-path routing are discovering routes and then maintaining these routes based on certain metrics. Examples of such metrics include Estimated Transmission Count (ETX) [27], Expected Transmission Time (ETT) [28], etc. The authors of [29] present a new multi-path routing protocol for heterogeneous networks where they choose QoS as a routing metric. However, it is important to note that unlike unipath routing, multi-path routing metrics are aggregate in nature, i.e., paths at each hop are chosen to maximize/minimize the *sum* of the individual paths at each hop and not choose the best path each hop. To reiterate, since multi-path routing protocols are intended to increase (decrease) say aggregate bandwidth (end to end delay, for instance), the routes selected by these protocols need to facilitate it. This implies that such

routes need to be disjoint (not have any common nodes or links) to increase fault tolerance, since the failure of a single node/link can cripple the entire network and be detrimental to the multi-path routing philosophy. However, the cost for discovering such routes is expensive in terms of both time and resources. Further, because of the nature of networks, non-disjoint routes are more abundant [23]. Additionally, node-disjointness (no common node between two paths) is a stricter requirement than even link-disjointness (no common link between two paths), making them least abundant and thus, hardest to find. Due to these practical considerations, in most multi-path routing, more often than not non-disjoint routes are selected. This causes a huge security risk, since the compromise of such paths could effectively partition the network. While such a problem does arise with even uni-path routing because of the aggregate nature of metrics in multi-path routing, it is more severe in multi-path routing. Another interesting point of multi-path routing is that while it might ensure failure independence, nodes belonging to different paths might still be in the transmission range of each other causing interference with each other. Such routes would then cause more harm than benefit as they would have to wait for the transmission medium to be free and thus be unable to perform concurrent transmissions. This presents a unique opportunity to an attacker who can use such nodes to partition a network. Even though most routing protocols try to choose paths that are as *transmission independent* as possible to ensure the least interference between routes, it is not always possible to do so due to network topologies and mobility. Thus, despite their inherent advantages, the innate natural disadvantages make multi-path routing protocols an attractive target for attacks. This has led to a focus on security in multi-path routing protocols. Much of this focus is on either information eavesdropping or optimizing security mechanisms for multi-path routing. Some of these attacks can be prevented or countered through cryptographic techniques. For example, OSPF [30], [31] uses MD5 [32] to guard against false packet injection. Digitally signed statements can also be used in OSPF to prevent false advertisement by legitimate users. In the wireless network domain, such cryptographic schemes for secure broadcast and false data injection prevention are described in [33], [34], [35]. Recently, frameworks have been designed to insulate against information eavesdropping in routing protocols, without compromising on performance [19]. This work presents a formulation of a game that integrates metrics of multi-path routing with security, based on which a system administration can incorporate one or more metrics of multi-path routing protocols. Other works such as [36] present routing protocol based on secret sharing over multiple paths. The authors of [18] present a routing protocol that is designed to prevent adversaries from overhearing information and focuses on node-anonymity to prevent identification of end nodes, by forwarding nodes. However, there are other attacks that cannot be countered through cryptographic techniques. Link cut attacks in wired networks, first investigated in detail in [37], are one such type of attack. In wireless networks, link cuts can be achieved through jamming or interference [38]. In reality, blocking a certain link in a wireless network usually means blocking all signals from a certain node or compromising the node completely. As mentioned above this may be relatively easy to achieve for wireless nodes deployed in automated, unattended or hostile scenarios, accentuating the need for research on blocking attacks – an aspect that has been ignored by the aforementioned works. We adopt some computation complexity related techniques to analyze this particular aspect in multi-path routing security. Specifically, we use techniques related to the basic set cover and partial set cover problems. The basic set cover problem is NP-hard and extensive research has been done on its approximation algorithms [39], [40]. A generalization of the set cover problem is the partial set cover problem detailed in [41], [42]. The complexity class $\#P$ was first introduced in [43]. Sociological orbits in wireless networks utilized in describing the node-mobility scenario were introduced in [44].

3 Assumptions and Threat Model

3.1 Assumptions

The network and the threat model in this paper conform with the following conditions:

1) We consider managed networks where each node has a unique identity. In other words, the mapping between network nodes and their identities remains one-to-one, a property that can be verified in any managed network. This will preclude node replication attacks.
2) The attacker while having the resources cannot deploy his own devices (nodes) to the network.
3) The adversary is a global adversary in the sense that the adversary wants to severe the network and can choose the way the network is to be severed. By this we mean that he is not limited to any particular localized area in the network.
4) Physical capture of nodes is allowed; there exists a cost for each capture/compromise of nodes which is assumed to be computable for the sake of simplicity.
5) An attacker can also compromise nodes, however, he does not control certain elements such as mobility of the nodes or modification/addition of the hardware of the captured nodes. This assumption is perfectly legitimate since our model considers that the attacker does not know all the details of the network and it will exponentially increase the cost of gathering these details.
6) Although the attacker may have a fair knowledge of the workings of any system especially in wireless mesh networks, we do not explicitly consider insider attacks. Insider attacks are possible in any

organization's system or networks. However, they are also complex in the sense that there are possibly many ways an insider attack can be staged. Consideration of insider attacks and its analysis will be quite involved, since there will be too many parameters to consider and hence is outside the scope of this paper.

3.2 Threat Model

Blocking, node-isolation and network-partitioning type attacks are easy to launch and are effective in the wireless networks domain due to channel constraints and dynamic network topologies. We emulate adversarial behavior by attacking the multi-path schemes through intelligent blocking and node-isolation type attacks and study the impact. We also try to design best-case scenarios for these attacks to succeed. Both low node-mobility and high node-mobility scenarios are considered. For comparison purposes, we also launch similar attacks on conventional single-path protocols and measure their impact. The minimum cost blocking (MCB) problem can be stated as a special case of node blocking in a network at minimum cost to the attacker. Here the attacker wants to partition the network, thus ceasing flow of data, by either capturing and blocking a key node or by routing all data through a particular node. As we consider multi-path routing protocols, the attacker has to consider the operation of multi-path routing since multiple paths will exist from the source to the destination. While a non-trivial but easy solution is to launch a blackhole [45] or wormhole [46] attack, this would force the attacker to deploy his own nodes or capture a node close to the destination/source which would increase his attack cost due to the nodes' close proximity to base stations. In a blackhole attack, a particular node in a network falsely advertises a route (based on metrics specific to the protocol) to the destination node so as to force the route discovery algorithm to choose a route through it. The actual blackhole attack occurs when the malicious node drops packets and hence blocks paths to the destination. Similarly, in a wormhole attack, an attacker records packets (or bits) at one location in the network, tunnels them (possibly selectively) to another location, and retransmits them into the network. However, it has to be also noted that multi-path routing is not necessarily affected by wormhole attacks [47]. For these reasons and for stated assumptions in Sec. 3.1, we do not consider blackhole and wormhole attacks explicitly in this paper. Further, sybil attack [48], [49] where a node can be assigned multiple identities is precluded from our threat model since the focus of this paper is primarily the blocking attack.

4 A Multi-node MCB Case in Wireless Networks

The general problem of blocking possible traffic flow between a pair of vertices in a connected graph is known as the max-flow min-cut problem [22], which can be solved in polynomial time for both cases of minimum edge cut and minimum node cut. In this section, we first consider a particular case of blocking between a pair of nodes in wireless networks. This, if viable, is a simple attack that could bring down the network. Consider a situation where the adversary has already compromised a set of nodes in the network.[2] The adversary can now stage an attack by blocking some nodes in the network such that all traffic between a certain pair of nodes will pass through at least one of the compromised nodes. Though this is conceivable, we show that it is NP-hard to find the minimum cost set of nodes so that all traffic between the source destination pair will pass through one of the compromised nodes. In the most simple case, we have a source destination pair s_1 and t. Another node (called C) is compromised which is *only* connected to the destination node t and another node s_2. We need to find a minimum node cut to separate s_1 and t in the graph with the node C removed. The minimum cut has the following property: it will separate node t from nodes s_1 and s_2, at the same time, keep nodes s_1 and s_2 connected. In this case, the cut will cause all traffic flow from s_1 to t to pass through C. The formal problem definition is as follows (decision version):

Definition 4.1: (3-node Induced Flow MCB). Suppose we have an undirected graph $G = (V, E)$, where $|V| = n$, and every node $v_i \in V$, $1 \leq i \leq n$, has an associated positive integer cost c_i.[3] Given three nodes s_1, s_2, t, and an integer b can we find a set of nodes in V, such that the total cost of nodes in V is no more than b, and removal of all nodes in this set will separate t from s_2 and s_1, at the same time, keep s_2 and s_1 connected?

Next we show that even in the case where every node has unit cost, the problem is NP-complete.

Theorem 4.2: The 3-node Induced Flow MCB is NP-complete even if every node has a unit cost.

Proof: We prove this result by reducing the MAX2SAT problem to this problem. Given an instance of MAX2SAT with m variables, r clauses, and integer value k, we can construct a two-layer graph as follows: the first layer has two end points s_1 and s_2, between which are pairs of variable nodes (see Figure 1). In the second layer, node t is connected to all intermediate nodes. All the nodes represented in thick dots in the figure are cliques. In the first layer, every thick node is a clique of size $(m + r)$. In the second layer, every thick node is a clique of size $(m + r)^2$, and any neighboring node of the thick node is connected to every node in the clique. The two layers are connected as follows: the two variable nodes corresponding to a variable and its negation in another layer are connected, and for every clause in the MAX2SAT instance, we connect the first variable in the first layer to the second variable in the

2. By compromise we mean that the adversary is in control of the node.

3. There is always a cost associated with compromising a node which we denote as cost of compromise.

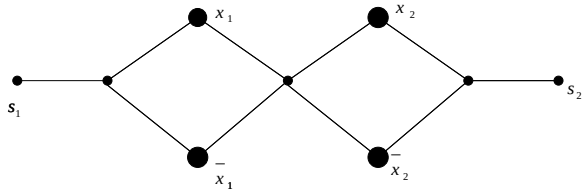

Fig. 1. The first layer of the constructed instance

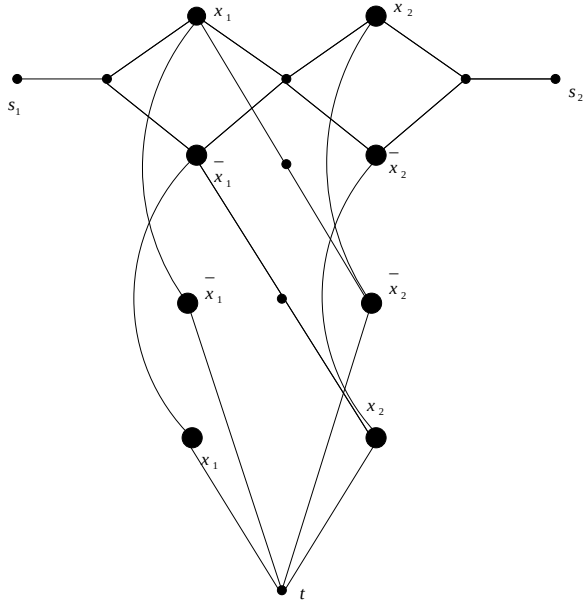

Fig. 2. The constructed instance of 3-node Induced Flow MCB

second layer through an intermediate node (we call the resulting edges as clause edges). Figure 2 is the graph constructed for the instance $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$.

We set b to be $m(m+r)^2 + m(m+r) + r - k$.

We have the following observations:

1) Since s_1 and s_2 must be connected, for every variable node pair in the first layer, a variable and its negation cannot be chosen in the cut simultaneously.
2) Since s_1 and s_2 must be separated from t, one of the two appearances (in the two layers) of every variable must be chosen in the cut.
3) Since the variable node in the second layer has clique size $(m+r)^2$, then for every variable and its negation in the second layer, only one of them can be chosen in the cut.

From the observations, we can conclude that for every variable, one must choose it or its negation (but not both) in both layers. So, the cost of the chosen variable nodes will be $m(m+r)^2 + m(m+r)$. If the original MAX2SAT has an assignment that can satisfy k clauses, then we can choose the intermediate node of the unsatisfied clause edges, and the variables in the truth assignment (in both layers). We can then find a cut no more than $m(m+r)^2 + m(m+r) + r - k$, such that the conditions are satisfied. Conversely, if a cut of no more than $m(m+r)^2 + m(m+r) + r - k$ can be found, then an assignment can be found (according to the cut) to satisfy at least k clauses. □

Similarly, we can define a multi-node induced flow MCB, in which we have $u + v$ nodes $A_1, \ldots, A_u, B_1, \ldots, B_v$ in the graph, and we would like to find the minimum cut that can separate $A_1, \ldots, A_u$ from $B_1, \ldots, B_v$, and at the same time, keep $A_1, \ldots, A_u$ connected and $B_1, \ldots, B_v$ also connected. Obviously, the 3-node Induced Flow MCB is a special case of the multi-node Induced Flow MCB.

We can show that multi-node Induced Flow MCB (where the number of nodes is not fixed) is MAXSNP-hard.

Theorem 4.3: Multi-node Induced Flow MCB is MAXSNP-hard.

Proof: We can use a similar reduction as in the proof of the NP-hardness of 3-node Induced Flow MCB. Given an instance of MAX2SAT with m variables, we construct an instance of multi-node Induced Flow MCB, which is similar to the instance constructed in the proof of the NP-hardness of 3-node Induced Flow MCB. In the constructed instance of multi-node Induced Flow MCB, we have nodes $A_1, \ldots, A_u$, and $B_1, \ldots, B_v$, where we need to find a cut to separate $A_1, \ldots, A_u$ from $B_1, \ldots, B_v$, at the same time, keep all nodes in $A_1, \ldots, A_u$ connected and all nodes in $B_1, \ldots, B_v$ connected. In the constructed graph, we also have two layers, but every layer is similar to the first layer in our construction in the proof of NP-hardness of the 3-node Induced Flow MCB. Node A_i $(1 \leq i \leq u)$ is connected to the two variable nodes x_i and $\overline{x_i}$ in the first layer. Node B_i $(1 \leq i \leq v)$ is connected to the two variable nodes x_i and $\overline{x_i}$ in the second layer. For every clause, we also construct the clause edges with an intermediate node (same as that in the proof of NP-hardness of 3-node Induced Flow MCB). In the constructed graph, every node has cost 1. We set the bound b to be $2m + r - k$. Figure 3 is the graph constructed for the instance $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$.

It is easy to see, since we need to keep $A_1, \ldots, A_u$ connected and $B_1, \ldots, B_v$ connected, that for every variable, one must choose to block the variable or its negation (but not both) in both layers. So we can see that the MAX2SAT instance (denoted as I) has an assignment which satisfies at least k clauses if and only if the constructed multi-node Induced Flow MCB instance (denoted as I_1) has a blocking cost at most b. Suppose the optimal solution of the MAX2SAT instance is $OPT(2SAT)$. Then the optimal solution of the corresponding multi-node Induced Flow MCB is $OPT(MCB)$. The cost of the solution found for the constructed multi-node Induced Flow MCB instance is $c(I_1)$. The cost of the corresponding solution of the original MAX2SAT instance is $c(I)$ and we have $OPT(2SAT) \geq 3r/4$. We can also assume that every variable of the MAX2SAT should appear in at least one of the clauses,

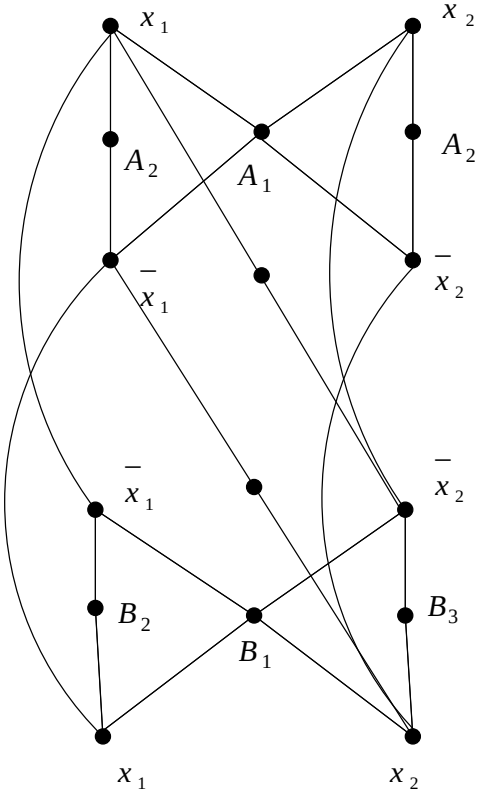

Fig. 3. The constructed instance of multi-node Induced Flow MCB

then we have $r \geq m/2$. Now we have

$$OPT(MCB) \leq 2m + \frac{r}{4} \leq \frac{17}{3} OPT(2SAT)$$

$$c(I_1) - OPT(MCB) \leq OPT(2SAT) - c(I)$$

This means the reduction is an L-reduction [22], and consequently, multi-node Induced Flow MCB is MAXSNP-hard.

□

We also present an approximation algorithm for the 3-node Induced Flow MCB. The idea is to use linear programming (LP) formulation.

Algorithm 4.4: Step 1: Solve the following LP L_1:

$$\text{Minimize} \sum_{u \in V} c_u q_u$$

subject to

$$\sum_{\text{all neighbors v of u}} f_{uv} = \sum_{\text{all neighbors v of u}} f_{vu}, \forall u \in V, u \neq s_1, s_2 \quad (1)$$

$$\sum_{\text{all neighbors v of } s_1} f_{s_1 v} = 1 \quad (2)$$

$$\sum_{\text{all neighbors v of } s_2} f_{v s_2} = 1 \quad (3)$$

$$0 \leq f_{uv} \leq 1, \forall u, v \in V$$

$$y_{s_1,u} \leq q_u + y_{s_1,v}, \forall (u,v) \in E \quad (4)$$

$$y_{s_2,u} \leq q_u + y_{s_2,v}, \forall (u,v) \in E \quad (5)$$

$$q_{s_2} = q_{s_1} = q_t = y_{s_1,s_1} = y_{s_2,s_2} = 0 \quad (6)$$

$$y_{s_1,t} = y_{s_2,t} = 1 \quad (7)$$

$$\sum_{\text{all neighbors v of u}} f_{uv} + y_{s_1,u} \leq 1 \; u \; inV, u \neq s_1, s_2 \quad (8)$$

$$\sum_{\text{all neighbors v of u}} f_{uv} + y_{t_2,u} \leq 1 \; u \; inV, u \neq s_1, s_2 \quad (9)$$

$$0 \leq y_{s_1,u} \leq 1, \; 0 \leq y_{s_2,u} \leq 1, \; 0 \leq q_u \leq 1, \forall u \in V \quad (10)$$

Here q_u is a label we assign for every node u. Equation (1) guarantees that every node has a balanced flow, and the total flow from s_1 to s_2 is 1. Inequalities (4), (5), (6), (7) guarantee that in every path from s_1 (or s_2) to t, the summation of all labels q_u along this path will be at least 1. Inequalities (8) and (9) mean that if a node is labeled, then no flow should pass through it (if L_1 has integer solution, this can be guaranteed).

Step 2. Find a path from s_1 to s_2, which satisfies the following condition: for every node u in the path, there is a flow of size at least $1/(n-3)$ passing through u. This can be done because in the above LP, we find a fractional flow of size 1 from s_1 to s_2.

Step 3. Change the cost of all nodes in the identified path in Step 2 to infinity, and add a new node s, which is connected only to s_1 and s_2. Then, find a minimum cut from s to t, and take this cut as the solution of the problem.

Theorem 4.5: The algorithm described above achieves a ratio of $n-3$.

Proof: Suppose the solution of L_1 (if one exists) is C_1. The optimal solution is OPT and the solution found by the algorithm is C_2. It is easy to see $C_1 \leq OPT$, and every path P from s_1 (or s_2) to t must have a summation of label values at least 1 so that we have

$$\sum_{\text{u:flow passes u=0}} q_u \geq \frac{1}{n-3}$$

Then

$$C_2 \leq (n-3) \sum_{u \in V} q_u \leq (n-3) C_1 \leq (n-3) OPT$$

□

If every node has unit cost, then we can modify the above algorithm slightly to get a second algorithm, where all the steps are unchanged except in the LP

formulation. The inequalities (8) and (9) are changed respectively to

$$\sum_{\text{all neighbors v of u}} f_{uv} + 2y_{s_1,u} \leq 1,\ u \in V, u \neq s_1, s_2$$

$$\sum_{\text{all neighbors v of u}} f_{uv} + 2y_{t_2,u} \leq 1,\ u \in V, u \neq s_1, s_2$$

We have the following result.

Theorem 4.6: The modified algorithm achieves a ratio of $d+1$, where $d = min(d_{s_1}, d_t) + min(d_{s_2}, d_t)$. Here d_{s_1}, d_{s_2}, and d_t are degrees of s_1, s_2, and t, respectively.

Proof: First we consider the following LP (denoted as L_2).

$$\text{Minimize} \sum_{u \in V} c_u q_u$$

subject to:

$$y_{s_1,u} \leq q_u + y_{s_1,v},\ \forall (u,v) \in E$$

$$y_{s_2,u} \leq q_u + y_{s_2,v},\ \forall (u,v) \in E$$

$$q_{s_2} = q_{s_1} = q_t = y_{s_1,s_1} = y_{s_2,s_2} = 0$$

$$y_{s_1,t} = y_{s_2,t} = 1$$

$$0 \leq y_{s_1,u} \leq 1,\ 0 \leq y_{s_2,u} \leq 1,\ 0 \leq q_u \leq 1/2,\ \forall u \in V$$

Suppose the solution of L_2 is C_3 and the solution of the modified algorithm is C_4. Then we can see that $C_3 \leq (d+1)OPT/2$ because for every node in the cut of the optimum solution of the original problem, if we label it as $1/2$, then to guarantee that every path from s_1 to t has total label value 1, we could label $min(d_{s_1}, d_t)$ more nodes with label $1/2$. Similarly, to guarantee every path from s_2 to t has total lablel value 1, we can label $min(d_{s_2}, d_t)$ more nodes with label value $1/2$. So we have $C_3 \leq (d+1)OPT/2$. Then

$$C_4 \leq 2C_3 \leq (d+1)OPT$$

□

5 Multi-path MCB Problem

We now present the Multi-path MCB problem for the stationary-nodes/low-mobility scenario. The network is modeled as an undirected graph G, with vertex set V and edge set E. Here, every vertex represents a node in the network and a link between two vertices implies that corresponding nodes are within each other's radio range. A directed graph may better represent the network for real-world situations since nodes may have different radio ranges, signal strength may be different in each direction, and links may not be completely bidirectional. However for simplifying the problem description we assume an undirected graph, emphasizing that all our results are equally applicable to the general case of directed graphs.

5.1 Multi-path MCB Optimization Problem

Suppose that in the graph $G(V,E)$, $|V| = k$. Every node v_i in V is associated with a cost c_i which is the cost of compromising the node. There are $m = \sum_{i=1}^{k} n_i$ paths $P_{11}, \ldots, P_{1n_1}, \ldots, P_{k1}, \ldots, P_{kn_k}$. Here, $P_{i1}, \ldots, P_{in_i}$ ($i = 1, \ldots, k$), are paths originating from node i (or equivalently, paths belonging to node i). What is the minimum cost to compromise a subset of nodes such that a certain percentage of paths belonging to a node are compromised? That is, for every node i ($i = 1, \ldots, k$), what is the minimum cost to compromise at least R_i ($0 \leq R_i \leq n_i$) paths out of all paths belonging to this node (i.e., paths $P_{i1}, \ldots, P_{in_i}$). This is a typical optimization problem. The corresponding decision problem is described below.

5.2 Multi-path MCB Decision Problem

Given: Graph $G(V,E)$, where every node v_i in V has a cost c_i of compromise, the set of nodes in paths $P_{11}, \ldots, P_{1n_1}, \ldots, P_{k1}, \ldots, P_{kn_k}$ and integers C and R_i ($0 \leq R_i \leq n_i$).

Statement: Is there a subset V' of V such that compromising V' will block at least R_i paths out of $P_{i1}, \ldots, P_{in_i}$, for every node v_i ($i = 1, \ldots, k$), and the total cost of nodes in V' is no greater than C?

In reality, the adversary may not need to block all the nodes in a network. However, since our description and algorithms apply to the general case of blocking traffic from a subset of nodes, we can simply let all paths related to nodes not in the target subset to be empty. It is easy to show that the problem is NP-complete.

Theorem 5.1: The MCB decision problem is NP-complete.

Proof: The problem is a general case of the partial set cover problem [20], which is a well known NP-complete problem. So multi-path MCB is NP-complete. □

6 Approximation Algorithms for Multi-path MCB, No Mobility

In this section we present two algorithms for the MCB problem with stationary nodes. The first one is a greedy algorithm and the second one LP-based. We derive the approximation ratio for both of them. We first define the notion of "cover"which will be used frequently in later discussions and then list some notations needed to describe the algorithms.

Definition 6.1: When a node (or a node within a subset of nodes) is on a path, we say that the node (or the subset of nodes) covers that path. When R_i paths belonging to a node i are covered, we say that node i is covered.

6.1 Notations

T: The set of nodes that have been chosen at the beginning of an iteration (an iteration includes all sub-steps of Step 2 in Algorithm 6.2).

E_i: Effective number of node i, or the number of effective paths the node i will cover in the current iteration of the algorithm. An effective path means that the path has not been covered yet and the corresponding target node to which that path belongs has not been blocked yet.

W_{ij}: Number of paths that belong to node j and are covered by node i.

Y_j: Number of already covered paths that belong to node j.

α_i: Cost-effective index of node i.

D: Set of nodes currently covered (used in Algorithm 6.4).

O_i: Number of paths belonging to node i covered by the set of nodes returned by the function call SetCover (used in Algorithm 6.4).

6.2 The Greedy Algorithm and Approximation Ratio

Our first algorithm, a greedy one, selects the most cost-effective node iteratively and at the same time removes the covered paths and the paths unusable in the future. Unusable paths are those originating from a node i with at least R_i paths already blocked, as covering these paths would be inconsequential.

The algorithm runs until the nodes in T have covered the required paths for all the nodes in V, i.e., T covers at least R_i paths for node i, where $i = 1, \ldots, k$. This condition is termed as "Done."

Algorithm 6.2:

1. $T \leftarrow \phi$, and mark all paths and nodes as uncovered;

2. **While** not *Done*, iterate the following sub-steps:

2.1. **For** every remaining node in $V \backslash T$, say, node i, in the current iteration, compute its effective number E_i as follows:

$$E_i \leftarrow 0$$

2.1.1. **For** every node j that is not covered yet, compute $min(max((R_j - Y_j), 0), W_{ij})$. Update E_i as follows:

$$E_i = E_i + min(max((R_j - Y_j), 0), W_{ij})$$

2.2. Compute the cost-effective index α_i as follows:

$$\alpha_i = \frac{c_i}{E_i}$$

2.3. Choose node u with the lowest cost-effective index (α_u); Mark every path node u covers as covered; For every effective path p that node u covers, set the price of the effective path, i.e., price(p) = α_u; Iterate through all the currently uncovered nodes; Mark those nodes that have been covered by node u in this iteration as covered; Add node u to T, i.e.,

$$T \leftarrow T \cup u$$

3. Output T;

Note that in Step 2.1.1 of Algorithm 6.2, W_{ij} is the number of paths that belong to node j and are covered by node i, Y_j is the number of already covered paths that belong to node j. Thus $min(max((R_j - Y_j), 0), W_{ij})$ is essentially the number of effective (or useful) uncovered paths that belong to node j and are covered by node i.

Next we show that Algorithm 6.2 achieves an approximation ratio of $\ln R$, where $R = \sum_{i=1}^{k} R_i$.

Theorem 6.3: Algorithm 6.2 achieves an approximation ratio of $\ln R$.

Proof: The proof is similar to the proof for the ratio of the greedy algorithm for set cover problem in [50]. Suppose the optimum solution has a cost OPT. We number the covered effective paths in the algorithm in the order in which they are covered, and name them as $P_1, \ldots, P_R$. In every iteration in the algorithm, the new optimal solution (selected from $V \backslash T$) that covers the remaining nodes (that are not covered yet) has a cost at most OPT. Among them, there must be one node that has cost-effective index at most OPT/U, where U is the number of uncovered effective paths (otherwise the optimum solution will have a cost greater than OPT). In the iteration that covers path P_j, there are at least $R-j+1$ paths not yet covered. Because we choose the node with lowest cost-effective index, we have $price(P_j) \leq \frac{OPT}{R-j+1}$. The total cost of our algorithm will be

$$\begin{aligned} \sum_{j=1}^{R} price(P_j) &\leq (1 + \frac{1}{2} + \ldots + \frac{1}{R}) \times OPT \\ &\leq OPT \times \ln R \end{aligned}$$

□

If we adopt the algorithm $SetCover$ for partial set cover in [51], which is based on LP relaxation, then we get a new algorithm which is described next.

6.3 The LP Algorithm and Approximation Ratio

The LP Algorithm uses a function $SetCover(P, V \backslash T, c, R_j)$, where P is the set of all uncovered paths belonging to node j, c is the array of cost values for nodes in $V \backslash T$ (i.e., $c_j, \forall j \in V \backslash T$). The function $SetCover$ returns the selected sets (nodes) that cover at least R_j paths in P.

Algorithm 6.4:

1. $T \leftarrow \phi$, $D \leftarrow \phi$

2. While D does not contain all nodes in the graph, iterate the following sub-steps:

2.1. Choose node j with the highest R_j value; Call $SetCover(P, V \backslash T, c, R_j)$;

2.2. $D \leftarrow D \cup j$

2.3. **For** every node returned by the function, $T \leftarrow T \cup i$

2.4. Remove from P, every path that is covered by the nodes returned by the function call $SetCover$; $P \leftarrow P \backslash p$

2.5. **For** every $i \in V \backslash D$, adjust R_i as follows: $R_i = max(0, R_i - O_i)$; If R_i becomes 0 (it means that node i is blocked); $D \leftarrow D \cup i$

3. Output T.

Algorithm 6.4 repeatedly blocks a node in every iteration (Step 2), until all nodes are blocked. Note that in Step 2.5 of Algorithm 6.4, O_i is the number of paths belonging to node i that were covered by the set of nodes returned by $SetCover$.

Theorem 6.5: Algorithm 6.4 achieves an approximation ratio of $h \times k$, where h is the length (number of nodes in the path) of the longest path.

Proof: The approximation ratio of algorithm SetCover is h [51]. Obviously at every iteration the sum of the cost of selected nodes $< h \times OPT$, so the total cost of the solution returned by Algorithm 6.4 is $\leq h \times k \times OPT$. □

6.4 Approximation Ratios: Practical Significance

The approximation ratios obtained above are coarse performance measures for the algorithms. It is difficult to compare the two, i.e., the values $h \times k$ and $\ln R$, since they depend on specific problem instances. Also, these ratios are far from tight because precise analysis is very difficult. It is an open research issue to determine if any better algorithms (algorithms with guaranteed better ratios) exist. We evaluate the performance of these algorithms in the next section by conducting experiments on randomly generated graphs.

In a practical setting, if the graph (network) is sparse and the topology is known to the adversary, it would be easier for the adversary to successfully launch such blocking attacks. If the graph is dense, then launching an effective attack would be more difficult. From a protocol security and resiliency point of view, it would be ideal if the network topology information is hidden from the adversary, making it extremely hard to launch such attacks. However, in practice, complete topology obfuscation is not necessary. If the adversary has partial topological information, the above algorithms cannot be executed correctly. Thus, even partial topology obfuscation can be a significant deterrent against the full scope of such attacks. This provides motivation for introducing network node mobility where exact network topology is never accurately known. We extend our results to WMNs with node mobility in Section 8.

7 Experimental Validation

We evaluated the performance of the two low/no mobility multi-path MCB algorithms using randomly generated graphs that represent arbitrary wireless mesh networks. For convenience, we have denoted these algorithms as Greedy Algorithm (Algorithm 6.2) and LP Algorithm (Algorithm 6.4). For comparison purposes, we also evaluated the performance of a greedy algorithm for MCB in a single-path scheme. The single-path algorithm is similar to the multi-path MCB except that every node has only one path to the nearest (with the fewest number of hops in the path) router. The random graphs are generated in the following way: All nodes in the graph are randomly distributed in a 500m $\times$ 500m square region, and if the distance between two nodes is in the radio range, then the two nodes are connected in the graph. There are four gateways that are located in the four corners of the square region. Every node has four routes to the four gateways, and routing is based on the Dijkstra algorithm [52]. The goal of the attacker is to block the traffic of some target nodes. The experimental validation was done by writing a simple C++ based simulator, since the aim of these experiments is to verify the theoretical results by analyzing the performance of the algorithms from a graph theory perspective. No network simulators were needed since no network related properties are considered in our analyses. All tests were run on a Linux OS powered computer, with 1Gb RAM and an Intel Pentium III 1.33 GHz processor. For the source code of the simulation experiment, the reader may refer to Appendix.

We tested two scenarios. In the first scenario, each node has cost 1, which means that the effort needed to compromise different nodes is the same. In the second scenario, every node has a base cost 10, plus an additional cost inversely proportional to the distance between the node and the center of the whole square region; the maximum value of additional cost is 10. This scenario is based on the assumption that it is more difficult to compromise those nodes which are closer to the gateways, and this assumption is quite practical. The total number of nodes in the region is denoted by n, the radio range of a single node is r and the possibility that a node is selected as target node is denoted by u. All selected target nodes are at least one hop away from all gateways (which we denote as non-G1 nodes). Here n, r and u are adjustable parameters. When at least 3 out of the 4 paths to the gateways (here the path does not include the node itself and the gateway) of a node are locked, we consider that the node is blocked. We use our algorithms to find the subset of nodes with minimum total cost in order to block paths from some randomly selected nodes in the square region. The experimental results from the direct implementation of our algorithms are shown in figures 4 through 9. In all these figures, x-axis represents the r values of a single node, y-axis denotes the total cost of the subset of nodes found by the algorithms. All data points are the average of 100 runs. The value of r ranges from 100 to 180.

Figure 4 through Figure 7 are results for the scenario where every node has cost value 1. Figure 4 corresponds to u = 1/20 and n = 120. Figure 5 shows the results of u = 1/10 and n =120. Figure 6 shows the results of u = 1/20 and n = 100. Figure 7 shows the results of u = 1/10 and n = 100. The graphs in Figure 8 and Figure 9 show the results when the second scenario to generate cost value of nodes is used. Figure 8 shows the results for u = 1/20 and n = 120, while Figure 9 shows the results for u = 1/20 and n = 100.

The following conclusions can be drawn from these experimental results.

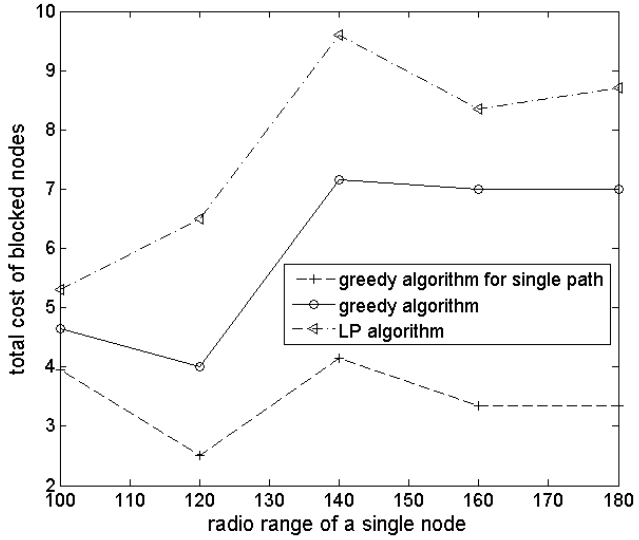

Fig. 4. Cost of Blocking: Scenario 1, u = 1/20 and n = 120

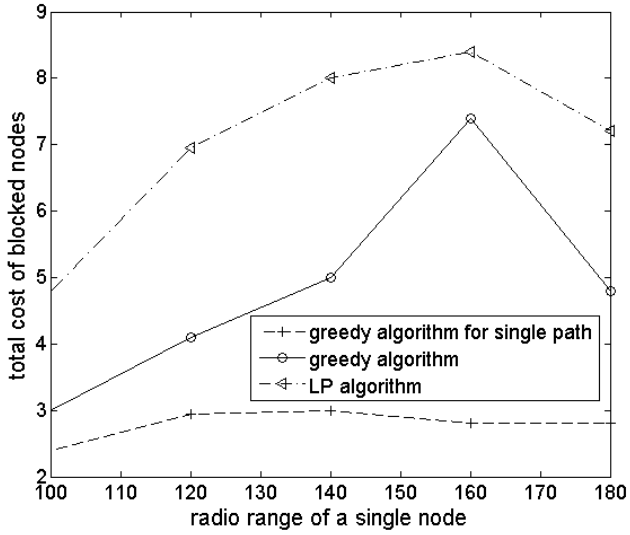

Fig. 6. Cost of Blocking: Scenario 1, u = 1/20 and n = 100

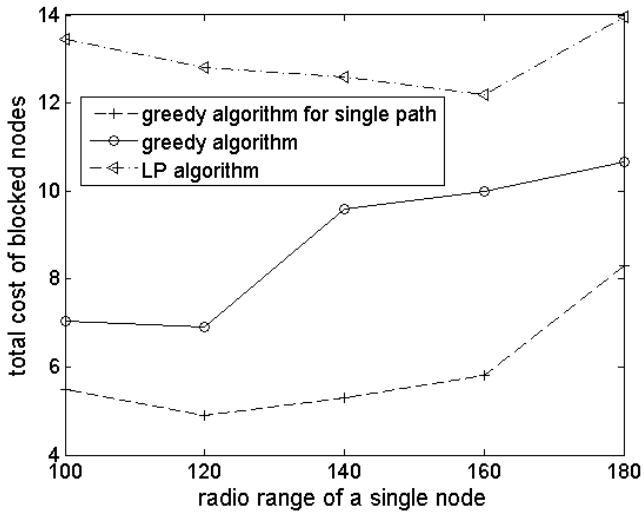

Fig. 5. Cost of Blocking: Scenario 1, u = 1/10 and n =120

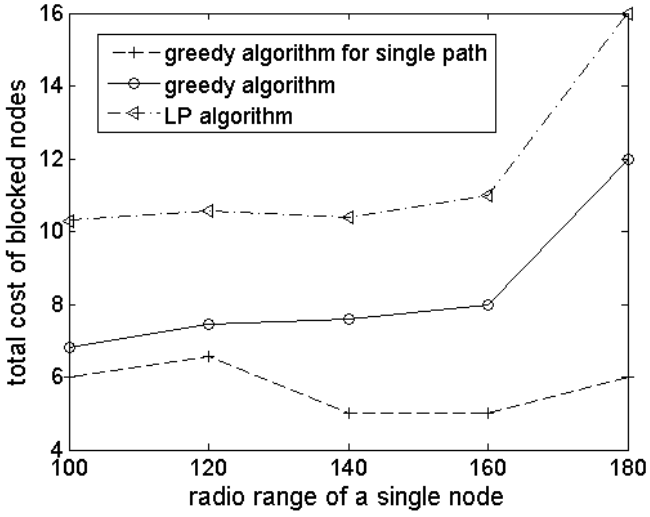

Fig. 7. Cost of Blocking: Scenario 1, u = 1/10 and n = 100

- The performance of Greedy Algorithm is better than that of LP Algorithm in most of the cases we tested. Intuitively, this is because the first algorithm is more like a "global" algorithm, and the second algorithm considers every node separately.
- In all the test cases, the cost of the single-path-blocking greedy algorithm is lower than the two multi-path algorithms. This is obvious and reasonable since it requires more effort to block more paths. But when the number of target nodes increases, the difference between the cost of single path blocking and multi-path blocking decreases. This is because in this case there will be more paths in the graph, and some nodes may become a bottleneck for several paths. These nodes would be easy targets for attacks.
- When the number of nodes increases, the cost for both single path blocking and multi-path blocking increases. This is also reasonable since in this situation, the graph become denser, and the targeted paths will become more disjointed.
- When the radio range of a single node increases, the trend of blocking-cost for target paths is not very obvious. In some cases, increasing the radio range results in a "peak" for the blocking cost. Intuitively, increase in radio range also increases the number of edges in the graph, making the target paths more disjoint. But when the number of edges reaches a threshold, it ceases to have a significant effect in the disjointedness of the paths.

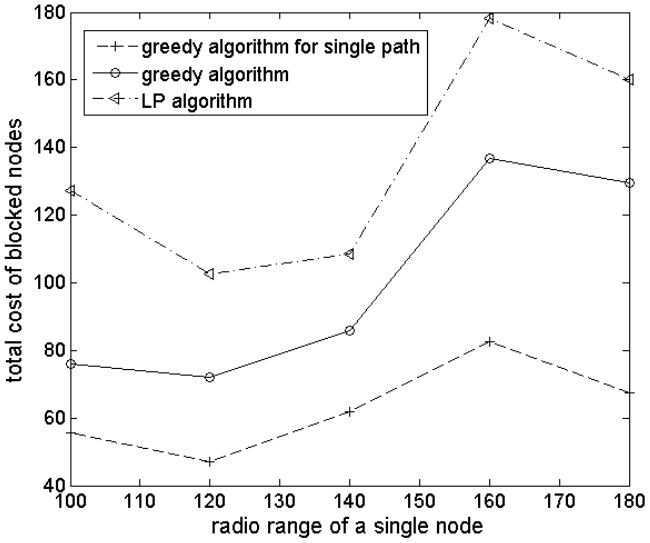

Fig. 8. Cost of Blocking: Scenario 2, u = 1/20 and n = 120

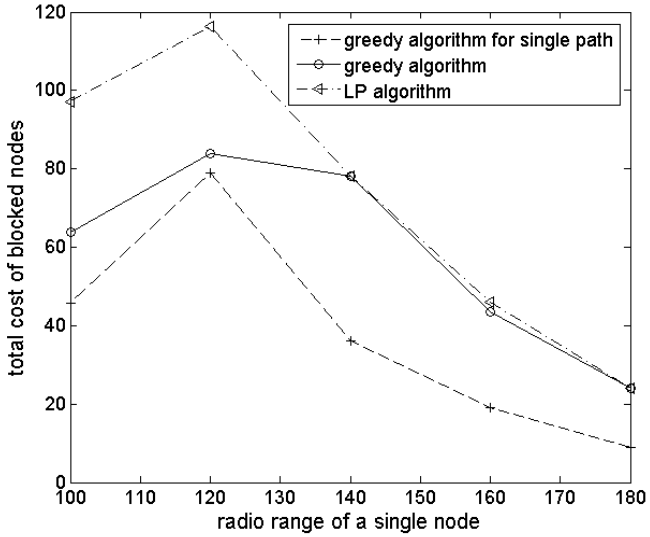

Fig. 9. Cost of Blocking: Scenario 2, u = 1/20 and n = 100

8 Extension to Mesh Networks with Patterned Mobility

In the previous sections, we considered only limited or no network node mobility. If network nodes are mobile then the analysis of the MCB problem becomes more complicated. We first briefly review graph theoretic modeling of node mobility and the concept of stochastic blocking to address the MCB problem for networks with mobile nodes.

Nodes in real wireless networks have some form of patterned mobility (demonstrated in [44] and the references therein). In WMNs, the mobility pattern of the nodes is predictable [44], [1]. The nodes move within mobility orbits and the position of a given node has a probability distribution over the positions of the orbit. Our analysis of the node-mobility blocking problem assumes that movement of the WMN nodes follows such a probabilistic patterned mobility model. We introduce the concept of Node-based Stochastic Graphs to characterize such patterned node mobility.

Definition 8.1: Node-based Stochastic Graph: It is an undirected graph with a subset of nodes that are dynamic, i.e., every such node is associated with a probability of existence. Formally, let $G = (V, E)$ be an undirected graph with n nodes, where $V = \{v_{11}, \ldots, v_{1t_1}, v_{21}, \ldots, v_{2t_2}, v_{h1}, \ldots, v_{ht_h}, v_{h+1}, \ldots, v_n\}$. V contains two types of nodes: fixed nodes and dynamic nodes (our definition of Node-based Stochastic Graphs is in line with the WMN architecture. WMN routers have no/low mobility and WMN nodes are mobile).

Nodes $v_{h+1}, \ldots, v_n$ are fixed nodes. Nodes $v_{i1}, \ldots, v_{it_i}$, $1 \leq i \leq h$ are possible positions of node v_i, $1 \leq i \leq h$. There is an associated probability p_{ij} for every v_{ij}, $1 \leq i \leq h$, $1 \leq j \leq t_i$, which means that node v_i has probability p_{ij} to be in position v_{ij} of G.

8.1 The Stochastic Blocking Problem

Since the network is dynamic, the adversarial goal would be to choose such a set of target nodes, whereby blocking them would result in the blocking probability being higher than some desired value. The formal description of the Stochastic Blocking is as follows.

Given: (1) A Stochastic Graph $S(V, E)$, where every node v_i in V has a cost c_i of compromise.

(2) The set of nodes in $m = \sum_{i=1}^{k} n_i$ paths

$$(P_{11}, P_{12}, \ldots, P_{1n_1}), (P_{21}, P_{22}, \ldots, P_{2n_2}),$$

$$\ldots, (P_{k1}, P_{k2}, \ldots, P_{kn_k}).$$

All source and destination nodes in these paths are fixed nodes.

(3) Integers $R_i, 0 \leq R_i \leq n_i$, and a value p, $0 \leq p \leq 1$.

Statement 1 (Optimization): Is there a subset V' of V such that compromising V' will block, with a probability p, at least R_i paths out of $P_{i1}, P_{i2}, \ldots, P_{in_i}$, $(1 \leq i \leq k)$, for every node v_i, $i = 1, 2, \ldots, k$, and the total cost of nodes in V' is minimized? This is an optimization problem; the corresponding decision problem (with same conditions) is stated below.

Statement 2 (Decision): Is there a subset V' of V such that, compromising V' will block, with a probability p, at least R_i paths out of $P_{i1}, P_{i2}, \ldots, P_{in_i}$, $(1 \leq i \leq k)$, for every node v_i, $i = 1, 2, \ldots, k$, and the total cost of nodes in V' is no greater than C? Here C is some pre-specified number.

Next we demonstrate that even determining the blocking probability of a given dynamic graph is $\#P$-hard. For that we first define a problem called $\#Blocking$, evaluate the hardness of $\#Blocking$, and show that Stochastic Blocking is harder than $\#Blocking$.

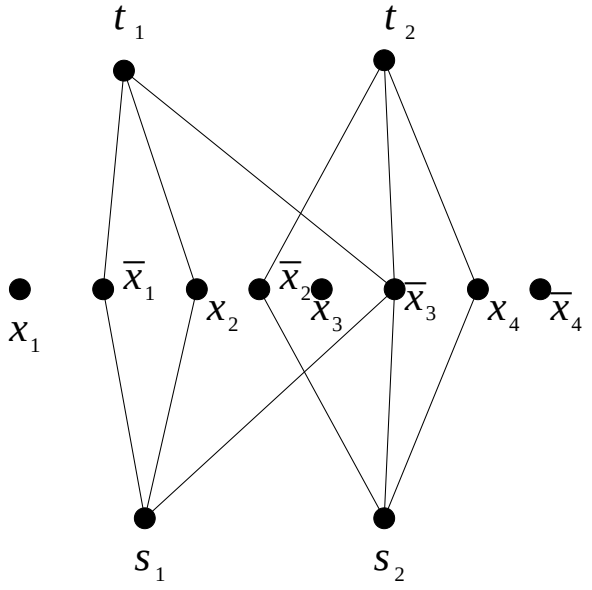

Fig. 10. An instance of $\#SAT$

8.2 $\#Blocking$: Evaluating Hardness of Stochastic Blocking

Definition 8.2: $\#Blocking$: Given the above graph model, the computational problem $\#Blocking$ is to determine the probability that at least R_i paths out of $P_{i1}, P_{i2}, \ldots, P_{in_i}$, $(1 \leq i \leq k)$ will be blocked.

It is evident that an efficient solution of $\#Blocking$ is a necessary precursor to solving Stochastic Blocking efficiently; it is required to determine the blocking probability before finding the optimal subset of nodes for blocking. That is, stochastic MCB should be at least as hard as $\#Blocking$. Next we show that $\#Blocking$ is $\#P - hard$ using the reduction of $\#SAT$ to $\#Blocking$.

Theorem 8.3: $\#Blocking$ is $\#P - hard$.

Proof: We can reduce $\#SAT$ to $\#Blocking$. Suppose that we have a 3SAT instance. We can create a stochastic graph as follows. Given the 3SAT instance, we create a dynamic node for every variable, which has two possible positions in the stochastic graph, and every position has probability 1/2. We also create a source and destination node for every clause, where the source and destination nodes are connected with 3 paths through the 3 dynamic nodes corresponding to the three variables in the clause. Now we require that at least one of these 3 paths is blocked for every such source destination pair. Then it is easy to see that the blocking probability is exactly the probability that the 3SAT instance is satisfied, which means $\#Blocking$ is $\#P$-hard. Figure 10 shows the $\#Blocking$ instance we constructed through this procedure for the $\#SAT$ following the $\#SAT$ instance. □

This result demonstrates that even determining the blocking probability is very hard in the patterned mobility model (the actual position of $\#P$ in the complexity hierarchy is unknown, but it is generally assumed to be harder than NP). So, the task of blocking would be even harder for the adversary. Additionally, the adversary may not know the actual mobility patterns and the possible orbits of the network nodes, further enhancing the degree of hardness. Thus, it would be extremely hard for the adversary to efficiently launch such blocking-type attacks against multi-path protocols with node mobility. The degree of hardness prevents the design of approximation algorithms for efficient blocking in the node mobility scenario and this is an open research problem. Our ongoing research focuses on further investigation of the blocking attacks for various mobility models and efficiency evaluation through simulation.

9 Conclusions

This paper demonstrates the superiority of multi-path protocols over traditional single-path protocols in terms of resiliency against blocking and node isolation-type attacks, especially in the wireless networks domain. Multi-path protocols for WMNs make it extremely hard for an adversary to efficiently launch such attacks. This paper is an attempt to model the theoretical hardness of attacks on multi-path routing protocols for mobile nodes and quantify it in mathematical terms. At this point, it is also worthwhile to mention about the impact of this study. We believe that the results of our research will impact a number of areas including the security and robustness of routing protocols in mesh networks, threshold cryptography and network coding. Moreover, even though we do not necessarily consider insider attacks, we would like to point out that our analysis does allow for an attacker to possess topological information of the network, which is the case of an insider attack. Even in this case, our analysis shows that staging a blocking attack is hard for the attacker, in a network of reasonable size.

As a part of our ongoing research, we plan to further investigate the approximation algorithms for the MCB problem. We also plan to investigate the problem in the settings related to ID-based key update protocols, which is very promising in wireless networks. In our discussions we assumed that the adversary has topological information of the network. It would be an interesting problem to study the additional difficulty associated with blocking when the topological information is effectively hidden from the adversary. Further, we would also like to evaluate our algorithms by running them on a real wireless mesh network and validate the results obtained by the C++ based experiments on random graphs. This paper also brings forth some interesting related problems. For example, if link-cut and node-compromising are combined together (i.e., one can either cut some links or compromise some nodes), then what is the minimum total cost to block traffic from specific nodes.

References

[1] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey," *Computer Networks Journal*, vol. 47, pp. 445–487, 2005.

[2] "http://wire.less.dk/wiki/index.php/meshlinks."
[3] "http://www.communitywireless.org/."
[4] "http://www.open-mesh.com/."
[5] "http://pdos.csail.mit.edu/roofnet/design/."
[6] C.-K. Chau, R. Gibbens, R. Hancock, and D. Towsley, "Robust multipath routing in large wireless networks," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 271 –275.
[7] Y. Kato and F. Ono, "Node centrality on disjoint multipath routing," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, May 2011, pp. 1 –5.
[8] M. Razzaque and C. Hong, "Analysis of energy-tax for multipath routing in wireless sensor networks," *Annals of Telecommunications*, vol. 65, pp. 117–127, 2010.
[9] J. So and N. H. Vaidya, "Load balancing routing in multi-channel hybrid wireless networks with single network interface," in *Second International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'05)*, Washington, DC, USA, August 2005.
[10] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, Jan. 2006.
[11] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, pp. 1204–1216, 2000.
[12] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, 2003.
[13] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *Public Key Cryptography 2001*, 2001, pp. 119–136.
[14] ——, "A length-flexible threshold cryptosystem with applications," in *Proceedings of the 8th Australasian conference on Information security and privacy*, ser. ACISP'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 350–364.
[15] L. Ertaul and W. Lu, "ECC based threshold cryptography for secure data forwarding and secure key exchange in MANET (i)," 2005, pp. 102–113.
[16] L. Ertaul and N. Chavan, "Security of ad hoc networks and threshold cryptography," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 1, June 2005, pp. 69 – 74.
[17] M. Wu, S. Chen, and J. Liao, "Data security in MANETs by integrating multipath routing and secret sharing," in *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, vol. 1, March 2010, pp. 72 –75.
[18] S. Yu, H. Li, H. Zhu, and C. Wang, "Improving network security and performance by multipath routing in ad hoc networks," in *Computational Aspects of Social Networks (CASoN), 2010 International Conference on*, September 2010, pp. 78 –81.
[19] S. Chen and M. Wu, "Game theoretic approach in multipath routing for tradeoff between routing security and performance," in *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, April 2010, pp. 717 –722.
[20] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. of Oper. Res.*, vol. 4, pp. 233–235, 1979.
[21] S. A. Cook, "The complexity of theorem proving procedures," in *Third Annual ACM Symposium on the Theory of Computing*, New York, 1971, pp. 151–158.
[22] C. Papadimitriou, *Computational Complexity*. Addison Wesley, 1993.
[23] S. Mueller, R. Tsang, and D. Ghosal, "Multipath routing in mobile ad hoc networks: Issues and challenges," in *In Performance Tools and Applications to Networked Systems, volume 2965 of LNCS*. Springer-Verlag, 2004, pp. 209–234.
[24] M. A. Moustafa, M. A. Youssef, and M. N. El-Derini, "MSR: A multipath secure reliable routing protocol for WSNs," in *Computer Systems and Applications (AICCSA), 2011 9th IEEE/ACS International Conference on*, December 2011, pp. 54 –59.
[25] Y. Zhang, J. Yang, H. Vu, and Y. Wu, "The design and evaluation of interleaved authentication for filtering false reports in multipath routing WSNs," *Wireless Networks*, vol. 16, pp. 125–140, 2010.
[26] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 11–25, October 2001.
[27] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.
[28] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *In ACM MobiCom*. ACM Press, 2004, pp. 114–128.
[29] K. Yang, Y. Wu, and H.-H. Chen, "Qos-aware routing in emerging heterogeneous wireless networks[quality-of-service-based routing algorithms for heterogeneous networks]," *Communications Magazine, IEEE*, vol. 45, no. 2, pp. 74 –80, February 2007.
[30] R. Coltun, D. Ferguson, and J. Moy, "Ospf for ipv6," Tech. Rep., 1999.
[31] S. Murphy, M. Badger, and B. Wellington, "OSPF with Digital Signatures," RFC 2154 (Experimental), Internet Engineering Task Force, Jun. 1997.
[32] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (Informational), Internet Engineering Task Force, Apr. 1992.
[33] L. Zhou and Z. J. Haas, "Securing ad hoc networks," *IEEE NETWORK MAGAZINE*, vol. 13, pp. 24–30, 1999.
[34] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 259–271.
[35] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security protocols for sensor networks," vol. 8, no. 5. Hingham, MA, USA: Kluwer Academic Publishers, Sep. 2002, pp. 521–534.
[36] S. Chen and M. Wu, "Anonymous multipath routing protocol based on secret sharing in mobile ad hoc networks," *Systems Engineering and Electronics, Journal of*, vol. 22, no. 3, pp. 519 –527, June 2011.
[37] S. M. Bellovin and E. R. Gansner, "Using link cuts to attack internet routing," in *Tech. Rep., ATT Research, 2004, Work in Progress 2003 USENIX*, 2003.
[38] "IEEE Standard for Information Technology: Wireless LAN Medium Access Control (MAC) and physical layer (PHY) Specifications," *IEEE Std 802.11-2007 - Revision of IEEE Std 802.11-1999*, 2007.
[39] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1973, pp. 38–49.
[40] L. Lovasz, "On the ratio of optimal integral and fractional covers," *Discrete Math.*, vol. 13, pp. 383–390, 1975.
[41] M. J. Kearns, *Computational complexity of machine learning*, ser. ACM distinguished dissertations. MIT Press, 1990.
[42] E. Petrank, "The hardness of approximation: Gap location," in *Computational Complexity*, vol. 4, 1994, pp. 133–157.
[43] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189–201, 1979.
[44] J. Ghosh, S. J. Philip, and C. Qiao, "Sociological orbit aware location approximation and routing (solar) in MANET," *Ad Hoc Netw.*, vol. 5, no. 2, pp. 189–209, Mar. 2007.
[45] H. Deng, W. Li, and D. Agrawal, "Routing security in wireless ad hoc networks," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 70 – 75, October 2002.
[46] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, March-3 April 2003, pp. 1976 – 1986.
[47] L. Qian, N. Song, and X. Li, "Detection of wormhole attacks in multi-path routed wireless ad hoc networks: A statistical analysis approach," *J. Network and Computer Applications*, vol. 30, no. 1, pp. 308–330, 2007.
[48] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Springer, 2002, pp. 251–260.
[49] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, May 2003, pp. 113 – 127.
[50] V. V. Vazirani, *Approximation algorithms*. Springer, 2004.
[51] R. Gandhi, S. Khuller, and A. Srinivasan, "Approximation algorithms for partial covering problems," *Journal of Algorithms*, vol. 53, pp. 55–84, 2004.
[52] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik. 1*, pp. 269–271, 1959.

Appendix

Source Code of the Simulation Experiment

1) Header Files –

a. Graph.h

```
/*
The code can be used by researchers with due acknowledgment to the authors
Qi Duan, Mohit Virendra, Shambhu Upadhyaya and Ameya Sanzgiri,
Univeristy at Buffalo, NY
*/
// graph.h: interface for the graph class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_GRAPH_H__2418F5C9_2FD6_430F_A8D3_FFD17F7CCD2F__INCLUDED_)
#define AFX_GRAPH_H__2418F5C9_2FD6_430F_A8D3_FFD17F7CCD2F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <iostream>
#include "time.h"
#include "queue.h"

class graph
{
public:
       graph(int size = 2, int fraction = 1);
       virtual ~graph();
       bool isConnected(int x, int y);
    void addEdge(int x, int y, int weight=1);

    // performs a Breadth First Search starting with node x
       int BFS(int x);

       // searches for the minimum length path
    // between the start and target vertices

    void minPath(int start, int target);
    void graph::Dijkstra(int source);
    int graph::randomPosition(int radioRange);
    int** graph::getMinDist();
       double* graph::getReturnValue()     ;
       int graph::greedy();
       int graph::update(int node);
       int graph::checkGain(int node);
       int graph::greedy_singlePath();
       int graph::update_singlePath(int node);
       int graph::checkGain_singlePath(int node);
       int graph::selectNode(int node);
       int graph::backupSelectNode();
       int graph::resumeSelectNode();
       int graph::genIncident(int fraction);
       int graph::perNode();
       int graph::checkAll();
       int graph::perNodeGreedy();
       int graph::reset();
```

```
private :
	double returnValue[2];
	int n;
	int **A;
	int **minPathDist;
	int **position;
	int *weight;
	int *selected;
	int *shortest;
	int *shortestLen;
	int *isGoal;
	int *isCovered;
	int *isCoveredPath;
	int *selected_bk;
	int **inci;
};

#endif // !defined(AFX_GRAPH_H__2418F5C9_2FD6_430F_A8D3_FFD17F7CCD2F__INCLUDED_)
```

b. Queue.h

```
/*
The code can be used by researchers with due acknowledgment to the authors
Qi Duan, Mohit Virendra, Shambhu Upadhyaya and Ameya Sanzgiri,
Univeristy at Buffalo, NY
*/
// queue.h: interface for the queue class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_QUEUE_H__08368E77_6622_4B92_8091_B2A41A59BFE3__INCLUDED_)
#define AFX_QUEUE_H__08368E77_6622_4B92_8091_B2A41A59BFE3__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <cstdlib>
struct node {
    int info;
    node *next;
};

class queue
{
public:
	queue();
	virtual ~queue();
    bool isEmpty();
    void add(int);
    int get();
private:
    node *first, *last;


};
```

```
#endif // !defined(AFX_QUEUE_H__08368E77_6622_4B92_8091_B2A41A59BFE3__INCLUDED_)
```

2) Cpp Files-

a. Graph.cpp

```
/*
The code can be used by researchers with due acknowledgment to the authors
Qi Duan, Mohit Virendra, Shambhu Upadhyaya and Ameya Sanzgiri,
Univeristy at Buffalo, NY
*/
// graph.cpp: implementation of the graph class.

//

//////////////////////////////////////////////////////////////////////

#include "math.h"
#include "graph.h"

#define EMPTY -10000
#define INFI 100000
#define WIDTH 500
#define BRIDGENUM 4
#define BANDWIDTH 200
#define COST 1
#define RANGE 80
#define THRESH 3
#define COEFF 3
#define NOTFOUND -1
#define MAX_DIST 3000

//////////////////////////////////////////////////////////////////////

// Construction/Destruction

//////////////////////////////////////////////////////////////////////



graph::graph(int size, int fraction)
{
       int i, j;

    if (size < 2) n = 2;
    else n = size;
    A = new int*[n];

    for (i = 0; i < n; ++i)
       A[i] = new int[n];

    for (i = 0; i < n; ++i)
           for (j = 0; j < n; ++j)
               A[i][j] = 0;

    for(i=0;i<n;i++)
              A[i][i] = 0;

    minPathDist = new int*[n];
```

```
    for (i = 0; i < n; ++i)
       minPathDist[i] = new int[2*n];

    for (i = 0; i < n ; i++)
        for (j= 0;j<n;j++)
            minPathDist[i][2*j] = minPathDist[i][2*j+1] = -1;

       //initialize random generator
    srand ( time(NULL) );

    position = new int*[n];
    for (i = 0; i < n; ++i)
       position[i] = new int[2];
    weight = new int[n];
    for (i = 0; i < n; ++i)
        weight[i] = 1 ; // + ( rand() % 5) ;
       isGoal = new int[n];
       for (i = 0; i < n; ++i)
              isGoal[i] = 0;
       selected = new int[n];
    for (i = 0; i < n; ++i)
        selected[i] = 0;
       selected_bk = new int[n];
    for (i = 0; i < n; ++i)
        selected_bk[i] = 0;

       int pathNum = BRIDGENUM*(n-BRIDGENUM);
       isCoveredPath = new int[pathNum];
    for (i = 0; i < pathNum; ++i)
        isCoveredPath[i] = 0;

       isCovered = new int[n-BRIDGENUM];
    for (i = 0; i < n-BRIDGENUM; ++i)
        isCovered[i] = 0;

       shortest = new int[n-BRIDGENUM];
    for (i = 0; i < n-BRIDGENUM; ++i)
        shortest[i] = 0;

       shortestLen = new int[n-BRIDGENUM];
    for (i = 0; i < n-BRIDGENUM; ++i)
        shortestLen[i] = 0;

       inci = new int*[pathNum];

    for (i = 0; i < pathNum; ++i)
       inci[i] = new int[n-BRIDGENUM];

    for (i = 0; i < pathNum ; i++)
        for (j= 0;j<n-BRIDGENUM;j++)
            inci[i][j] = 0;
}

graph::~graph()
{
    int i;

       for ( i = 0; i < n; ++i)
              delete [] A[i];
       delete [] A;
    for ( i = 0; i < n; ++i)
```

```
                delete [] minPathDist[i];
        delete [] minPathDist;
        for ( i = 0; i < BRIDGENUM*(n-BRIDGENUM); ++i)
                delete [] inci[i];
        delete [] inci;
    for ( i = 0; i < n; ++i)
                delete [] position[i];
        delete [] position;
    delete [] weight;
        delete [] selected;
        delete [] selected_bk;
    delete [] isCovered;
        delete [] isCoveredPath;
}



bool graph::isConnected(int x, int y)
{
        return (A[x][y]  > 0);
}

void graph::addEdge(int x, int y, int weight)
{
         A[x][y] = A[y][x] = weight;
}

int graph::BFS(int x)
{
    queue Q;
    bool *visited = new bool[n];
    int i;

    for (i = 0; i < n; ++i)
        visited[i] = false;
    Q.add(x);
    visited[x] = true;
    //printf("Breadth First Search starting from vertex %d : \n",x);
    while (!Q.isEmpty()) {
        int k = Q.get();
        for (i = 0; i < n; ++i)
            if (isConnected(k, i) && !visited[i]) {
                Q.add(i);
                visited[i] = true;
            }
    }
    //printf("\n");
    for (i = 0; i < n; i++ )
        if ( !visited[i] ) {
           return 0;
             }
    //delete [] visited;
    return 1;
}



void graph::minPath(int start,int target)
{
    queue Q;
    int i, p, q;
```

```
    bool found;
    struct aux { int current, prev; };
    aux *X = new aux[n+1];
    int *Y = new int[n+1];
    bool *visited = new bool[n+1];

    for (i = 1; i <= n; ++i)
           visited[i] = false;
    Q.add(start);
    visited[start] = true;
    found = false;
    p = q = 0;
    X[0].current = start;
    X[0].prev = 0;
    while (!Q.isEmpty() && !found) {
        int k = Q.get();
        for (i = 1; i <= n && !found; ++i)
                    if (isConnected(k, i) && !visited[i]) {
                           Q.add(i);
                           ++q;
                           X[q].current = i;
                           X[q].prev = p;
                           visited[i] = true;
                           if (i == target) found = true;
                    }
        ++p;
    }
    printf("The minimum length path from %d to %d is:\n",  start, target);
    p = 0;
    while (q) {
        Y[p] = X[q].current;
        q = X[q].prev;
        ++p;
    }
    Y[p] = X[0].current;
    for (q = 0; q <= p/2; ++q) {
        int temp = Y[q];
        Y[q] = Y[p-q];
        Y[p-q] = temp;
    }
    for (q = 0; q <= p; ++q)
              printf( "%d \n",Y[q] );
    printf( "Length = %d\n" , q-1 );
    delete [] visited;
    delete [] X;
    delete [] Y;
}

void graph::Dijkstra(int source)
{
    int i,j,minDistNode;
    int nodeCount = 0;
    int *dist,*prev,*isAdded;
       int minDist;
    dist = new int[n];
       prev = new int[n];
       isAdded = new int[n];
    minPathDist[source][2*source] = minPathDist[source][2*source+1]= -1;
    for (i = 0; i < n; i++) {
        if ( i == source )
            dist[i] = 0;
```

```
            else
                dist[i] = INFI;
            prev[i] = EMPTY;
        isAdded[i] = 0;
    }
    while (nodeCount < n) {
        minDistNode = 0;
        minDist = INFI;
        for (i = 0; i < n; i++) {
               if ( !isAdded[i] && dist[i] < minDist  ) {
                   minDist = dist[i];
                minDistNode = i;
                     }
        }
        if (minDist == INFI) {
            //printf("no path any more! node count is %d \n",nodeCount);
            break;
        }
        isAdded[ minDistNode ] = 1;
        nodeCount ++;
        minPathDist[source][minDistNode*2 ] = dist[minDistNode];
        minPathDist[source][minDistNode*2 + 1 ] = prev[minDistNode];
        if  (minDistNode==source) prev[ minDistNode ] = source;
        for (j = 0; j < n; j++) {
                     if (A[minDistNode][j] > 0 )
                            if(dist[minDistNode] + A[minDistNode][j] < dist[j] ) {
                                   dist[j] = dist[minDistNode] + A[minDistNode][j];
                                   prev[j] = minDistNode;
                            }
        }
    }
    return;
}

int graph::randomPosition(int radioRange)
{
    int i,j,dist;
       int diaDist[4], minDiaDist;

       for (i = 0; i < n; i++) {
              selected[i] = 0;
        for ( j= 0; j < n; j++)
                     A[i][j] = 0;
       }

       //generate random weight
       //for (i = 0; i < n; ++i)
      //  weight[i] = 1 + ( rand() % 10) ;

       // fix four corners
    position[0][0] = 0;
    position[0][1] = 0;
    position[1][0] = 0;
    position[1][1] = WIDTH - 1;
    position[2][0] = WIDTH - 1;
    position[2][1] = 0;
    position[3][0] = WIDTH - 1;
    position[3][1] = WIDTH - 1;

    for (i = BRIDGENUM; i < n ; i++) {
        //printf(" node i = %d", i);
```

```
        int find = 0;
        while (!find) {
                    position[i][0] = (rand() % (WIDTH - 2) ) ;
                    position[i][1] = (rand() % (WIDTH - 2) ) ;
            find = 1;
            for ( j= BRIDGENUM; j < i; j++) {
                            if ( abs ( position[i][0] - position[j][0] )
                                  + abs (position[i][1] - position[j][1] )  < 10  ) {
                                   find = 0;
                                   break;
                            }
                    }
        }
    }

    for (i = BRIDGENUM; i < n; i++) {

                //compute distance
        for ( j= 0; j < n; j++) {
                    if (i!= j) {
                           double L1 = position[i][0] - position[j][0];
                           double L2 = position[i][1] - position[j][1];
                           double L = sqrt(L1*L1 + L2*L2);
                           dist  = (int) L ;
                           if ( dist <= radioRange )  {

                                   A[i][j] =A[j][i] = dist;
                           }
                           else
                                   A[i][j] =A[j][i] =  0;
                    }
                    if (j < BRIDGENUM )
                           diaDist[j] = dist;
        }
              minDiaDist = MAX_DIST;
              for ( j= 0; j < BRIDGENUM; j++)
                    if  ( diaDist[j] < minDiaDist )
                           minDiaDist = diaDist[j];
              weight[i] = 10 + 10 - floor ((float) minDiaDist*20/(float)(sqrt(2)*WIDTH)
);
       }

    /*for (i = 0; i < n; i++) {
        printf("\npos %d is %d %d \n",i,position[i][0],position[i][1]);
        for ( j= 0; j < n; j++)
               if (A[i][j] )
                  printf(" dist[%d][%d] = %d ",i,j,A[i][j]);
    } */
    // check if the graph is connected
    if (BFS(0) )  {
              //printf (" get a connected graph!\n" );
        return 1;
       }
    else {
              //printf (" get an unconnected graph!\n" );
        return 0;
       }
}


/* generate incident matrix */
```

```
int graph::genIncident(int fraction)
{
	int i,j,k,next,current,index_X,index_Y,shortestIndex,shortestLength,length;

	//printf("enter genInci \n" );
	int pathNum = BRIDGENUM*(n-BRIDGENUM);
	for (i = 0; i < pathNum ; i++)
	 for (j= 0;j<n-BRIDGENUM;j++)
	     inci[i][j] = 0;

	for(i = BRIDGENUM; i < n; i++)
	 Dijkstra(i);
	for ( i = BRIDGENUM; i < n; i++ )  {
		//printf("enter loop1 %d \n",i );
		shortestIndex = -1;
		shortestLength = MAX_DIST;
		for ( j = 0; j < BRIDGENUM; j++ ) {
			length = 0;
			next = current = j;
			while (next != EMPTY ) {
				length ++;
				next = minPathDist[i][2*current+1];
				if (next !=EMPTY && next >= BRIDGENUM && next!= i) {
					index_X = BRIDGENUM*(i-BRIDGENUM) + j;
					index_Y = next-BRIDGENUM;
					if (index_X < BRIDGENUM*(n-BRIDGENUM) && index_Y < n-
BRIDGENUM )
						inci[index_X][index_Y] = 1;
					else printf("out of boundry! \n" );
					current = next;
				}
				else if (next == i)
					next = EMPTY;
			}
			if (length < shortestLength ) {
				shortestLength = length;
				shortestIndex = j;
			}
		}
		shortest[i-BRIDGENUM] = shortestIndex ;
		shortestLen[i-BRIDGENUM] = shortestLength ;
		//printf("shortest  for %d is %d length %d \n ",i,
		//      shortest[i-BRIDGENUM],shortestLen[i-BRIDGENUM]);
	}
	for (i = BRIDGENUM; i < n; ++i)  {
		if (shortestLen[i-BRIDGENUM] > 1 ) {
	     isGoal[i] =  ( (rand() % fraction) >0 ? 0:1 ) ;
			//printf (" goal is %d \n", i);
			//break;
		}
		else
			isGoal[i] = 0;

	}

	//printf( "goal count is %d \n",count);
	return 0;
}

/* greedy alogrithm for single path MCB*/
int graph::greedy_singlePath()
```

```
{
	int i,j, maxIndex, count, cost = 0;
    float gain, maxGain;

	reset();
	while (!checkAll() ) {
		//printf(" enter loop \n");
		maxGain = 0;
		maxIndex = -1;
		for ( i = BRIDGENUM; i< n; i++) {
			gain = (float)checkGain_singlePath(i)/(float)weight[i];
			if (gain > maxGain) {
				maxGain = gain;
				maxIndex = i;
			}
		}
		//printf (" max index for single path is %d \n", maxIndex);
		if (maxIndex >= BRIDGENUM )
			update_singlePath (maxIndex);
	}
	for ( i = BRIDGENUM; i< n; i++)
		if (selected[i] )
			cost += weight[i];

	return cost;
}

/* greedy alogrithm for MCB*/
int graph::greedy()
{
	int i,j, maxIndex, count, cost = 0;
    float gain, maxGain;

	reset();
	while (!checkAll() ) {

		maxGain = 0;
		maxIndex = -1;
		for ( i = BRIDGENUM; i< n; i++) {
			gain = (float)checkGain(i)/(float)weight[i];
			if (gain > maxGain) {
				maxGain = gain;
				maxIndex = i;
			}
		}
		if (maxIndex >= BRIDGENUM )
			update (maxIndex);
	}
	for ( i = BRIDGENUM; i< n; i++)
		if (selected[i] )
			cost += weight[i];

	return cost;
}

int graph::perNodeGreedy()
{
	int i,j,k,gain,index,count,maxIndex,cost=0;
	float weightedGain,maxGain;

	reset();
```

```
	for (i = BRIDGENUM; i< n; i++) {
		while (!isCovered[i-BRIDGENUM] && isGoal[i]) {
			maxGain = 0;
			maxIndex = -1;
			for (j = BRIDGENUM; j< n; j++) {
				if (selected[j] == 1)
					gain = 0;
				else {
					gain = 0;
					count = 0;
					for (k = 0; k < BRIDGENUM; k++) {
						index = (i-BRIDGENUM)*BRIDGENUM + k;
						count += isCoveredPath[index] ;
						if (!isCoveredPath[index] && inci[index][j-
BRIDGENUM] )
							gain++;
					}
					if ( count >= THRESH )
						gain = 0;
					else if (gain >= THRESH - count)
						gain = THRESH - count;
				}
			  weightedGain = (float) gain/ (float)weight[j];
				if (weightedGain > maxGain ) {
					maxGain = weightedGain;
					maxIndex = j;
				}
			}
			update(maxIndex);
		}
	}

	for ( i = BRIDGENUM; i< n; i++)
		if (selected[i] )
			cost += weight[i];

	return cost;
}

int graph::checkGain(int node)
{
	int i,j, totalGain=0, gain = 0, count, index;

	if (node < 0 || node > n)  {
		printf ( "node index out of boundry !\n" );
		return -1 ;
	}
	if (selected[node] == 1)
		return 0;
    for (i = BRIDGENUM; i< n; i++) {
		gain = 0;
		if ( !isCovered[i-BRIDGENUM] && isGoal[i]) {

			count = 0;
			for (j= 0; j< BRIDGENUM; j++)  {
				index = (i-BRIDGENUM)*BRIDGENUM + j;
				count += isCoveredPath[index];
				if ( !isCoveredPath[index] && inci[index][node-BRIDGENUM] )
					gain ++;
			}
			if ( count >= THRESH )
```

```
                        gain = 0;
                    else if (gain >= THRESH - count )
                        gain = THRESH - count;
                }
                totalGain = totalGain + gain;
    }

        return totalGain;
}

int graph::checkGain_singlePath(int node)
{
        int i,j, totalGain=0, gain = 0, count, index;

        if (node < 0 || node > n)  {
                printf ( "node index out of boundry !\n" );
                return -1 ;
        }
        if (selected[node] == 1)
                return 0;
        //printf(" enter check gain \n" );
    for (i = BRIDGENUM; i< n; i++) {
                gain = 0;
                if ( !isCovered[i-BRIDGENUM] && isGoal[i]) {
                        index = (i-BRIDGENUM)*BRIDGENUM + shortest[i-BRIDGENUM];
                        if ( inci[index][node-BRIDGENUM] )
                                gain ++;
                }
                totalGain = totalGain + gain;
    }

        return totalGain;
}

int graph::update(int node)
{
        int i,j, index, count ;

        if (node < 0 || node > n) {
                printf ( "node index out of boundry !\n" );
                return -1 ;
        }
        for (i = BRIDGENUM; i< n; i++) {
                count = 0;
                for (j= 0; j< BRIDGENUM; j++) {
                        index = (i-BRIDGENUM)*BRIDGENUM + j;
                        if (inci[index][node-BRIDGENUM] )
                                isCoveredPath[index] = 1;
                        if ( isCoveredPath[index] )
                                count ++ ;
                }
                if ( count >= THRESH )
                        isCovered[i-BRIDGENUM] = 1;
        }
        selected[node] = 1;
        return 0;
}

int graph::update_singlePath(int node)
{
        int i,j, index, count ;
```

```
	if (node < 0 || node > n) {
		printf ( "node index out of boundry !\n" );
		return -1 ;
	}
	for (i = BRIDGENUM; i< n; i++) {
		index = (i-BRIDGENUM)*BRIDGENUM + shortest[i-BRIDGENUM];
		if (inci[index][node-BRIDGENUM] )
			isCovered[i-BRIDGENUM] = 1;
	}
	selected[node] = 1;
	return 0;
}

int graph::checkAll()
{
	int i,j,done;

	done = 1;
	for (i = BRIDGENUM; i< n; i++) {
		if ( !isCovered[i-BRIDGENUM] && isGoal[i]) {
			done = 0;
			break;
		}
	}
	return done;
}


int** graph::getMinDist()
{
    return minPathDist;
}

double* graph::getReturnValue()
{
    return returnValue;
}

int graph::selectNode(int node)
{
	if (node < 0 || node >= n ) {
		printf (" index out of bound!\n" );
		return -1;
	}
	else if (selected[node] == 1) return 0;
	selected[node] = 1;
	return 1;
}

int graph::backupSelectNode()
{
	int i;
	for (i = 0; i <n ; i++)
		selected_bk[i] = selected[i];
	return 0;
}

int graph::resumeSelectNode()
{
	int i;
```

```
	for (i = 0; i <n ; i++)
		selected[i] = selected_bk[i];
	return 0;
}

int graph::reset()
{
	int i;
	for (i = 0; i <n ; i++)
		selected[i] = 0;

	int pathNum = BRIDGENUM*(n-BRIDGENUM);
	for (i = 0; i < pathNum; ++i)
	 isCoveredPath[i] = 0;
	for (i = 0; i < n-BRIDGENUM; ++i)
	 isCovered[i] = 0;
	return 0;
}
```

b. Queue.cpp

```
/*
The code can be used by researchers with due acknowledgment to the authors
Qi Duan, Mohit Virendra, Shambhu Upadhyaya and Ameya Sanzgiri,
Univeristy at Buffalo, NY
*/
// queue.cpp: implementation of the queue class.
//
//////////////////////////////////////////////////////////////////////

#include "queue.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

queue::queue()
{
	first = new node;
    first->next = NULL;
    last = first;

}

queue::~queue()
{
    delete first;
}

bool queue::isEmpty() {
    return (first->next == NULL);
}

void queue::add(int x) {
    node *aux = new node;
    aux->info = x;
    aux->next = NULL;
    last->next = aux;
```

```
    last = aux;
}

int queue::get() {
    node *aux = first->next;
    int value = aux->info;
    first->next = aux->next;
    if (last == aux) last = first;
    delete aux;
    return value;
}
```

3) Main Test File

```
/*
The code can be used by researchers with due acknowledgment to the authors
Qi Duan, Mohit Virendra, Shambhu Upadhyaya and Ameya Sanzgiri,
Univeristy at Buffalo, NY
*/
#include "graph.h"
#include <iostream>


void Traversal()
{
    graph g(5);
    g.addEdge(1, 2); g.addEdge(1, 3); g.addEdge(2, 4);
    g.addEdge(3, 5); g.addEdge(4, 5);
    g.BFS(3);
}

void Maze()
{
    graph f(15);
    f.addEdge(1, 2); f.addEdge(1, 3); f.addEdge(2, 4);
    f.addEdge(3, 14); f.addEdge(4, 5); f.addEdge(4, 6);
    f.addEdge(5, 7); f.addEdge(6, 13); f.addEdge(7, 8);
    f.addEdge(7, 9); f.addEdge(8, 11); f.addEdge(9, 10);
    f.addEdge(10, 12); f.addEdge(10, 15); f.addEdge(11, 12);
    f.addEdge(13, 14); f.addEdge(14, 15);
    f.minPath(1, 10);
}

void mPath()
{
    graph f(6);
    f.addEdge(0,1,2);
    f.addEdge(0,2,1);
    f.addEdge(2,3,1);
    f.addEdge(1,4,3); f.addEdge(1,5,5);
    if ( f.BFS(0) )
        printf("the graph is connected!\n");
    else
        printf("the graph is not connected!\n");
    f.Dijkstra(0);
```

```
    for (int i = 0 ; i< 12 ; i++ ) {
        int **mDist = f.getMinDist();
        printf(" %d ", mDist[0][i] );
    }
    printf ("\n");
}

void conTest1()
{
       int i,j;

    graph f(6);

    f.addEdge(0,1,2);f.addEdge(1,2,1);f.addEdge(1,3,1);f.addEdge(2,4,1);

    f.addEdge(3,4,1);f.addEdge(4,5,2); f.selectNode(0);f.selectNode(5);
       f.selectNode(2);f.selectNode(3);

}

void conTest2()
{
       int i,j;

    graph f(6);

    f.addEdge(0,2,1);f.addEdge(0,3,1);f.addEdge(0,4,1);f.addEdge(4,5,1);
       f.addEdge(1,2,1);f.addEdge(1,3,1);f.addEdge(1,4,1);f.addEdge(1,5,1);
    f.addEdge(0,1,2);
       f.selectNode(0);
       f.selectNode(2);f.selectNode(3);f.selectNode(4);f.selectNode(5);

}

void rPos()
{

}

int main(void) {
    int cost,cost1,cost2,i,j;

       float aveCost,aveCost1, aveCost2;
    //conTest2();
    //rPos();
    int round = 20;

       //graph f(120, 15);
       //while ( !f.randomPosition(180) );
       for (i = 100; i < 200; i+= 20) {

              //printf(" \n " );
              cost1 = 0;
              cost2 = 0;
              cost = 0;
              for (j = 0; j < round; j++ ) {
                     graph f(100, 20);
             while ( !f.randomPosition(i) );

             f.genIncident(20);
```

```
                    cost1 += f.greedy();
                cost2 += f.perNodeGreedy();
                    cost += f.greedy_singlePath();

                }
                aveCost = (float)cost / (float) round;
                aveCost1 = (float)cost1 / (float) round;
                aveCost2 = (float)cost2 / (float) round;
            printf(" %f %f %f \n",aveCost,aveCost1,aveCost2);
                //printf(" %f %f \n",aveCost1,aveCost2);
        }
    return 0;

}
```