BeRGeR: Byzantine-Robust Geometric Routing

Brown Zaz, Mikhail Nesterenko, and Gokarna Sharma

Department of Computer Science, Kent State University, Kent, OH 44242, USA
{zbrown, mikhail, sharma}@cs.kent.edu

Abstract. We present BeRGeR: the first asynchronous geometric routing algorithm that guarantees delivery of a message despite a Byzantine fault without relying on cryptographic primitives or randomization. The communication graph is a planar embedding that remains three-connected if all edges intersecting the source-target line segment are removed. We prove the algorithm correct and estimate its message complexity.

1 Introduction

Geometric routing, also called geographic routing, routing uses node locations to transmit a message from the source to the target node. Nodes may either obtain their coordinates from GPS devices or compute virtual coordinates [1,2,3,4]. Geometric routing has several attractive features. Nodes do not need to store network topology information beyond their immediate neighbors. Moreover, such routing may be stateless as nodes do not have to retain information after forwarding a packet. Compared to flooding-based ad hoc routing algorithms [5,6], geometric routing is more resource-efficient.

Geometric routing may therefore be used in cases where maintaining more extensive routing information is not practicable. It may operate in environments with high topological volatility such as vehicular networks [7] or large collections of resource-poor devices such as wireless sensor networks [8].

Despite the claim of hostile environment applicability, little research has been done on fortifying geometric routing against faults using geometric routing techniques themselves. In this paper, we address this issue.

Geometric routing. The elementary form of geometric routing is greedy. It can be used on either a planar or a non-planar graph embedding.

In greedy routing, the packet is forwarded to the neighbor with the closest Euclidean distance to the target. However, greedy routing fails if it reaches a local minimum. A local minimum is a node that does not have an immediate neighbor closer to the target. To recover from a local minimum and guarantee message delivery, packets are sent to traverse faces of a planarized subgraph [9]. Finding a maximum planar subgraph of a general graph is NP-hard [10]. However, for certain graphs, the task may be solved efficiently. A graph is unit-disk if a pair of its nodes u and v are neighbors if and only if the Euclidean distance between them is no more than 1. Such a graph approximates a wireless network. In a unit-disk graph, a connected planar subgraph may be found by local computation at every node using Relative Neighborhood or Gabriel Graph [11,12,13,14].

A sequential geometric routing algorithm, such as the classic GFG/GPSR [11,13], routes a single packet in the greedy mode until a local minimum is encountered. The algorithm then switches to recovery mode, which involves traversing the faces of a planar subgraph of the original communication graph. Specifically, the algorithm traverses the faces that intersect the line segment that connects the source and the target.

Algorithms such as GFG/GPSR have a problem of face traversal direction choice. A face may potentially be traversed in two directions: clockwise and counter-clockwise. Face traversal may be inefficient if its traversal direction is selected inappropriately: the traversal distance may be long in one direction and short in the other. GOAFR+ [15] finds the shorter traversal direction by reversing it once the packet reaches a pre-determined ellipse containing source and target nodes.

A concurrent geometric routing algorithm CFR [16] sends multiple packets to traverse the faces intersecting the source-target line in both directions in parallel. This naturally selects the shortest face traversal direction.

Byzantine fault tolerance. A Byzantine node [17,18] may behave arbitrarily. This is the strongest fault that can affect a node in a distributed system. A reputation-based approach [19,20] is considered to deal with Byzantine faults. In such an approach, a node deviating from the algorithm may be marked as faulty and avoided by its neighbors. However, a faulty node may actively resist reputation compromise, for example by accusing other nodes of faults or waiting until its malicious influence causes maximum damage. Hence, in general, such approaches may only alleviate rather than eliminate the problem.

The power of the faults may also be mitigated with cryptography [17,21,22] or randomization [23,24]. Synchrony assumptions may help with fault handling as well [25]: if packet transmission may be delayed only for a finite amount of time, then fault information may be obtained from lack of packet receipt. In a completely asynchronous system, such information is not available.

Cryptography may be too expensive for resource poor nodes, the source of true randomness may not be achievable and synchrony may be impossible to guarantee. If none of these primitives are available, the solution requires that the number of correct processes be large enough to overwhelm the faulty ones.

The complexity of Byzantine fault handling increases if the network is not completely connected. In this case, nodes may not communicate directly; they have to rely on intermediate nodes to forward the packets. Faulty nodes may tamper with such forwarding. To counter such faulty behavior, packets are sent along alternative routes. To enable this, the network should be sufficiently connected. In general topology, message transmission is possible only if the network is $2x + 1$-connected [17,26,27], where x is the maximum number of Byzantine faults.

In this paper, we study Byzantine-robust geometric routing in asynchronous networks. We do not use cryptography, randomization or reputation. Instead, we use distributed geometric routing to bypass the faults.

Related work. Sanchez et al [19] and Pathak et al [20] use both cryptographic and reputation-based approaches to secure geometric routing. Adnan et al [28] propose to secure geometric routing through pairwise key distribution. Boulaiche and Bouallouche [29] use message authentication codes to prevent message tampering. Maurer and Tixeuil [30] consider containing faulty Byzantine nodes in control zones such that messages will be sent there only with the authentication form border nodes. Zahariadis et al [31] propose a reputation-based approach to geometric routing security. Several papers discuss counteracting spurious locations reported by faulty nodes to secure geometric routing [32,33,34]. Recently, the problem of consensus has been explored in the geometric setting [35]. Zaz and Nesterenko [36] consider Byzantine-robust geometric routing but offer no solution to the problem. To the best of our knowledge, no Byzantine-robust asynchronous geometric routing algorithm without cryptography, randomization or reputation has been proposed.

Several related problems have been addressed with concurrent geometric routing. In this paper, we consider unicasting: sending a message from a single source to a single target. Alternatively, in multicasting, the same message is delivered to a set of nodes in the network [37]. Sequential geometric multicasting algorithms [38,39,40] optimize message transmission routes by forwarding the same packet to multiple targets for a part of the route. MCFR [41] concurrently sends packets along all the appropriate faces achieving faster delivery at the expense of a greater number of transmitted packets. Another related problem is geocasting; in this problem, the source needs to deliver messages to every node in a particular target area. There are several sequential geocasting algorithms [11,37,42]. Adamek et al [43] present a concurrent geocasting solution. None of these algorithms consider Byzantine tolerance.

Our contribution. We present BeRGeR, an asynchronous unicast concurrent geometric routing algorithm that handles a single Byzantine fault. We assume the source and target nodes are connected by three internally node-disjoint paths that do not intersect the source-target line, formally prove BeRGeR correct under this assumption, and analyze its message complexity.

2 Preliminaries

Communication model. A finite connected graph G is embedded in a geometric plane. Two nodes may not share the same coordinates. We, therefore, use node coordinates for both navigation and node identification. Two nodes adjacent to the same edge are *neighbors*. Each node knows its own coordinates and the coordinates of its neighbors. Neighbors communicate by passing packets. The communication is bi-directional, so the graph G, is undirected. The packet transmission is FIFO, reliable and asynchronous. We assume Lamport's "oral messages" model [17] where the message is not signed but the packet recipient always correctly identifies the sending neighbor.

Nodes are either correct or faulty. A *correct* node operates according to the specified algorithm. A *faulty* node is Byzantine: it may behave arbitrarily including sending packets, dropping received packets or not communicating at all.

If a node sends a packet without receiving it first, the node *originates* the packet. A node *forwards* a packet if it receives it and then sends it to another node. If a node forwards a packet to more than one neighbor, the node *splits* the packet. If a node receives but does not forward the packet, it *drops* the packet. We assume that a faulty node drops all packets that it receives and originates all packets that it sends. The case of a faulty node forwarding a packet correctly is equivalent to dropping the packet and originating an identical packet.

Message transmission, source, target. A *message* is the gainful content to be transmitted by a sequence of forwarded packets. An arbitrary *source* node $s \in G$ is to transmit a message m_s to another arbitrary *target* node $t \in G$. Besides the message, a packet may carry the source and target coordinates as well as other auxiliary information.

To simplify the presentation, we assume that s and t are not neighbors. For our purposes, the message content is immaterial provided that two messages can be compared for equality. The target may receive the message from multiple neighbors, the target *delivers* the message once it passes correctness checks. Forwarding a packet with the same message creates a message path. Consider two message paths from node u to node v. These paths are *internally node-disjoint* if the only nodes the paths share are u and v.

Planarity, faces, traversal. An embedding of a graph is *planar* if its edges intersect only at nodes. For short, we call such a planar embedding a *planar graph.* In a planar graph G, a *face* is a region on the plane such that any pair of its points can be connected by a continuous curve inside the face. If the graph is finite, then the area of all but one face is finite. The finite area faces are *internal* faces and the infinite area face is the *external* face.

To traverse a face of a planar graph, the packets are routed using right- or left-hand-rule. Consider a node $v \in G$ that receives a packet from its neighbor u. In the *right-hand-rule*, v forwards the packet to the next clockwise neighbor after u. In the *left-hand-rule*, v forwards the packet to the next counter-clockwise neighbor after u. We call the obtained traversal paths respectively *right* and *left* and denote them R and L. The right path traverses an internal face counter-clockwise and the external face clockwise. The left path traverses the internal and external faces in the opposite direction to the right path.

Let $G - \overline{st}$ be the graph G, without the edges intersecting the source-target line segment $\overline{st}$. The *green face* is the union of faces that intersect $\overline{st}$. In other words, the green face is the unique face that contains segment $\overline{st}$ in $G - \overline{st}$. This green face in $G - \overline{st}$ may be either internal or external. A *green node* is a node adjacent to the green face. The source and target are thus green nodes. Given a green node k, the *k-blue face* is a union of non-green faces adjacent to k. A *k-blue* node is a non-green node adjacent to the k-blue face.

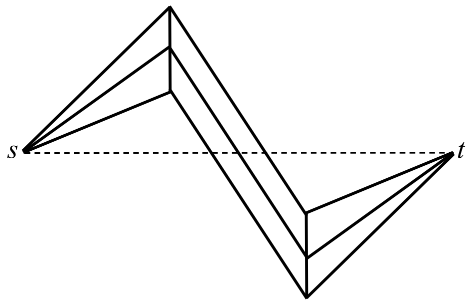

Fig. 1. A graph G, violating the Triconnectivity Assumption. Even though the graph itself is three-connected, $G - \overline{st}$ is disconnected.

The Byzantine-Robust Geometric Routing Problem. An algorithm that solves the Byzantine-Robust Geometric Routing Problem delivers the message m_s sent by the source to the target subject to the following three properties:

Validity: if the target delivers message m_t, then $m_t = m_s$;
Liveness: the target eventually delivers m_t;
Termination: every packet is forwarded a finite number of times.

Fault and connectivity assumptions. We consider a solution to the problem with at most one faulty node $f \in G$. We assume that s and t are correct.

The Byzantine node may originate a spurious message or stop the correct message from propagating. Thus, there need to be at least 3 node-disjoint paths to bypass the faulty node [26]. Hence, we consider 3-connected graphs.

Finding internally node-disjoint paths is difficult even if the graph is 3-connected. Our idea is to spatially separate them: one path uses the left-hand-rule traversal of the green face $G - \overline{st}$ the other — right-hand-rule. To achieve this, when forwarding the message, each green node ignores the edges that intersect $\overline{st}$. However, in general, this may eliminate potential paths to the target or leave the graph entirely disconnected. See Figure 1 for an example. To prevent such disconnect, we posit the following graph connectivity assumption:

Assumption 1 (the Triconnectivity Assumption) *$G - \overline{st}$ is 3-connected.*

3 BeRGeR Description

In this section we present BeRGeR: a Byzantine-Robust Geometric Routing algorithm that solves the Reliable Message Delivery Problem with a single faulty node in a connected planar graph subject to the Triconnectivity Assumption.

Algorithm outline. The algorithm operates as follows. The source concurrently sends two packets, called *cores*, to traverse the green face in the opposite traversal directions. That is, the source sends a left core and a right core.

A green node, i.e. a node adjacent to the green face, may be faulty. The faulty node may send a core with a spurious message. To prevent this, the target waits to receive both cores with matching messages before delivering their message. In this case, however, a faulty green node may drop the packet altogether and prevent message delivery at the target. To counteract this, BeRGeR has a mechanism of bypassing, or skipping, every green node. As the green nodes forward the core along the border of the green face, they add the nodes that the core visited to the packet. In addition, each green node splits the core by sending a *thread* packet that skips the next green node k. This thread packet traverses the union of k-blue and green faces.

A *braid* is a set of threads that carry matching messages in the same direction (L or R). A braid *matches* a core if it contains threads that skip each node that the core has visited and every such thread carries the same message as the core. This way, if there is a faulty green node, at least one thread skips it. Therefore, if a faulty node attempts to drop a packet or originate a forged packet, the target does not collect a matching braid and core. Since the faulty node may be only on the left or right side of the green face, a matching braid and core arrive on the opposite side regardless of the faulty node's actions.

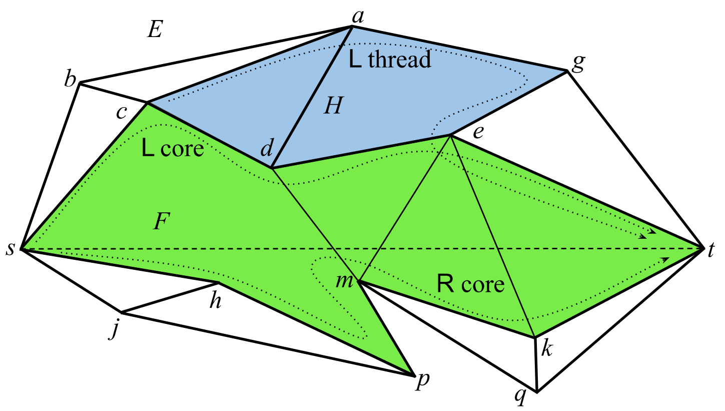

Fig. 2. BeRGeR example operation. L core and R core traverse the green face F. The second L thread, the d-thread, skips node d and traverses the union of the d-blue face H, and the green face F.

Refer to Figure 2 for an illustration of the algorithm operation. The planar graph shown in the picture complies with the Triconnectivity Assumption. Note that greedy routing fails on this graph if the packet reaches node p since p is a local minimum. Face F is green. Nodes s, c, d, e, t, k, m, p and h are adjacent to F. These nodes are, therefore, green. Face H is the d-blue face adjacent to F. Nodes c, a, g, e are d-blue since they are not green but adjacent to the d-blue face H. Face E is the external face of the graph. The source sends two core packets that traverse F: left core L and right core R. The source also sends a left and a right thread that are not shown. The figure illustrates a single L thread that is generated by c when c receives a core. That is, c splits this core and sends thread that skips the next green node d. This thread traverses part of H and then F to reach t.

Algorithm 1: BeRGeR variables and functions

```
1  constants
2     $n$                                         // node coordinates
3     $N$                                         // set of neighbor coordinates
4     $\{L, R\}$                                  // left and right packet direction
5  variables
6     $T$                                         // set of packets received by target
7  functions
8  nextNode($p, s, t, c, k$)
9     $M := \{i \in N : i \neq k$ and onThisSide$(s, t, n, i)\}$
10    return $i \in M : i$ is nearest to $p$ in direction $c \in \{L, R\}$
      // node selection direction L: counter-clockwise, R: clockwise
11 onThisSide($s, t, n, i$)
      // return true if the $n$-$i$ and $s$-$t$ line segments do not intersect
12    return $\overline{ni} \cap \overline{st} = \emptyset$
13 invalid($p, m, s, t, k, \ell$)
14    if $k = p$                                  // sender is skipping itself
15     or $(n = s$ and $k = \perp)$               // a core arrives at $s$
16     or $n \in \ell$ then                       // traveled in a cycle
17        return true
18    else
19        return false
```

Algorithm details. BeRGeR pseudocode is shown in Algorithms 1 and 2. Algorithm 1 shows constants, variables and functions used in BeRGeR. Specifically, each node maintains its own geometric coordinates n and the set of coordinates

Algorithm 2: BeRGeR Actions

```
20 source node, input:
21     $t$                                                              // target
22     $m$                                                              // message
23 source node, initial action:
24     send($m, s, t, \mathsf{R}, \bot, \langle\rangle$) → nextNode($t, s, t, \mathsf{R}, \bot$)        // new R core
25     send($m, s, t, \mathsf{L}, \bot, \langle\rangle$) → nextNode($t, s, t, \mathsf{L}, \bot$)        // new L core
26                       k := nextNode($t, s, t, \mathsf{R}, \bot$)
27     send($m, s, t, \mathsf{R}, k, \langle s\rangle$) → nextNode($t, s, t, \mathsf{R}, k$)          // new R thread
28                       k := nextNode($t, s, t, \mathsf{L}, \bot$)
29     send($m, s, t, \mathsf{L}, k, \langle s\rangle$) → nextNode($t, s, t, \mathsf{L}, k$)          // new L thread

30 every node:
31 receive ($m, s, t, c, k, \ell$) from $p$ ⟶
32     if invalid($p, m, s, t, k, \ell$) then
33         return                                           // drop invalid packets
34     if $k = \bot$ then                                          // if this is a core
35         $\ell$.append($p$)                // append sender to list of visited nodes
36     if $n \neq t$ then                                     // packet is not at target
37         send($m, s, t, c, k, \ell$) → nextNode($p, s, t, c, k$)       // forward packet
38         if $k = \bot$ then                             // if received packet is core
39             $k$ := nextNode($p, s, t, c, \bot$)               // find next green node
40             if $k \notin \ell$ then                               // if $k$ is unvisited
41                 send($m, s, t, c, k, \langle n\rangle$) → nextNode($p, s, t, c, k$)   // new thread
42     else                                                   // packet is at target
           // green nodes neighboring $t$
43         coreR := nextNode($s, s, t, \mathsf{L}, \bot$)
44         coreL := nextNode($s, s, t, \mathsf{R}, \bot$)
           // node(s) neighboring $t$ next to green nodes; may be
              identical
45         threadR := nextNode($s, s, t, \mathsf{L}$, coreR)
46         threadL := nextNode($s, s, t, \mathsf{R}$, coreL)
47         if      $k = \bot$ and $c = \mathsf{R}$ and $p =$ coreR then
48             $T$.add($m, s, \mathsf{R}, \bot, \ell$)                          // record R core
49         else if $k \neq \bot$ and $c = \mathsf{R}$ and $p \in \{$coreR, threadR$\}$ then
50             $T$.add($m, s, \mathsf{R}, k, \langle\rangle$)                       // record R thread
51         else if $k = \bot$ and $c = \mathsf{L}$ and $p =$ coreL then
52             $T$.add($m, s, \mathsf{L}, \bot, \ell$)                          // record L core
53         else if $k \neq \bot$ and $c = \mathsf{L}$ and $p \in \{$coreL, threadL$\}$ then
54             $T$.add($m, s, \mathsf{L}, k, \langle\rangle$)                       // record L thread
55         if $\exists \ell_1, \ell_2 : \{(m, s, \mathsf{L}, \bot, \ell_1), (m, s, \mathsf{R}, \bot, \ell_2)\} \subset T$ and $\ell_1 \cap \ell_2 = \{s\}$ then
56             deliver $m$                                         // matching cores
57         else if $\exists (m, s, c, \bot, \ell) \in T : \forall i \in \ell, (m, s, c, i, \langle\rangle) \in T$ then
58             deliver $m$              // matching core and braid of threads
```

of its neighbors N. Constants L and R denote packet traversal direction. The target node t maintains a set of packets T that it received. The target checks T to see if the received packets satisfy message delivery conditions. Function `nextNode`(p, s, t, c, k) uses the neighbor p of n to select the node according to traversal direction c, either L or R. This selection excludes node k if it is skipped by a thread, and it considers only the neighbors on the same side of $\overline{st}$. This includes s and t themselves. This check is done in function `onThisSide`(s, t, n, i). This function ensures that algorithm packets do not use the edges that cross the green face.

Algorithm 2 shows BeRGeR actions. There are two: the source node initial action that originates the packets (see line 23), and the receipt action taken by every node when it receives a packet (see line 31). As input, the source node s takes target coordinates t and the message m to be delivered to it. In the initial action, the source sends four packets: two cores and two threads. The cores go in left and right directions along the green face. The threads skip the first green nodes in the two directions.

The packet format is as follows: the message m; the source s, and target t; traversal direction L or R; the node k to be skipped or $\perp$ if it is a core packet; and the list of visited nodes. A core packet starts with an empty list. This list is only updated for cores. A thread carries its originator in the visited list.

Let us describe the packet receive action. First, the packet is checked for validity (see line 13). The packet is dropped if it has traveled in a cycle, a core is passing through s, or the sender p, is sending a thread that skips p. Otherwise, if the received packet is a core, p is appended to the visited node list ℓ and further processing depends on whether the packet has arrived at the target.

If the packet recipient is not the target (see line 36), then the node forwards the packet and, if it is a core packet, then the node also sends a thread skipping the next green node, provided that green node is not in the visited list ℓ.

If the packet recipient node is the target (see line 42), the recipient checks that the packet comes from an expected node and then records the receipt of the core or thread packet in T. Then, the target determines if message delivery conditions are met. Specifically, if T has a record of two matching cores or a matching core and a braid. *Matching cores* carry the same message in the opposite traversal directions. *Matching core and braid* are a core and a set of threads such that they are in the same traversal direction, carry the same message and, for every node that the core visited, there is a thread that skips it. If the target receives matching cores or a matching core and a braid, the target delivers the message that they carry.

4 BeRGeR Correctness Proof

Lemma 1. *A core packet traverses a single face of $G - \overline{st}$ and a thread skipping node k traverses a single face of $G - \overline{st} - \{k\}$.*

Proof. By the design of the algorithm, a packet traverses a single face. In forwarding a core packet, each node ignores the edges that intersect $\overline{st}$. If the source originates a core packet, this packet traverses the green face of $G - \overline{st}$. Similarly, thread skipping k originated by the source traverses the union of the green and the k-blue face of $G - \overline{st} - \{k\}$. A faulty node may send a packet in some other face, in which case it traverses that face. $\square$

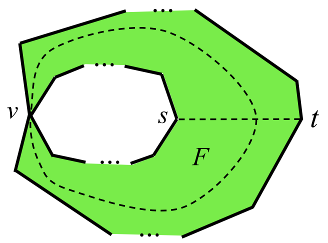

Fig. 3. Illustration for the proof of Lemma 2. If left and right core paths share node v, then v can be connected by a continuous curve that separates s from t. Hence, every path from s to t contains v.

Let *left core path* be the left-hand-rule traversal path on the green face from the source to the target. Similarly, *right core path* be the right-hand-rule traversal path on the green face from s to t.

Lemma 2. *Left and right core paths are internally node-disjoint.*

Proof. We prove the lemma by contradiction. Suppose the opposite: the left and right core paths share an internal node v. See Figure 3 for illustration. In this case, we can draw a closed curve that starts and ends in v and whose interior is inside the green face. This curve separates the plane into two areas: one of them contains s and the other t. This means that every path from s to t contains v. However, the Triconnectivity Assumption states that $G - \overline{st}$ is three-connected. This means that there must be at least three internally node-disjoint paths between s and t. Hence, our initial supposition is not correct. Therefore, left and right core paths must be internally node-disjoint. $\square$

Let *left core* and *right core* are packets following left and right core paths respectively.

Lemma 3 (Core validity). *If the target receives a left core and a right core carrying messages m_{lc} and m_{rc} respectively and $m_{lc} = m_{rc}$, then $m_{lc} = m_s$.*

Proof. The target receives a core packet from a node adjacent to the green face. According to Lemma 1, such packet traverses the green face only. According to Lemma 2, left and right core paths are disjoint. Since there is at most one fault in the network, at least one of these paths is fault-free. Therefore, either left or right core packets are forwarded by correct nodes only. In this case, it carries the message sent by the source. Hence, if the target receives two identical messages from both left and right core paths, this message is sent by the source. □

Lemma 4 (Thread validity). *If a thread skips a green node k and reaches a correct node, it is not originated by k.*

Proof. Only the source and the faulty node may originate threads. We assume that the source is correct. Therefore, we only have to consider the case of k being faulty. Note that k may potentially originate a thread that skips k, or may originate a core that splits into a thread that skips k. If k originates a thread that skips itself, then, by the design of the algorithm (see line 14), the recipient does not forward it.

Consider now the case where k originates a core. In this case, this core contains k in the list of visited nodes ℓ. Therefore, no correct node that receives this core sends a thread that skips k (see line 40). In either case, the thread that skips k is not originated by k. □

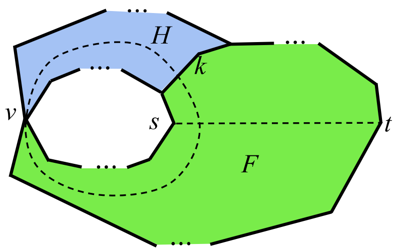

Fig. 4. Illustration for the proof of Lemma 5. If a blue face H, bypassing green node k contains a green node v, that lies on the right core path, then v can be connected by a continuous curve inside $F \cup H$ that separates s from t and, hence, every path from s to t either contains k or v.

Lemma 5. *The path traversed by a left thread does not contain any node of the right core path. Similarly, the path traversed by the right thread does not contain nodes of the left core path.*

Proof. We prove the lemma for the left thread. The argument for the right thread is similar.

Let us consider a left thread that skips some node k. According to Lemma 1, the thread either traverses the left core path or the k-blue face. Due to Lemma 2, the right and left core paths are internally node-disjoint. Hence, the left core path does not contain right core path nodes.

Let us now consider the nodes adjacent to the k-blue face. We prove this part by contradiction. Suppose the opposite: there is a node v that is adjacent to the k-blue face but lies on the right core path. We show that in this case all paths from s to t contain either v or k. Specifically, we show that the path that does not contain k, goes through v.

Indeed, removing node k and adjacent edges merges the green face and the k-blue face. See Figure 4 for illustration. In this figure, F is the green face and H is the k-blue face. In this case, inside this joint face, we can draw a closed curve that starts and ends in v. Therefore, any path from s to t that does not contain k, has to go through v. That is, all paths from s to t contain either v or k.

However, the Triconnectivity Assumption requires that there are three internally node-disjoint paths from s to t. Hence, our supposition is incorrect and the path traversed by the left thread, either along the left core path or along the k-blue face, does not contain a node on the right core path. □

Recall, that a *braid* is a set of threads with the same direction, left or right, and carrying the same message. A braid *matches* a core packet c if (i) c and the braid carry the same message and (ii) for every node i that c carries in its visited list ℓ there is a thread in the braid that skips i.

Lemma 6 (Braid validity). *If the fault is adjacent to the green face and the target receives a core and a matching braid carrying m_b, then $m_b = m_s$.*

Proof. We prove the lemma for the left direction. The argument for right direction is similar. The fault may be on the left side of the green face or on the opposite side. We consider these cases separately.

Let the target receive a left core and matching left braid while the fault is on the right side. In this case, according to Lemma 2, all the nodes of the left core path are correct. By the design of the algorithm (see line 47), the target accepts a core packet only if it comes from a neighbor adjacent to the green face. Due to Lemma 1, this core traverses the green face only. Since the fault is on the right core path, this left core was forwarded by correct nodes only. Therefore it carries m_s.

Let us now consider the case where the target receives a left core and a matching left braid while the fault is also on the left core path. Let f be the faulty node; that is, the faulty node is on the received core path. A correct recipient that forwards the packet records the packet sender in the visited list ℓ of the packet. Therefore, f is present in ℓ of the core packet that the target receives. Since the target receives a braid that matches this core packet, it also receives a thread that skips f. According to Lemma 4, f may not originate such a thread. Hence, the thread that skips f is originated by the source so it carries m_s. Since this thread is in the braid that carries the same message and matches the core message, they all carry m_s. □

Lemma 7 (Liveness). *The target eventually receives either (i) matching left and right core packets or (ii) a matching core and a braid.*

Proof. If the faulty node is not adjacent to the green face, then both matching left and right core packet reach the target.

Let us examine the case of a faulty node is adjacent to the green face. This means that it lies on a core path. Assume, without loss of generality, that the node is on the right core path. Then, according to Lemma 2, the left core path is fault-free. Moreover, due to Lemma 5 the paths of all the left threads are also fault-free. That is, in case the faulty node is on the right core path, the left core and a matching left braid reach the target. □

Lemma 8 (Termination). *Every packet is forwarded a finite number of times.*

Proof. According to Lemma 1, every packet traverses a single face in either $G - \overline{st}$ or $G - \overline{st} - \{k\}$. The originator of the packet is recorded in the visited list ℓ of the packet. When received, the originating node drops such a packet. That is, if the packet is not dropped by the source, target or the faulty node, it is dropped by the originating node once the packet traverses the entire face.

The faulty node may record a spurious packet originator in ℓ. However, according to Lemma 1, this packet traverses some face, arrives back to the faulty node, where it is assumed to be dropped.

In a finite graph, this means that every packet is forwarded a finite number of times. □

Theorem 1. *BeRGeR: Byzantine Robust Geometric Routing algorithm solves the Reliable Message Delivery Problem with a single Byzantine node in a planar graph subject to the Triconnectivity Assumption.*

Proof. In BeRGeR, the target delivers message m_t in two cases: either it receives two matching core packets or it receives a matching core packet and a braid. According to Lemma 3, if the target receives matching core packets, the packets carry the message sent by the source. According to Lemma 6, if the target receives a matching core and a braid, they carry the source message also. That is, the target delivers only the message sent by the source. Hence, BeRGeR satisfies the Validity Property of the Reliable Message Delivery Problem.

Moreover, Lemma 7 guarantees that the target eventually receives matching cores or a matching core and a braid. That is, BeRGeR guarantees that a message is delivered by the target, which means that the algorithm also satisfies the Liveness Property.

Lemma 8 shows that BeRGeR satisfies the Termination Property as well. □

5 Constant Packet Size Extension and Complexity Estimate

Constant packet size extension. In BeRGeR, a core packet carries the path that it travels, making the packet size potentially linear with respect to the network size. However, the modification to constant size cores is relatively straightforward. This modification is achieved at the expense of stateless packet forwarding. For that, the sender transmits the message in numbered fixed-size packets to the neighbor. The neighbor receives the packets and reassembles the message. If any of the packets are missing, the whole message is discarded. Since we assume no packet loss, a correct node transmits all packets. The faulty node either transmits the packets or fails to do so. The latter is equivalent to no packet transmission of the original algorithm. The correctness of the original BeRGeR is preserved.

Algorithm message complexity. Let us analyze the message complexity of BeRGeR during its fault-free operation. Let N and E be the respective number of nodes and edges in graph G. In the core path, each node may appear at most once. Hence, the length of the core path is in $O(N)$. Each core message transmission may be separated into $O(N)$ constant-size packets. Since the path of the core message is in $O(N)$, the total number of packets for a single core is in $O(N^2)$.

For each node on the core path, the algorithm generates a thread. Each thread travels at most E edges and the threads are constant size. Therefore, the number of thread packet transmissions generated by a single core is in $O(EN)$. There are left core and right core. Hence, the total number of sent packets is in $O(2N^2 + 2EN)$. In a planar graph, $E \in O(N)$ by Euler's formula. Thus, the overall BeRGeR message complexity is in $O(N^2)$.

6 Future Work

As described in this paper, BeRGeR is a unicast algorithm. To achieve Byzantine fault tolerance, it employs the same message concurrency techniques that are used to solve geocasting [43] and multicasting [41]. We expect BeRGeR can be adapted in a straightforward manner to produce Byzantine-robust solutions to these two problems.

BeRGeR requires the Triconnectivity Assumption to operate correctly. It states that the subgraph $G - \overline{st}$ needs to be 3-connected. In general, to enable fault tolerant transmission, any graph G needs to be $(2x + 1)$-connected, where x is the maximum number of faults. However, we are unsure whether this needs to hold for the subgraph $G - \overline{st}$. Trying to relax this assumption would be an interesting research pursuit.

Byzantine-robust routing needs the communication graph to be three-connected. The maximum planar graph connectivity is 5. Thus, potentially, such a graph may enable a geometric routing algorithm that can tolerate up to 2 Byzantine faults. Finding such an algorithm is another research challenge.

BeRGeR assumes planar subgraph for its operation. There are several articles that extend geometric routing to non-planar graphs [44,45,46]. It would be interesting to investigate whether these techniques are applicable to BeRGeR.

It is notoriously difficult to do performance evaluation of Byzantine tolerant algorithms as Byzantine behavior is difficult to simulate. Indeed, rather than erratically dropping messages or skipping steps, the faulty nodes may collude to cause maximum damage at the weakest point of the algorithm [47]. However, it would be interesting to compare the performance of BeRGeR to non-tolerant algorithms to evaluate the expense of adding fault-tolerance to geometric routing algorithms.

Acknowledgments

We would like to thank Sam Kosco for many helpful discussions leading to the algorithm in its present form.

References

1. Tomasz Imieliński and Julio C Navas. Gps-based geographic addressing, routing, and resource discovery. *Communications of the ACM*, 42(4):86–92, 1999.
2. Young-Bae Ko and Nitin H Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 101–110. IEEE, 1999.
3. Young-Bae Ko and Nitin H Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.
4. Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *in Proc. 11 th Canadian Conference on Computational Geometry*. Citeseer, 1999.
5. David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996.
6. Charles Perkins and Elizabeth Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.
7. Georgios Karagiannis, Onur Altintas, Eylem Ekici, Geert Heijenk, Boangoat Jarupan, Kenneth Lin, and Timothy Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *IEEE Communications Surveys & Tutorials*, 13(4):584–616, 2011.
8. A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: a wireless sensor networking for target detection, classification, and tracking. *Computer Networks, Special Issue on Future Advances in Military Communication and Technology*, 46(5):605–634, December 2004.
9. Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless networks*, 7(6):609–616, 2001.
10. P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proc. of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 727–738, 1979.
11. P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *The Journal of Mobile Communication, Computation and Information*, 7(6):48–55, 2001.
12. K Ruben Gabriel and Robert R Sokal. A new statistical approach to geographic variation analysis. *Systematic Biology*, 18(3):259–278, 1969.
13. B. Karp and H.T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobille Computing and Networking (MobiCom)*, pages 243–254. ACM Press, August 2000.
14. Godfried T Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

15. Fabian Kuhn, Rogert Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 63–72, 2003.
16. Thomas Clouser, Mark Miyashita, and Mikhail Nesterenko. Concurrent face traversal for efficient geometric routing. *Journal of Parallel and Distributed Computing*, 72(5):627–636, May 2012.
17. Leslie Lamport, Robet Shostak, and Marrshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
18. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.
19. Adrián Sánchez-Carmona, Sergi Robles, and Carlos Borrego. Privhab+: A secure geographic routing protocol for dtn. *Computer Communications*, 78:56–73, 2016.
20. Vivek Pathak, Danfeng Yao, and Liviu Iftode. Securing location aware services over vanet using geographical secure path routing. In *2008 IEEE International Conference on Vehicular Electronics and Safety*, pages 346–353. IEEE, 2008.
21. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
22. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.
23. Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.
24. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.
25. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.
26. Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.
27. Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1:26–39, 1986.
28. Ali Idarous Adnan, Zurina Mohd Hanapi, Mohamed Othman, and Zuriati Ahmad Zukarnain. A secure region-based geographic routing protocol (srbgr) for wireless sensor networks. *PloS one*, 12(1):e0170273, 2017.
29. Mehdi Boulaiche and Louiza Bouallouche-Medjkoune. Hsecgr: highly secure geographic routing. *Journal of Network and Computer Applications*, 80:189–199, 2017.
30. Alexandre Maurer and Sébastien Tixeuil. Containing byzantine failures with control zones. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):362–370, 2014.
31. Theodore Zahariadis, Panagiotis Trakadas, Helen C Leligou, Sotiris Maniatis, and Panagiotis Karkazis. A novel trust-aware geographical routing scheme for wireless sensor networks. *Wireless personal communications*, 69:805–826, 2013.
32. Mariano García-Otero, Theodore Zahariadis, Federico Alvarez, Helen C Leligou, Adrián Población-Hernández, Panagiotis Karkazis, and Francisco J Casajús-Quirós. Secure geographic routing in ad hoc and wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2010:1–12, 2010.
33. Tim Leinmüller, Christian Maihöfer, Elmar Schoch, and Frank Kargl. Improved security in geographic ad hoc routing through autonomous position verification. In

Proceedings of the 3rd international workshop on Vehicular ad hoc networks, pages 57–66, 2006.

34. Adnan Vora and Mikhail Nesterenko. Secure location verification using radio broadcast. *IEEE Transactions on Dependable and Secure Computing*, 3(4):377–385, 2006.
35. Joseph Oglio, Kendric Hood, Gokarna Sharma, and Mikhail Nesterenko. Byzantine geoconsensus. In *International Conference on Networked Systems*, pages 19–35. Springer, 2021.
36. Brown Zaz and Mikhail Nesterenko. Using braids for byzantine resistant geographic routing on polyhedral networks. In *Joint Mathematics Meet*, San Francisco, USA, Jan 2024.
37. Juan A. Sánchez, Pedro M. Ruiz, and Ivan Stojmenovic. GMR: Geographic multicast routing for wireless sensor networks. In *3d IEEE Communication Society Conference on Sensors and Ad Hoc Communications and Networks (SeCon)*, pages 20–29. IEEE, September 2006.
38. Martin Mauve, Holger Füßler, Jörg Widmer, and Thomas Lang. Position-based multicast routing for mobile ad-hoc networks. *Mobile Computing and Communications Review*, 7(3):53–55, 2003.
39. Shibo Wu and K. Selcuk Candan. GMP: Distributed geographic multicast routing in wireless sensor networks. In *26th IEEE International Conference on Distributed Computing Systems (26th ICDCS'06)*, page 49, Lisboa, Portugal, July 2006. IEEE Computer Society.
40. Kai Chen and Klara Nahrstedt. Effective location-guided tree construction algorithms for small group multicast in MANET. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM-02)*, volume 3 of *Proceedings IEEE INFOCOM 2002*, pages 1180–1189, Piscataway, NJ, USA, June 23–27 2002. IEEE Computer Society.
41. Jordan Adamek, Mikhail Nesterenko, James Scott Robinson, and Sébastien Tixeuil. Concurrent geometric multicasting. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018.
42. Jie Lian, Kshirasagar Naik, Yunhao Liu, and Lei Chen. Virtual surrounding face geocasting with guaranteed message delivery for ad hoc and sensor networks. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 198–207. IEEE, 2006.
43. Jordan Adamek, Mikhail Nesterenko, James Scott Robinson, and Sébastien Tixeuil. Stateless reliable geocasting. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 44–53. IEEE, 2017.
44. Thomas Clouser, Adnan Vora, Timothy Fox, and Mikhail Nesterenko. Void traversal for efficient non-planar geometric routing. *Ad hoc networks*, 11(8):2345–2355, 2013.
45. Fabian Kuhn, Rogert Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 69–78, 2003.
46. Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 217–230, 2005.
47. Kevin Driscoll, Brendan Hall, Håkan Sivencrona, and Phil Zumsteg. Byzantine fault tolerance, from theory to reality. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2003.