# EFFICIENT TOPOLOGY MANAGEMENT AND GEOGRAPHIC ROUTING IN HIGH-CAPACITY CONTINENTAL-SCALE AIRBORNE NETWORKS

Benjamin D. Newton

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2017

## ABSTRACT

Benjamin D. Newton: Efficient Topology Management and Geographic Routing
in High-Capacity Continental-Scale Airborne Networks
(Under the direction of Kevin Jeffay and Jay Aikat)

Large-scale high-capacity communication networks among mobile airborne platforms are quickly becoming a reality. Today, both Google and Facebook are seeking to form networks among high-flying balloons and drones in an effort to provide Internet connections from the stratosphere to users on the ground. This dissertation proposes an alternative, namely using the cargo and passenger aircraft already in the skies as the principal components of such a network. My work presents the design of a network architecture to overcome the challenges of managing the topology of and routing data within these continental-scale highly-dynamic networks.

The architecture relies on directional communication links, such as free-space optical communication links (FSO), to achieve high data rates over long distances. However, these state-of-the-art communication systems present new networking challenges. One such challenge is that of managing the physical topology of the network. Such a topology must be explicitly managed, ensuring that each directional data link is pointed at and connected with an appropriate neighbor (which is also pointing back) to yield an acceptable global topology. To overcome this challenge, a distributed topology management framework and associated topology generation algorithms were designed, implemented, and tested via simulation. The framework is capable of managing the topology of thousands of nodes in a continental-scale airborne network and has no communication overhead except that required to exchange position information among nearby nodes.

A second component of the work concerns routing data at high data rates through a constantly changing network topology. To address this issue Topology Aware Geographic Routing (TAG), a position-based routing protocol was developed that strategically uses local topology information to make better local forwarding decisions, decreasing the number of hops required to deliver a packet, when compared with other geographic routing protocols. In addition, unlike other similar protocols, TAG is able to reliably deliver packets even when the topology changes while the packet is in flight.

These protocols are tested and validated in a series of simulations where nodes trace the trajectories recorded from thousands of actual flights. These simulations indicate that the topology management framework and TAG are able to perform well in large-scale high-density conditions, over long durations, and are able to support tens of thousands of 1 Mbps flows.

To my sweetheart Erin, without whom this work would not have been possible,
and to my children Libby, Emma, and Samuel, without whom this work would not have been worthwhile.

## ACKNOWLEDGEMENTS

It is impossible for me to appropriately express my gratitude to the numerous individuals who have made this work possible, but I will endeavor to list here some of the people who have made this work possible.

In academia, there exists an academic family tree where Ph.D. recipients can trace their academic roots. One can search for their adviser and be presented with a line of academic forefathers stretching back in time. Unlike most Ph.D. students, it has been my pleasure to have two academic ”parents” who have guided my dissertation research: Kevin Jeffay and Jay Aikat. Kevin served as my adviser and Jay as my co-adviser. Though it took some convincing initially, Kevin has always supported my cutting-edge research that is, as he says, on the “lunatic fringe”. Kevin helped me not get too lost in the “weeds” of my research and consistently gave great advice about how to proceed at each step of the process. His expert advice and wisdom guided and shaped every aspect of my research.

Jay, having herself graduated as one of Kevin’s students, was able to bridge the gap between student and adviser. She attended almost every weekly research meetings and provided needed insights, encouragement, and support for my research. Though Kevin was appointed as the department chair, and Jay accepted a top position at RENCI while serving as my advisers, they both always gave the highest priority to my needs even though they had many other important duties.

The other members of my dissertation committee include: Jasleen Kaur, Ketan Mayer-Patel and Ron Alterovitz. As I met with each of them over the course of my research they gave me ideas and suggestions that shaped and improved my research. Thank you to each of them for their service and support. Thank you to all the staff of the Computer Science department, especially Murray Anderegg, John Sopko, and Missy Wood, for their help and support. Thank you also to Gary Bishop and Fred Brooks with whom I had the privilege of meeting regularly in a Bible study group. These weekly doses of wisdom, wit, and worship are among the highlights of my time in Chapel Hill.

I must also thank Andy Wilson and Danny Rintoul from Sandia National Labs. Andy was, like myself, doing interesting research using aircraft trajectories when we met. We both saw an opportunity to collaborate that couldn’t be passed up. Consequently, I spent a summer in Albuquerque working on improving the prediction capabilities of Tracktable, an open source software library for processing and analyzing trajectory data that has become an integral part of my dissertation research.

I am grateful for my wife's sister Amelia Robinson for her expert editing advice, and for my wife's parents, Brion and Deborah Robinson, for their continued support of me, my family, and our dreams. Among many other things, they sacrificed frequent-flyer miles to enable regular visits, and drove across the continent more than once. They even watched our kids for several days so my wife and I could attend a conference. Thank you to them for their continued support.

It is hard to express the gratitude I feel for my parents Dave and Sandy Newton. They raised an inquisitive boy and have shaped and molded him into a scholar. Without their continued support, this work would not have been possible. Thank you, Mom and Dad, for the packages from home, for your regular visits, and for your generous gifts! Thank you also, to my sister, Stacy Schultz, and her husband Brian for always believing in me.

My children have given up many evenings and Saturday afternoons when they wish they could have been with Dad. Thank you to my 11-year-old Libby, for helping find acronyms in my manuscript and for her interest and support of my research. Thank you to my 6-year-old Emma for squealing "Daddy" and running to give me a hug upon my arrival at home. And thank you to my 1-year-old Samuel, who always tried to help write my dissertation by typing on the keyboard whenever I set my laptop down.

Lastly, thank you to my supportive, sincere, sensitive, steadfast, sympathetic, super, sunny, softhearted, sweetheart Erin. She gave me the daily courage, motivation, and inspiration I needed to tackle this impossible dream. For her love and support, I will be eternally grateful.

## TABLE OF CONTENTS

## LIST OF TABLES

**LIST OF FIGURES**

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ACK | acknowledgment |
| AODV | Ad-hoc On-demand Distance Vector |
| AFR | Adaptive Face Routing |
| AO | Adaptive Optics |
| ASDI | Aircraft Situation Display to Industry |
| ATM | Air Traffic Management |
| ATC | Air Traffic Control |
| AAL | American Airlines |
| ADS-B | Automatic Dependent Surveillance - Broadcast |
| BER | Bit Error Rate |
| BFR | Bounded Face Routing |
| CPU | Central Processing Unit |
| CT | Central Time |
| DIR | Compass Routing |
| CBTC | Cone Based Topology Control |
| CBR | Constant Bit Rate |
| CLDP | Cross-link Detection Protocol |
| CDF | Cumulative Distribution Function |
| CTR | Critical Transmitting Range |
| dBmW | decibels-milliWatts |
| DARPA | Defense Advanced Research Projects Agency |
| DC-DT | Degree-Constrained Delaunay Triangulation |
| DC-GG | Degree-Constrained Gabriel Graph |
| DCP-KRUSKAL | Degree-Constrained Planar Kruskal |
| DCP-TRT | Degree-Constrained Planar Tree-based Reliable Topology |
| DT | Delaunay Triangulation |
| DAL | Delta Airlines |
| DoD | Department of Defense |

| | |
|---|---|
| DSDV | Destination Sequenced Distance Vector |
| DSR | Dynamic Source Routing |
| ECEF | Earth-centered Earth-fixed |
| EIGRP | Enhanced Interior Gateway Routing Protocol |
| ERAST | Environmental Research Aircraft and Sensor Technology |
| ENY | Envoy |
| FR | Face Routing |
| FAA | Federal Aviation Administration |
| FCC | Federal Communications Commission |
| FEC | Forward Error Correction |
| FSO | Free-Space Optics |
| FSOC | Free-Space Optical Communication |
| FOENEX | Free-space Optical Experimental Network Experiment |
| GG | Gabriel Graph |
| GLSR | Geographic Load Share Routing |
| Gb | Gigabit |
| Gbps | Gigabits per second |
| Gb/sec | Gigabits per second |
| GB | Gigabyte |
| GIG | Global Information Grid |
| GPS | Global Positioning System |
| GPSR | Greedy Perimeter Stateless Routing |
| GR | Greedy Routing |
| GFG | Greedy-Face-Greedy |
| GOAFR | Greedy Other Adaptive Face Routing |
| GOAFR+ | Greedy Other Adaptive Face Routing Plus |
| GPVFR | Greedy Path Vector Face Routing |
| ID | Identifier |
| INS | Inertial Navigation System |

| | |
|---|---|
| IP | Internet Protocol |
| Kbps | kilobits per second |
| KB | Kilobyte |
| km | kilometer |
| LED | light-emitting diod |
| LOS | line of sight |
| LTE | Long-Term Evolution |
| LLCD | Lunar Laser Communications Demonstration |
| LMST | Local Minimum Spanning Tree |
| LTRT | Local Tree-based Reliable Topology |
| MTU | maximum transmission unit |
| MAC | media access control |
| Mb | Megabit |
| Mbps | Megabits per second |
| MB | Megabyte |
| ms | Millisecond |
| MST | Minimum Spanning Tree |
| MANET | mobile ad-hoc network |
| MFR | Most Forward within Radius |
| MPR | Multipoint Relay |
| MPLS | Multiprotocol Label Switching |
| NASA | National Aeronautics and Space Administration |
| NAS | National Airspace System |
| NCO | Network Centric Operation |
| OPVFR | Oblivious Path Vector Face Routing |
| OSPF | Open Shortest Path First |
| OAGC | Optical Automatic Gain Control |
| ORCA | Optical RF Communications Adjunct |
| OLSR | Optimized Link State Routing |

| | |
|---|---|
| OAFR | Other Adaptive Face Routing |
| OBFR | Other Bounded Face Routing |
| OFR | Other Face Routing |
| PT | Pacific Time |
| PDR | Packet Delivery Ratio |
| PER | Packet Error Rate |
| PVEX | Path Vector Exchange Protocol |
| PAT | Position Acquisition and Tracking |
| PIF | power in the fiber |
| RF | radio frequency |
| RNG | Relative Neighborhood Graph |
| RERR | Route Error |
| RREP | Route Reply |
| RREQ | Route Request |
| s | Second |
| SINR | Signal to Interference and Noise Ratio |
| SWA | Southwest Airlines |
| TAG | Topology Aware Geographic Routing |
| TCP | Transmission Control Protocol |
| TRT | Tree-based Reliable Topology |
| TTL | Time to live |
| UDT | Unit Delaunay Triangulation |
| UDG | Unit Disk Graph |
| UK | United Kingdom |
| UPS | United Parcel Service |
| UTC | Coordinated Universal Time |
| UAV | Unmanned Aerial Vehicle |
| UDP | User Datagram Protocol |

# CHAPTER 1: INTRODUCTION

As wireless digital communication networks mature and advance, their reach has been extending into previously unimaginable regions, including outer space [BSM$^+$09], under water [DSSM15], and underground (where the propagation medium is the soil) [CaRCR15]. Yet even more fascinating has been the recent surge of interest in forming networks in the Earth's stratosphere, that has propelled the field of Airborne Networking from its origins inside military research labs out onto the center stage. Well-known companies like Google and Facebook are making huge investments in their own large-scale airborne networks as they race to define and control these futuristic networks in the skies.

An airborne network is a digital communication network which includes nodes, such as balloons, blimps, or fixed-wing aircraft, that are airborne. Among the many challenges associated with these highly-dynamic networks is that of forming long-range high-bandwidth connections between them, and efficiently routing data over those connections in a manner that is scalable to potentially tens of thousands of nodes. Directional data links are data links which focus their transmissions in a certain direction. These links can facilitate the required high-rate connections over long ranges, but their use violates many of the assumptions made by traditional networking protocols. Further, these directional data links require explicit management of the network topology, so that it can quickly adapt and evolve as nodes join, swiftly move within, and leave the network. These frequent changes to the network topology induce large amounts of routing overhead and packet loss even in routing protocols designed for mobile networks. This dissertation details a distributed topology management framework and a geographic routing algorithm capable of efficiently routing high-bandwidth traffic in a large-scale airborne network connected by directional data links. The effectiveness of these algorithms is demonstrated through a series of experiments that simulate the formation of an airborne network of commercial aircraft tracing the paths of actual flights in the United States airspace.

## 1.1 Airborne Networks of Balloons, Drones, and Commercial Aircraft

Thirty-four million Americans (10% of the United States population) have no access to high-speed Internet at home [FCC16]. No provider is able or willing to provide high-rate connections to these people's homes, most of which are in very rural areas. The situation is much worse in developing countries, where it is estimated that 31% of the population do not even live within range of a 3G mobile network [Fac16]. Building the traditional infrastructure required to connect these users to the Internet is an extremely expensive and slow process; land rights must be purchased, cables must be laid or towers erected, and costly communications equipment must be installed. These barriers may be one of the reasons that Facebook and Google are looking to the stratosphere for a creative solution to the problem of connecting more users (and customers) to the Internet.

In June 2013 Google announced their plans to create an airborne network of high-flying balloons to provide internet service to users on the ground throughout the world [TP13], [DTBW13]. Facebook quickly followed suit, purchasing UK-based Ascenta [Gar14], a maker of solar-powered drones, and announcing that they were working on ways to beam Internet to people from the sky [Kel14]. Since that time both companies have made significant progress toward their goals, with Google launching balloons in New Zealand, California, and Brazil [Sim16], and Facebook testing its full-scale solar-powered Internet drone that may one day stay aloft for three to six months[New16].

Although both approaches are unique (each with its own challenges), the fundamental idea is the same: form a high-capacity backbone network between airborne nodes, connect it to ground station gateways, and provide Internet connections from each node to users on the ground (using an additional communication system). A common challenge to either proposal, however, is the sheer scale of such a network. According to Google, each of the balloons in their proposed architecture can provide coverage to a ground area with a diameter of about 40 km [Mur15]. This means that to actually provide Internet connections to just the continental United States, Google would need over 6,000 precisely-positioned balloons.

A fundamentally different approach than that being pursued by Google and Facebook is the idea of using the existing fleet of passenger and cargo aircraft as the principal components of an airborne network. Doing so would alleviate the need to send so many drones or balloons into the stratosphere and would have the added benefit of enabling high-speed Internet on board the aircraft, with connections far superior to the low-rate and latency prone services offered today. However, regardless of whether drones, balloons, commercial aircraft, or perhaps some mixture of all three are used as nodes in a future airborne network, the fundamental challenge remains effectively managing the topology of the network and efficiently routing data within that dynamic network topology.

## 1.2 Free-Space Optical Communication Links

Free-space optical communication (FSO) is expected to be an essential technology in achieving the high-bandwidth connections over long distances required for the airborne network backbone. Google is investigating the use of ultra-bright light-emitting diodes (LEDs) for communication between balloons [DTBW13], and Facebook claims to have developed free-space laser transceivers that improve upon the state-of-the-art by an order of magnitude, yielding data rates in the tens of Gbps [Mag]. The United States Armed Forces are also linking aircraft together with low-power infrared lasers. A recently completed Defense Advanced Research Projects Agency (DARPA) program demonstrated air-to-air connections with ranges up to 200 km and air-to-ground connections with slant ranges (line-of-sight distance between data links at different altitudes) up to 130 km while maintaining data rates from 3 to 9 Gb/sec using a hybrid FSO/RF link [DOD13], [SPM$^+$11], [APC$^+$12].

Conveniently, free-space optical communication links do not require a Federal Communications Commission (FCC) license or spectrum allocation, and are smaller, lighter, more secure, and more power efficient than high-bandwidth radio frequency (RF) links. FSO links, however, are very sensitive to atmospheric turbulence, and line-of-sight obstructions (such as clouds). It has been shown that a combination of Adaptive Optics (AO), Optical Automatic Gain Control (OAGC), and Forward Error Correction (FEC) can overcome the effects of the severe atmospheric turbulence which impacts these links [SPM$^+$11]. Other techniques such as link layer retransmissions can further improve the link quality. Note, also, that atmospheric turbulence and cloud obstructions are less prominent at high altitudes where it is anticipated that the bulk of the airborne network connections will be formed. In addition, the topology management framework (discussed next) could take cloud positions into account, and route around them as in [JRO$^+$09].

Unlike standard omnidirectional wireless RF links, that transmit in all directions and connect to all nodes within range, FSO links focus their transmission in a specific direction. Thus, a pair of FSO links must be pointed at one another to form a point-to-point wireless connection. An aircraft that has exactly three FSO links must determine which three aircraft to point its links at and connect with, among the dozens that may be within range. Somehow the topology of the network must be explicitly managed so that nodes know which FSO links should be pointed at one another to form the network's connections.

## 1.3 Distributed Explicit Topology Management

In traditional wireless networks, where omnidirectional communications are the norm, topology control involves coordinating the transmission power of network nodes to generate a topology with certain properties [San05]. Topology control employed on a sensor network, for example, may be used to manage a network such that connectivity to all nodes is maintained, but power consumption is minimized. The use of directional FSO links drives the need for a topology management framework which is related to topology control, but which explicitly determines which nodes should point to and connect with one another. In addition, when using directional links, the degree, or number of incident edges at a node in the topology, is limited to the number of physical directional data links at that node. Thus, the topology management framework must also guarantee that degree constraints are met while seeking to form a desirable global topology.

Imagine a centralized topology management framework where, once connected, all nodes would send their position and connection information through the network to a selected control node. This special control node could determine the best global topology for the network and then command each node to make appropriate connectivity changes to realize that topology. This design, however, has several disadvantages besides the fact that it presupposes the connectivity of the nodes. First, it creates a single point of failure at the control node for the network. Next, significant amounts of extra overhead would be required to send the position information of all nodes to the control node, potentially causing congestion on links near the control node. Further, this design places much of the computational burden on the single control node, requiring it to compute a global topology that includes all the nodes. Lastly, and implicit in each of the previous disadvantages, is the issue of scalability. As the number of nodes in a network managed by such a scheme increase, the bandwidth would eventually be completely consumed in just managing the topology of the network. Clearly, a distributed solution is warranted, especially when networks of tens of thousands of nodes are desired.

If, instead, nodes could exchange their position information with only nearby nodes, and each node could use only that local information to compute and implement a local topology, the result would be an efficient and much more scalable management of the topology. The computation load would be spread across the network. There would no longer be a single point of failure, and there would be no need for some handshaking procedure to explicitly negotiate connections between the nodes. The only communication overhead of such a scheme, in fact, would be the cost of exchanging position information with nearby nodes (something already done by commercial aircraft using a separate system). In addition, the only associated "costs" could be a slight decrease in the number of connections formed, leaving some links unused, and an increase in the frequency of connection changes. Could the topology actually be managed in this way? One of the main contributions of this work is a topology management framework which operates exactly as described above. The design of this distributed management framework and the topology generation algorithms it requires are detailed in Chapter 3.

## 1.4 Routing in Large-Scale Airborne Networks

A network formed by connecting aircraft using a limited number of directional data links is necessarily highly-dynamic. Frequent changes to the topology are required as nodes join, move within, and leave the network. Nodes that are not directly connected in this dynamic topology must communicate by relaying their messages through a series of intermediate nodes, each acting as a router. A routing protocol or algorithm is needed to determine, at each node along that path, where the message should be forwarded next.

Mobile Ad-hoc network (MANET) routing protocols are designed to support networks formed among mobile nodes, but even these protocols often fail to efficiently route packets in highly-dynamic, large-scale networks [NAJ15]. Consider, for example, a network of ten thousand mobile nodes, and two nodes on opposite edges of this network attempting to communicate. A reactive MANET routing protocol (one that seeks to establish routes "on demand"), will need to buffer packets until it can find a path through the network to the distant node. Once a path is found packets may begin flowing. However, unless the nodes along the path are essentially stationary, there is a high probability the path will quickly become invalid. All it takes to make the path obsolete is one node along the path to leave the network or move out of range of an adjacent node supporting the path. Packets must then be delayed or dropped while the path is repaired, requiring extra management packets be sent, and increasing overhead.

A proactive routing protocol, that seeks to always maintain routes between every pair of nodes, could be used instead, but this only exacerbates the situation, with the protocol generating what has been described as "torrents" of link status change messages in an effort to recover from any failure on any path [KK00]. The overhead associated with these messages can quickly saturate the available network bandwidth for large networks with a frequently changing topology. Despite the high cost, only a fraction of the packets may actually arrive at their intended destinations because the routing information so quickly becomes stale. In contrast, geographic routing protocols don't actively maintain routes, and never need to determine or repair paths through the network.

Geographic routing protocols (or position-based routing protocols) forward packets based on the geographic position of the destination, not by identity or address. The nodes supporting a geographic routing protocol need only store minimal information about other nearby nodes, and never global topology or path information. There is no need to set up a path because hop-by-hop decisions are made about where the packet should be forwarded next. This allows the protocols to quickly adapt to changes in the network while avoiding stale information.

There are two main approaches to geographic routing: *greedy routing* and *face routing*. In greedy routing the packet is forwarded at each hop to whichever directly connected neighbor is nearest the packet's destination [Fin87]. Unfortunately, if the packet reaches a node whose neighbors are all further away from the destination than it is, the packet has reached a dead end (local minimum), and cannot make greedy progress.

Face Routing (Compass Routing II) is the other main geographic routing approach [KSU99]. Face routing makes use of the right-hand rule that states that the entire boundary of any face of a planar graph can traced by, at each vertex proceeding along the incident edge that is sequentially counterclockwise about the vertex from the arrival edge (aka at each vertex/roundabout always take the immediate right turn). Various face routing methods utilize the right-hand rule to traverse individual faces and then switch to the next face along a path of faces from the source to the destination.

Greedy and face routing are often combined producing so-called greedy-face-greedy protocols. These protocols start routing greedily until a local minimum is reached, at which point face routing is used to route around the face between the local minimum and the destination. Greedy routing then resumes.

Unfortunately most geographic greedy-face-greedy routing methods: (1) assume that a node is directly connected to all other nodes within some maximum range (unlike directional links), (2) assume that the topology does not change while a packet is being routed, and (3) assume that the topology graph is planar (has no edges that cross). The third assumption can be satisfied by making a minor adjustment to the topology management framework, that explicitly controls the topology. However, the first two assumptions are not realistic for a routing protocol in a large-scale airborne network where directional data links are used. What is needed is a geographic routing algorithm that can quickly respond to changes in the network, even while a packet is in flight, and can work with topologies formed by directional links. Such a routing algorithm could also benefit from the local topology information available at the nodes as a by-product of topology management.

## 1.5 Thesis Statement

*A topology management framework and a routing algorithm can be developed that are capable of connecting thousands of commercial aircraft, flying normal routes, into a large-scale airborne network sufficient to support tens of thousands of 1 Mbps data flows.*

## 1.6 Summary of Main Contributions

In support of this thesis, two algorithms are designed that are sufficient to connect thousands of airborne nodes into a topology which is actively managed over time and route large amounts of data between those airborne nodes. In addition, these algorithms are implemented and tested via simulation. The results of these simulations, where nodes follow actual aircraft trajectories, demonstrate the effectiveness of these algorithms and the capability of the airborne network. A summary of each of these contributions is given below.

### 1.6.1 Distributed Topology Management Framework

The first contribution of this work is a topology management framework and associated topology generation algorithms that can operate in a distributed manner on a large set of nodes. The topology management framework periodically runs on each node, using a topology generation algorithm to compute a local topology graph given only the positions of nearby nodes. If any local connections have changed since the last iteration, the framework makes the appropriate connection changes (re-pointing its links) and then waits for the next iteration. Although there are instances where nodes disagree, and connections do not form, a good topology generation algorithm can minimize the impact of this. There are numerous topology generation algorithm variants that could be used, but the best algorithms have low computation time, and compute topologies that are robust, inclusive, and contain short paths between pairs of nodes. The Degree Constrained Gabriel Graph (DC-GG) and Degree Constrained Delaunay Triangulation (DC-DT) perform best among several algorithms designed and tested.

### 1.6.2 Topology Aware Geographic Routing Algorithm

Routing within large mobile networks is challenging, especially at high data rates and when node movement is highly dynamic. The next contribution of this work is Topology Aware Geographic Routing (TAG), a position-based routing algorithm that strategically uses local topology information to make better local forwarding decisions, decreasing the number of hops required to deliver a packet when compared with other geographic routing protocols, while essentially guaranteeing delivery, and sending no overhead packets. In addition TAG is able to reliably deliver packets even in topologies that violate the often used but unrealistic *unit disk graph* and *quasi-static* assumptions. A variety of simulations support the claim that TAG outperforms GOAFR+, GFG, and OLSR in both theoretical environments and in a simulated, real-world, continental-scale airborne network.

## 1.7 Organization of Dissertation

The remainder of this dissertation is organized as follows.

Chapter 2 gives an in-depth look at several pertinent topic areas and provides an overview of related research that has been conducted.

Next, in Chapter 3, the new distributed topology management framework is detailed, and experiment results that compare various topology generation algorithms are presented.

Chapter 4 then details the design of a new topology aware geographic routing protocol that is able to route packets in the highly-dynamic airborne network topology while using local topology information (obtained as by-product of topology management) to improve its efficiency. Results of simulations comparing this new protocol to other state-of-the-art geographic routing protocols in an airborne network are also presented.

The results of a variety of simulations demonstrating the effectiveness of the resulting airborne network system are then presented in Chapter 5. These simulations utilize actual flight path data collected by the FAA to replay realistic node movements.

Finally, in Chapter 6, several areas of future work are described and the contributions of and main conclusions from this work are summarized.

# CHAPTER 2: BACKGROUND AND RELATED WORK

This chapter begins by introducing the reader to the burgeoning field of airborne networking, then briefly introduces various topics pertinent to understanding this dissertation. A discussion of works related to this research follows, where specific implementations of many of the topics covered in the background section are described.

## 2.1 Background

### 2.1.1 Airborne Networks

An airborne network has been loosely defined as an infrastructure that provides communication transport services through at least one node that is a platform capable of flight [USA07]. Until recently, most airborne networking research was conducted by military research facilities and defense contractors. Many of these military "airborne networks" included only a single airborne node, and simulated networks often included only a couple dozen nodes. The last few years, however, have seen a huge increase in both the scale of proposed airborne networks and the level of interest in airborne networking. This surge can be largely attributed to Google and Facebook who have allocated large amounts of time and money in an attempt to build the first large-scale civilian airborne network. The airborne networks being designed by these companies include not dozens, but thousands or even tens of thousands of nodes spread around the globe. The following sections provide more detail about these two main types of airborne networks, namely Military Airborne Networks and Civilian Airborne Networks.

#### 2.1.1.1 Military Airborne Networks

The United States Department of Defense (DoD) has pioneered Network Centric Operation (NCO), and Network-centric Warfare theory [DOD05a]. This theory advocates for a robustly networked military force, suggesting that such a network will lead to increased information sharing among troops, commanders and support personnel, better situational awareness, more collaboration, and faster decision making; ultimately yielding a decisive warfighting advantage. The notion is that future wars will be won, not necessarily by might, but by forces that can quickly and effectively disseminate information.

To achieve this vision, the Global Information Grid (GIG) [DOD05b] was proposed. The GIG is an all-encompassing communications plan, that includes everything needed to collect, process, manage, and disseminate information to warfighters, decision makers, and support personnel. The GIG includes a space backbone layer, with communication satellites, a terrestrial network layer, providing surface connectivity, and an airborne networking layer, connected to both space and surface networks. Airborne networks are thus an integral part of the grid and a fundamental component of the network-centric vision.

Military airborne networks can be further divided into two broad classes: airborne subnets, and backbone networks. Airborne subnets include several existing tactical and sensor networks. These various networks connect a small set of airborne platforms together to support a specific task. In contrast, backbone airborne networks form a high-bandwidth backbone Internet Protocol (IP) network to interconnect satellite networks, airborne subnets, sea and ground based mobile ad-hoc networks, and fixed terrestrial networks into a single network. Besides enabling communication between forces, these backbone networks are also a key to operating under satellite denied conditions, a current concern of the Pentagon.

#### 2.1.1.2 Civilian Airborne Networks

As mentioned previously, both Google and Facebook are progressing their own the massive civilian airborne networks that they hope will expand the reach of the Internet without the need of costly terrestrial infrastructure. These companies, however, were not the first inspired by the vision of providing high-speed internet from semi-permanent relay nodes in the stratosphere. Several companies have previously attempted to bring similar ideas to market.

Around 1998 SkyStation International began marketing what they called Stratospheric Telecommunications Service. They planned to sail large airships (blimps), that could stay aloft for a period of years, in the skies above large cities. These blimps would provide users on the ground with broadband service at what was then a high rate of 2 to 10 Mbps. At about the same time Raytheon and Angel Technologies Corporation similarly made an attempt to provide Internet access via manned high altitude aircraft circling over major cities [MC98] [MC00]. They promised consumers 24-7 access to video data and Internet at rates from 1 to 5 Mbps. The aircraft would act as a relay between a terrestrial gateway and a special “self-pointing” antenna mounted on a customer’s roof. Their testing resulted in the first successful commercial wireless broadband link between a moving aircraft and the ground. Both companies, however, failed to “take off”. Several factors are likely to blame for the failures including deficient communications and aircraft technology, the rapid expansion of terrestrial wired and wireless networks leading to a failure of their business models, and the impact of the terrorist attacks on September 11, 2001, on the aviation industry and the economy.

Over the next decade, similar projects continued to make small advances in delivering telecommunications from high-altitude airborne platforms, including projects in Japan, Korea, and Europe, and the United States. AeroVironment Inc., for example, worked with the National Aeronautics and Space Administration (NASA) in the Environmental Research Aircraft and Sensor Technology (ERAST) program to develop Helios, a high-altitude unmanned aerial vehicle (UAV) that was intended to perform atmospheric research tasks and serve as a communications platform. Unfortunately, the Helios Prototype broke up and fell into the ocean in 2003, and the entire NASA ERAST program was terminated [NBPD$^+$04].

A decade later (2013) Google Inc. announced its intent to form an airborne network of high-altitude balloons that would provide Internet service to users on the ground, including those living in remote areas [Sim16]. These balloons would float freely in the stratosphere, circling the globe, while forming a network among themselves. The balloons would provide communication services to every corner of the globe, including places with no current communications infrastructure. Project Loon, as it is called, has performed tests in New Zealand, Brazil, and Australia. Patent documents reveal that Google plans to create a hierarchy of balloons, with super-node balloons forming a backbone using free-space optical links, and sub-nodes connecting to the backbone and the ground antennas using radio frequency (RF) [DTBW13]. Another patent also reveals that Google anticipates clumping balloons over certain areas to meet expected historical demand, or to facilitate advance requests for service during special events [TP13]. Project Loon is currently battling for permits to run test flights over the United States. It is quite possible that Google will actually bring this seemingly “Loon-y” idea to fruition.

Not to be outdone, less than a year after Google’s announcement, Facebook announced its own ambitious Airborne Networking project. Rather than relying on balloons, Facebook is instead banking on high-altitude long-endurance solar-powered drones [Kel14]. These drones will circle high above commercial aircraft traffic and form high-capacity free-space optical connections with terrestrial gateways and other drones. In July 2016, Facebook conducted the first flight test of their full-scale prototype drone named Aquila [Guy16]. The drone is designed to eventually be able to remain aloft for up to six months at a time. With the wingspan rivaling that of a 747, Aquila flies while using only 5,000 watts of power, about as much as a high-end microwave. In addition to long-endurance drones, Facebook is also investing in Free-Space Optics (FSO) research. Its researchers recently reported significant progress in obtaining multi-gigabit per second rates using proprietary laser transceivers [Mag].

### 2.1.2 The National Airspace System (NAS)

This dissertation focuses on large-scale high-capacity airborne networks. One such network could be formed among commercial aircraft that are already flying in the skies every day. To simulate a realistic large-scale airborne network, that is large-scale both in numbers of nodes and geography, our simulations play back the actual flight paths of air traffic within the United States. This section gives an overview of the air traffic within the United States and how these data were obtained.

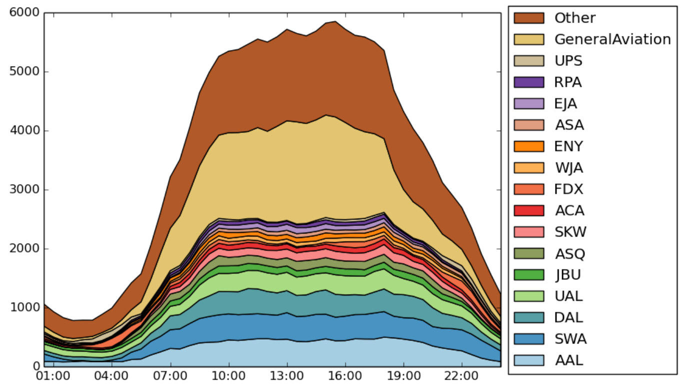


**Figure 2.1: A view of the number flights in the NAS over 24 hours, stratified by the company designation in the call sign. Time on the x-axis is UTC-6 (Central Time).**

The United States National Airspace System (NAS) includes the airspace, facilities, services, regulations, equipment, and personnel required to ensure safe air travel for an average of 50,000 flights that use the NAS each day. The NAS includes over 19,000 airports, 71,000 pieces of equipment, 14,000 air traffic controllers, and serves nearly 600,000 active certified pilots operating over 280,000 aircraft [FAA16]. The NAS is administered by the Federal Aviation Administration (FAA), which reports that on most days around 7,000 aircraft are constantly in the air over the United States. Figure 2.1 shows the number of flights in the NAS over 24 hours. The bands represent the portion of air traffic attributed to various companies and categories. Notice that for the given day the maximum number of flights in the air is just under 6,000. It is important to note that the data set these numbers are derived from does not include flights of military aircraft or flights of aircraft whose owners have requested to have their flight tracking information blocked. The second band from the top represents the general aviation flights (those using their aircraft registration number, also called the tail number, as their call sign). The share of flights by major United States air carriers can be seen in the bands starting at the bottom with American Airlines (AAL), followed by Southwest Airlines (SWA), then Delta Airlines (DAL), etc. Other airline abbreviations may not appear as familiar such as ENY, the symbol for Envoy Airlines. Envoy is a regional air carrier that operates under the American Eagle brand. The "other" band at the top combines all the remaining airlines together whose fraction of flights is too small to plot individually. This includes international airlines, small charter airlines, etc. One other interesting feature to note is the contribution of the cargo airlines. This can be seen on the left-hand side of the figure around 4 a.m. Notice the two bands that widen as the most others shrink. These are the hundreds of FedEx and United Parcel Service (UPS) flights that dominate the early morning air traffic, ensuring that packages are ready to be sent out on trucks at the crack of dawn. If a large percentage of the commercial aircraft flying in the early morning hours (including cargo aircraft) participate in the airborne network, most of the nodes will be able to remain connected to the network. However, it is unlikely that the sparse commercial air-traffic in these early-morning hours will be sufficient to provide reliable network connections to areas far removed from the busiest air-traffic routes. For this reason, it may be beneficial to also employ a set of drone aircraft to fill gaps in the network coverage during these times.

Figure 2.2 shows a similar view of the air traffic, but over the course of a week (July 1 to July 7, 2015). The diurnal variation (daily cycle) is quite evident, with much fewer flights during the early morning hours. Minor variations can be seen from day to day, but the general patterns are consistent. Notice also, the smaller number of flights on the July 4th weekend.

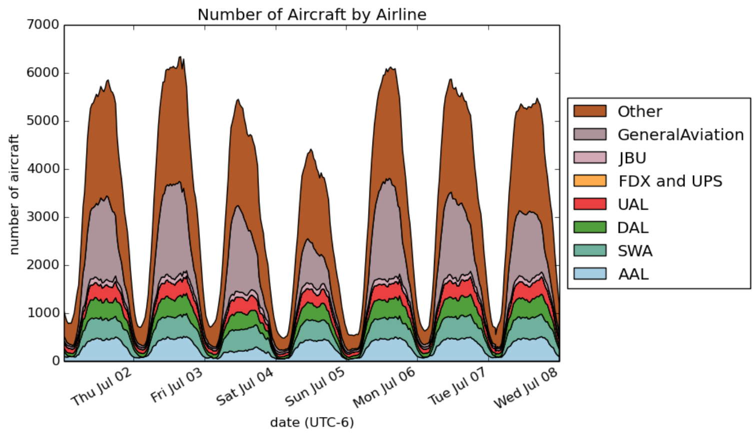


**Figure 2.2: A view of the number of flights in the NAS over 7 days, stratified by the company designation in the call sign.**

#### 2.1.2.1 ADS-B

An important part of the topology management scheme introduced in Chapter 3 is the exchange of position information with nearby nodes. Automatic Dependent Surveillance - Broadcast (ADS-B) essentially already performs this function for commercial aircraft, a fact we capitalize on.

ADS-B is a new Air Traffic Management and Control (ATM/ATC) surveillance system that has been certified as a viable replacement for the FAA's traditional radar-based systems. In the new system, each aircraft broadcasts a radio transmission containing its current position, ground speed, identification, and other air traffic management information approximately once a second [FAA05]. These messages are received by ATC ground stations and other ADS-B equipped aircraft within a 100 to 200 nautical mile range (185 to 370 km or 115 to 230 statute miles) [SLM15][FAA05].

The new system provides data faster and at a higher resolution than traditional radar systems, enabling closer spacing of aircraft and decreased congestion. It also has the added benefit of improving the situational awareness of pilots, who can now learn the positions of nearby aircraft and view the air traffic on an integrated display. Further, ADS-B decreases the reliance on a ground infrastructure, and ADS-B ground station receivers are easier and less expensive to deploy than radar stations. Because of these benefits, the FAA has mandated that most aircraft operating in the controlled United States airspace must be equipped to send ADS-B messages by January 1, 2020. [FAA10]

#### 2.1.2.2 Aircraft Situation Display to Industry (ASDI)

In order to replay aircraft trajectories for airborne network simulations, Federal Aviation Administration (FAA) Aircraft Situation Display to Industry (ASDI) data is used.

The FAA grants to some industry and research partners, access to 5 minute delayed aircraft position information for most aircraft in the United States National Airspace System (NAS). This data, called Aircraft Situation Display to Industry (ASDI) [Vol00], includes not only position, velocity, and aircraft identification information, but also the flight plan information for most aircraft. Thus, rather than relying on some mobility model for the movement of aircraft in the simulations in this work, each node instead follows the real path of an actual flight tracked by the FAA. The simulations conducted for this dissertation make use of 1 month of collected ASDI data from July 2015. This 63 Gigabyte data set includes over 160 million flight position reports, and well over 1 million separate aircraft trajectories.

### 2.1.3 Directional Communication Links

As mobile users consume higher and higher amounts of network bandwidth, cost effective alternatives to legacy wireless links are needed. Traditional ground-based wireless RF networks are bound by provable limits of per-node throughput [GK00], and will likely not scale to meet future demands. The directional communication links described in this section are key to enabling high-capacity airborne networks.

An antenna that radiates uniformly in all directions in a plane is called an *omnidirectional antenna*. Omnidirectional antennas are commonly used in radios, cell phones, wireless networks, etc. In contrast, an antenna that radiates or receives power more efficiently in some directions than in others is called a *directional antenna*. Directional antennas can both improve the transmission and reception of communications and reduce interference from extraneous sources. The improved transmission and reception allow for signals to be sent over longer distances, with fewer errors, or at higher rates. In addition, directional links provide increased security, significantly limiting the potential for communication detection and interception. This is a double-edged sword, however, as protocols that rely on overhearing their neighbor's transmissions (snooping) won't function well in a network of directional links.

The measurement used to describe the efficiency of an antenna is called the antenna's power gain or simply *gain*. *Transmit antenna gain* measures how well an antenna converts input power into radio waves, while *receive antenna gain* measures how well the antenna converts radio waves into electrical power. Most directional antennas exhibit maximum transmit and receive gain in a single direction. These antennas are called *unidirectional antennas*. Antennas that can change the direction of their focused transmission are called *steerable directional antennas*. These types of antennas can be either *mechanically steered* or *electronically steered*. Mechanically steered antennas are mounted on a gimbal or similar construction, that can turn the antenna, effectively rotating the antennas radiation pattern relative to the platform. Electronically steered antennas (also called steerable beam directional antennas, phased array antennas, smart beam-steering antennas, or smart antennas) can be steered to point in different directions by independently controlling the phases of each antenna in an array of antennas, such that constructive interference causes more power to radiate in certain directions and destructive interference attenuates the signal in other directions.

Directional communication links can be further divided based on which portion of the electromagnetic spectrum they operate in. Below, directional RF links that send signals using radio waves are first described. Then, new free-space-optical communication links, that communicate using pulses of light are introduced.

#### 2.1.3.1 Directional Radio Frequency (RF) Communication Links

Directional RF links are generally characterized as being extremely reliable, even in the presence of clouds, fog, or atmospheric turbulence. They can operate over long ranges (e.g. 200 km), and at fairly high rates (e.g. 274 Mbps). Armed forces from various countries use (mechanically and electronically) steerable directional radio frequency (RF) links [BRMD07] to create high-bandwidth connections among aircraft, and between aircraft and ground stations. The Mini-T2 system [L3:11], for example, can achieve data rates up to 274 Mbps at ranges up to 150 nautical miles (278 km, 173 statute miles). Also, DARPA is currently developing an RF link that can enable communication between an aircraft and ground station over long ranges at 100 Gb/s [Kel15]. Unfortunately, however, high-frequency RF links can suffer substantial attenuation from rain, depending on the rain drop size distribution and the rain field characteristics. Also, since radio frequency bands are regulated by governments, the operation of RF links requires a spectrum allocation. In the United States, the Federal Communications Commission (FCC) manages and regulates spectrum use. Because of the scarcity of available spectrum, it can be extremely difficult and costly to secure the large allocation needed for a new high-rate RF system.

#### 2.1.3.2 Free-Space Optical Communication Links

Free-Space Optical Communication (FSO) links use pulses of light emitted from an infrared laser to form a wireless optical communication channel (like fiber-optics without the fiber). The main advantage of these types of links is their high capacity. Data rates up to 1.28 Tbps (Terabits per second) have been demonstrated over a short distance of 200 meters [CAC$^+$09]. However, other advantages abound, including full duplex operation, low interference, no Federal Communications Commission (FCC) license requirement, no spectrum scarcity. Further, the extremely narrow beamwidth of the laser makes interception of transmissions by a third party very difficult, yielding a highly secure link. FSO links are also generally lighter, smaller, and require less power than RF links. Interestingly, since the propagation speed through fiber is slightly slower than in air, FSO links can actually send data faster than fiber-optics links.

Naturally, these links come with a set of limitations and new challenges that must be overcome. First, atmospheric turbulence can cause the intensity of the optical radiation traversing the atmosphere to fluctuate randomly. This is called scintillation [APH01] and is analogous to the fading experienced by RF systems. The optical links are also sensitive to atmospheric absorption and scattering. Further, the links require an unobstructed line-of-sight (LOS) between FSO links, and thus their optical signals are severely attenuated by fog, haze, and clouds. Finally, the narrowness of the laser beam requires extremely accurate pointing. FSO links attached to or tracking mobile platforms (such as aircraft) must be mounted on a precision gimbal, and employ a sophisticated Pointing Acquisition and Tracking System (PAT) [RED13] to acquire connections and to continuously adjust their pointing to maintain a lock on their target.

Despite these limitations, Free-Space Optical Communication links are seen by many to be an "essential enabler" of airborne networks [BRSS12]. Many of the limitations of these links disappear or are reduced at high altitudes, making FSO links a natural choice for connecting a high-capacity airborne network or even a space network. In fact, NASA is already using laser communications in space. The recently completed the Lunar Laser Communications Demonstration (LLCD) [BSM$^+$09] proved that it is possible to receive, through a transportable ground terminal, high-rate laser communications from a satellite in lunar orbit over a quarter million miles away.

#### 2.1.3.3 Hybrid Free-Space Optical Radio Frequency (Hybrid FSO/RF) Communication Links

There is a natural synergy between directional RF and FSO links that can be exploited by using them in tandem [DTS05]. RF links generally have a smaller capacity than FSO links but are generally more dependable. FSO links have a huge capacity, but are not as dependable, especially in the face of excessive atmospheric turbulence. By combining an FSO and an RF link into a Hybrid link the benefits of both technologies can be enjoyed. The FSO link can transmit large amounts of data when available, and the dependable RF channel can be used for critical control messages, acknowledgments, or for exchanging data or dropped packets if the FSO link is momentarily down. Adding to the synergy, the link types complement one another where weather effects are concerned. High-capacity directional RF links perform poorly in rain, but in the same conditions, FSO links experience only a moderate attenuation. Similarly, FSO link performance is severely impacted by fog or haze, but RF link performance is not. Interestingly, the reason fog impacts an FSO connection is that the radii of the water droplets in fog are about the same size as the infrared wavelength, causing the infrared light to scatter [GW02]. Similarly, RF links perform poorly in rain, because the sizes of the raindrops are close to the wavelengths of microwave signals.

The United States military is actively pursuing hybrid FSO/RF link technology. DARPA recently demonstrated a hybrid FSO/RF link capable of providing 10 Gbps of bandwidth over ranges greater than 200 km. Specific tests show an air-to-ground connection maintaining a data rate of nearly 9 Gbps at ranges from 90 km to 130 km, and an air-to-air connection maintaining a data rate of 6 Gbps at ranges from 50 to 212 km [YHSJ15][DOD13]. These ranges and rates are possible because of the use of cutting-edge adaptive optics (AO) and optical automatic gain control (OAGC) demonstrated by DARPAs Optical RF Communications Adjunct (ORCA) and Free-space Optical Experimental Network Experiment (FOENEX) projects [SPM$^+$11]. Adaptive optics corrects optical distortions caused by atmospheric turbulence, while OAGC helps to keep the incoming signal levels consistent. Performance is further improved using a link layer retransmission scheme, that detects the onset of scintillation events and quickly retransmits dropped packets over the RF link.

Many of the experiments conducted as a part of this research use simulated links that emulate the fundamental characteristics of these hybrid FSO/RF links, but generally in pristine environments.

### 2.1.4 Network Topology

A network topology indicates the arrangement of communication links between nodes in a network. For traditional wired networks, the network topology is determined by the physical cables connecting network nodes, and it is generally fairly static. In contrast, most wireless networks use omnidirectional antennas to transmit their messages in all directions to all nodes within some maximum range. If nodes are mobile, the topology is dynamic and is implicitly determined by the environment, the transmission ranges (affected by the transmission power), and the node movement. In *directional networks* (networks that utilize steerable directional communication links, such as FSO links) a topology must be explicitly determined and managed as nodes move. Nodes have a finite number of links, and therefore a finite number directly connected neighbors. Each directional antenna or free-space optical terminal for these links must be commanded to point at its intended neighbor, who also must point one of its links back. More importantly, each node must determine which of its potentially many neighbors it should directly connect with.

This following sections explore some concepts relating to the network topology graph and controlling the topology.

#### 2.1.4.1 Unit Disk Graph

The region of a plane bounded by a circle is called a *disk*. For theoretical analysis, the topology of a wireless network is often modeled as a *unit disk graph* (UDG), where an edge exists between any two nodes whose separation distance is less than 1 unit (the maximum range). A unit disk graph can be visualized using disks with a radius of one unit, in which case the graph would include edges between a node and any node within its disk. Alternatively, the graph can be visualized using disks with a radius of half a unit (0.5), in which case the graph would include edges between any nodes whose disks intersect. Figure 2.3 shows a simple unit disk graph for a set of nodes that uses the latter visualization convention (half unit disks). Notice that the graph includes an edge between any two nodes whose half unit disks intersect (e.g. nodes $x$ and $y$), and does not include an edge between nodes whose disks do not intersect (e.g. nodes $y$ and $z$). Formally a unit disk graph can be defined as follows. Given $P$, a set of points in a plane, the unit disk graph $U(P)$ includes a vertex for each point and an edge $(u,v)$ between a pair of vertices if and only if $dist(u,v) \leq 1$, where $dist(u,v)$ is the Euclidean separation distance between $u$ and $v$. Notice, that the unit disk graph places no constraint on the degree of each node. In a unit disk graph, a node is always connected to all other nodes within the maximum transmission range (that has been normalized to 1 unit), and is never connected to any node outside that range [KWZ03a]. The unit disk graph is an accurate model of a 2-dimensional broadcast wireless network if: (1) all nodes have the same transmission power and the same receive gain, yielding the same maximum transmission radius, (2) every node's transmission pattern is a perfect circle (each node transmits and receives equally in every direction), and (3) there are no radio-opaque obstacles and no multipathing to interfere with this perfect transmission. While convenient for theoretical analysis, unit disk graphs rarely match real-world wireless network topologies [KWZ03a], [KGKS05b], [KGKS05a]. In this research, the Unit Disk Graph will be utilized to represent all of the connections that could potentially be formed among a set of nodes. Since not all of these connections can actually be formed simultaneously, an appropriate degree-constrained subgraph of the unit disk graph must be determined.

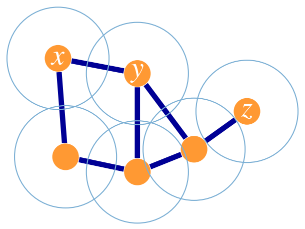


**Figure 2.3: Unit Disk Graph for the set of points given. Edges are included only for nodes whose half unit disks intersect.**

#### 2.1.4.2 Planar Graph

Some techniques (e.g. face routing) used in this research require graphs that are guaranteed to be planar. A *planar graph* is a graph that can be drawn in a plane without any of its edges crossing. If the positions of vertices of a non-planar graph are fixed in a 2-dimensional plane, a planar subgraph can be generated from the original graph by removing an edge for each pair of edges that intersect. There are many algorithms for generating a planar subgraph. One such algorithm is the Gabriel Graph (GG). Remember that a disk is the region of a plane bounded by a circle. A *closed disk* includes points on the circle (includes all points $\leq$ the radius). Let $disk(u,v)$ be a closed disk that has the line segment $uv$ as a diameter. Given a set of points $P$, the Gabriel Graph includes an edge $(u,v)$ if and only if $disk(u,v)$ contains no other points in $P$. Figure 2.4 displays the edge $(u,v)$, for example, since node $w$ (and all other nodes in $P$) are located outside of the disk of which the line segment between $u$ and $v$ is a diameter. If $w$ were located inside the disk, the edge $(u,v)$ would not be included in the Gabriel Graph.

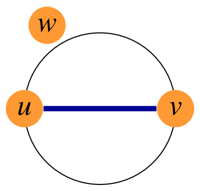


**Figure 2.4: A Gabriel Graph Example: $u$ and $v$ are connected since no other vertex is within the disk.**

Delaunay Triangulation is another algorithm that can be used to generate a planar subgraph of an input graph. Delaunay Triangulation creates a triangulation for an input set of points. A circumcircle is a circle that passes through all the vertices of a given polygon. Given a set of points $P$, the Delaunay Triangulation includes a triangle $tuv$ if and only if the circumcircle of that triangle contains no other points in $P$. Figure 2.5 shows that the resulting triangulation would include triangle $tuv$, since $w$ (and all other nodes in $P$) are located outside of the circumcircle. The Gabriel Graph is a subgraph of Delaunay Triangulation.

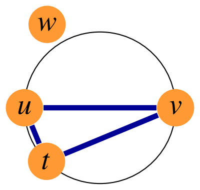


**Figure 2.5: A Delaunay Triangulation Example: the triangle $tuv$ is included since no other vertex is within the disk.**

A planar graph consists of a set of regions, called faces, bounded by edges. There may be many interior faces, that make up a graph, but always also one extra unbounded exterior face that encompasses all other space.

#### 2.1.4.3 Topology Control

Topology Control in wireless ad hoc network and sensor networks has been studied extensively. However, the main focus has been on networks using standard omnidirectional antennas. In these networks, topology control involves adjusting the transmission power of nodes to achieve a global topology that is connected and optimized for some metric, such as minimal energy use. In [San05] Santi gives a thorough overview of Topology Control research, dividing the field into Homogeneous and Nonhomogeneous methods, where the homogeneity refers to the transmit power or range of the nodes. In homogeneous topology control, the Critical Transmitting Range (CTR) is the common minimum range at which all nodes will transmit, such that the network remains connected. Nonhomogeneous methods allow for each node to be assigned a unique transmit range and associated transmission power, enabling even more specific control of the topology. Santi further divides nonhomogeneous methods into Location-Based, Direction-based, and Neighbor-based distributed protocols. In Chapter 3 a distributed topology control method is introduced that explicitly controls the topology of a network of mobile nodes connected using directional links.

### 2.1.5 Routing in Mobile Ad hoc Networks

Delivering messages between nodes in an ad hoc network is an important and difficult problem. In the routing problem, a source node $s$ and a destination or target node $t$ are elements of a set of points $P$, and the node $s$ is to send a message to the node $t$.

Mobile ad-hoc networks typically employ nodes that act as both routers and end-users. Nodes that are not within mutual transmission range of one another must communicate by relaying their messages through a series of intermediate nodes, each acting as a router. A routing protocol is needed to determine at each node in the series, where the message should be forwarded next. Traditional routing protocols (such as link state or distance vector based protocols) are not well suited for routing in networks where the topology changes frequently [EM04]. Mobile Ad-hoc network (MANET) routing protocols are designed to support networks of mobile nodes, but even these protocols can fail to efficiently route packets in highly-dynamic, large-scale networks [NAJ15]. Mobile ad hoc routing protocols can be partitioned into two main classes: topology-based protocols and position-based (geographic) protocols.

#### 2.1.5.1 Topology-Based Routing

Topology-based routing protocols use link information that exists in the network to make forwarding decisions. Topology-based protocols for mobile ad hoc networks can be further divided into three groups: (1) Proactive protocols, (2) Reactive (On Demand) protocols, and (3) Hybrid protocols.

Proactive MANET routing protocols use traditional routing algorithms to maintain paths (routes) between every pair of nodes in a network. Routes are proactively maintained between every pair of nodes, ensuring that packets flowing over a previously unused path can begin flowing without delay. Unfortunately, proactive protocols require that numerous control messages be sent to update the routing tables. These control messages, or overhead messages, may occupy an unacceptably large fraction of the network bandwidth, actively maintaining paths between nodes that never directly communicate. This overhead for proactive protocols, however, generally remains fairly constant as node mobility increases. Optimized Link State Routing Protocol (OLSR) [JMC$^+$01] and Destination-Sequenced Distance Vector (DSDV) [PB94] are examples of proactive MANET routing protocols. OLSR is a version of the classical link state algorithm that has been optimized for mobile networks. In OLSR each node selects Multipoint relays (MPRs) and uses only these nodes to broadcast control traffic to the network, significantly reducing the number of transmissions when compared with traditional flooding. In addition, link state information is only generated by MPRs, minimizing the number of OLSR control messages. DSDV, on the other hand, is based on the distance vector algorithm but it adds sequence numbers to each table entry to avoid routing loops as the network topology changes.

In an effort to improve efficiency, reactive MANET routing protocols only manage network paths that are currently in use. However, when a reactive protocol receives a packet destined for a node to which no path exists, the protocol must first perform *route discovery* to find a path through the network to the new destination. Only after a full end-to-end path to the destination is determined, can packets begin to flow. This leads to a potentially long delay for the first packet transmitted. Further, when the topology changes after the initial route discovery, a route must be repaired. Until the new path is discovered, packets must be buffered or dropped. To establish or repair a path through the network reactive protocols send overhead packets in the form of route request, route reply, and route error messages. Even though routes are only maintained for connections currently in use, a huge number of overhead packets may still be needed to handle frequent or sizable topology changes. Examples of reactive routing protocols include Ad-hoc On-demand Distance

Vector routing (AODV) [PR99] and Dynamic Source Routing (DSR) [JM96]. AODV determines a route to a new destination by broadcasting a route request (RREQ) throughout the network. When the request reaches the destination or a node with a route to the destination, it sends a route reply (RREP) back to the originator. As the RREP makes its way back to the originator, intermediate nodes update their routing tables to accommodate forwarding packets along the path. Once the originator receives the RREP it may begin sending packets to the destination. If at some point AODV detects a broken link, route error (RERR) messages are sent to the impacted nodes, and nodes can then reestablish a route by sending a new RREQ. Similarly, DSR uses route request and route reply messages to find a route to a destination, however, instead of storing route entries at the intermediate nodes along a path, DSR includes the full routing path with each data packet sent. In addition to the standard network layer (IP) header, transport layer (TCP/UDP) header and link layer headers, DSR adds its own header that includes the full routing path to the packet's destination. Intermediate nodes need only look at the path in each packet's header to determine where to forward the packet. Unfortunately, the scalability of DSR is limited, and long paths would quickly consume all the space in a packet, leaving no room for data. In Chapter 4 a new routing protocol is detailed that also uses an extra header in addition to the standard transport and network headers. Unlike DSR's header, however, the size of this header does not increase with the routing path length.

Hybrid ad hoc routing protocols seek to obtain the best of both worlds, by using proactive routing locally, and reactive routing globally. Packets destined for nearby nodes or nodes within the same "group" are routed using a proactive approach, while packets heading for distant nodes are routed with a reactive strategy. These complex protocols still suffer from large amounts of overhead when the topology changes frequently.

To better understand how the scale of a network can affect these protocols, consider a network of ten thousand mobile nodes, with two nodes on opposite edges of this network trying to communicate. A reactive protocol will need to buffer packets at the sender until it can find a path through the network to the distant node. Once a path is found packets may begin flowing. However, unless the nodes along the path are essentially stationary, there is a high probability the path will quickly become invalid. All it takes is one node along the path to leave the network or move out of range of an adjacent node supporting the path, to make the path obsolete. Packets must then be delayed or dropped while the path is repaired. Extra management packets will also likely need to be sent, incurring overhead.

Using, instead, a proactive routing protocol only exacerbates the situation, generating what has been described as “torrents” of link status change messages in an effort to recover from any failure on any path [KK00]. The overhead associated with these messages can quickly saturate the available network bandwidth for large networks with a frequently changing topology. Also, despite the high cost, only a fraction of the packets may actually arrive at their intended destinations because the routing information can so quickly become stale.

### 2.1.6 Position-Based Protocols and Geographic Routing

Geographic routing protocols [CCSM13] may be a solution to routing in large-scale highly-dynamic mobile networks. They forward packets based on the geographic position of the destination, not by identity or address. The nodes supporting a geographic routing protocol need only store minimal information about other nearby nodes, and never global topology or path information. There is no need to set up a path because hop-by-hop decisions are made about where the packet should be forwarded next. This allows a geographic routing protocol to quickly adapt to changes in the network while avoiding stale information.

Each node in a network using a geographic routing protocol is assumed to be aware of its own physical location and the positions of nearby nodes. Further, geographic routing protocols require that a sending node knows the position of a packet’s intended destination. This information is often obtained using a separate location service protocol [LJDC$^+$00]. As described previously, in our example airborne network application, the positions of nearby nodes are fortuitously already being exchanged via ADS-B [FAA10]. We call nodes that are within transmission range of a node its *neighbors*, and nodes that are within position exchange (ADS-B) range its *community members*. To avoid confusion in directional networks, a neighbor to which a connection currently exists is called a *directly connected neighbor*.

#### 2.1.6.1 Localized Routing Protocols

Some geographic routing protocols are localized. Localized routing protocols make decisions based only upon local information, which is information that already exists at the node or can be “efficiently” obtained by the node generally by communicating with nodes within one or two hops. A routing protocol is classified as localized if each node forwards packets based solely on: (1) a small amount of information stored in a packet’s header (i.e. source position, destination position, etc), and (2) local information (i.e. positions of neighbors). These restrictions imply that in general no node has knowledge of the full topology, and unlike source routing protocols, a full path cannot be included in the small packet header. Protocols may further be classified as memory-less or stateless if a packet is not allowed to change or store information at a node. This restriction ensures that nodes need not store per-packet state information. The result is that if a packet arrives at a node that it has previously visited, it won't know it has already visited that node unless it has stored the node's identifier in its packet header (the size of which is limited).

There are many similarities between this type of routing and maze solving. The edges of the graph can be imagined as corridors of a maze, and the vertices as small rooms with a doorway leading to a corridor for each of the potentially many adjacent edges. Figure 2.6 shows such a maze for a network topology, and the equivalent topology graph. Efficiently traveling through such a maze from a starting room to an ending room is essentially the same task as routing a packet from the source node to the destination node. For geographic routing protocols, those traveling through the analogous maze can be imagined to have a limited Global Positioning System (GPS) receiver. They essentially know their current position, the maze end position, and perhaps the positions of the rooms reached by each of the corridors leaving a room. Unlike many maze solving techniques, however, no breadcrumbs can be left along the way. The algorithms are "ecologically friendly", and there is no way to mark a certain room or corridor as visited. Also, the imaginary GPS is limited in that it may only store a finite number of previously visited rooms. It cannot store an entire path or necessarily detect that a user is "going in circles".

#### 2.1.6.2 Greedy Routing

In greedy routing (also called greedy forwarding and before that also known as Cartesian routing) each node a packet visits attempts to greedily forward the packet so that it makes maximum progress toward its destination [Fin87]. At each hop the packet is forwarded to whichever directly connected neighbor is nearest the packet's destination. Figure 2.7 shows an example of a packet being greedily routed from a source node $s$ to a destination or target node $t$. The dashed circle centered at $t$ shows that $x$ is the node nearest $t$ of all the nodes directly connected to $s$. Therefore $s$ greedily routes the packet to $x$. Figure 2.8 displays the remaining greedy hops for this example. Notice that upon receiving the packet node $x$ greedily forwards the packet to node $y$ because $y$ is the directly connected neighbor closest to $t$. The packet then continues to make greedy hops until reaching its destination $t$.

Greedy routing is generally efficient, but unfortunately, it alone cannot guarantee delivery in arbitrary networks. Consider, for example, the same network topology used in the previous examples, but with a different source node $s$, as shown in Figure 2.9. In this case, $s$ forwards the packet to $z$, but $z$ has no directly

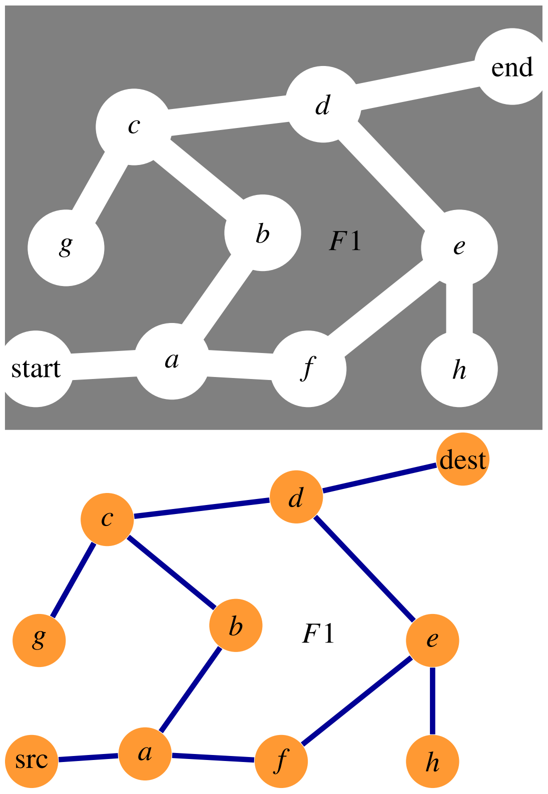


**Figure 2.6: A graphic showing how a network topology can be visualized as a maze.**

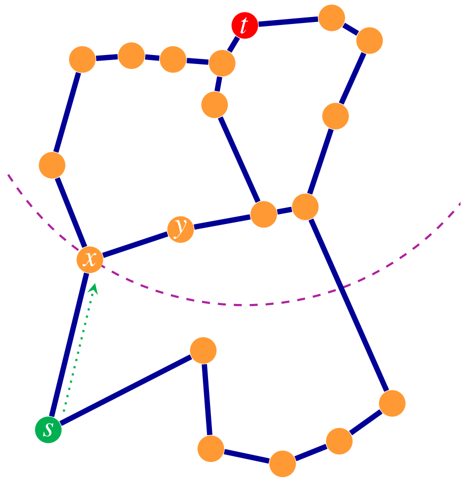


**Figure 2.7: Greedy routing example: $x$ is the node nearest the destination $t$ of all the nodes directly connected to $s$, therefore $s$ forwards the packet to $x$.**

connected neighbor closer to the destination $t$ than itself. When a packet reaches a node whose neighbors are all further away from the destination than it is, the packet has reached a dead end, also known as a local minimum, and cannot make greedy progress. If local minima exist in a topology, greedy routing must either drop the packet or employ some other method to assist in backtracking and routing around the void (or hole)

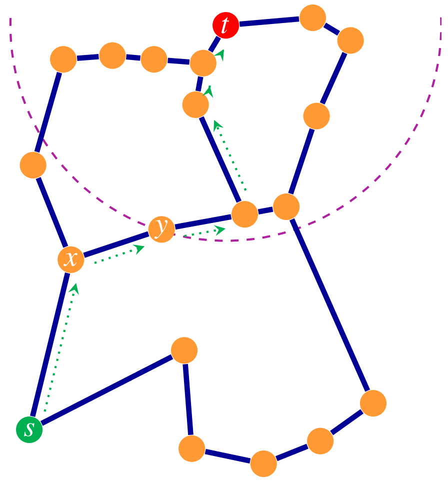


**Figure 2.8: A Greedy Routing example: $y$ is the nearest to $t$ of $x$'s directly connected neighbors therefore $x$ forwards the packet to $y$, and the packet continues to be greedily routed until reaching $t$.**

in the topology.

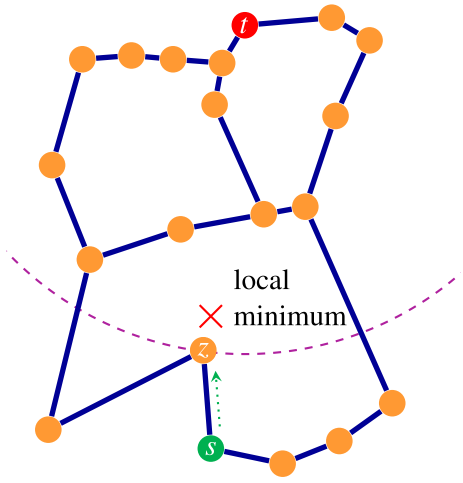


**Figure 2.9: A Greedy Routing local minimum example: $z$ is the closer than any directly connected node to $t$, therefore the packet cannot make greedy progress in the next hop, and greedy routing fails.**

#### 2.1.6.3 Face Routing

Face routing is another class of geographic routing protocols. In face routing a packet explores the boundaries of a sequence of faces between the source and destination. Crossing edges in a topology graph confuse the traversal of face boundaries. To ensure that face boundaries can be explored, face routing protocols require a topology graph to be planar. As described in Section 2.1.4.2, a planar graph is a 2-dimensional graph where no two edges cross, or in other words, where all edge intersections occur at vertices. Planar graphs partition the plane into a set of faces, regions bounded by the edges of the graph. A series of faces separate a source node from a destination node. Face routing traverses these faces, exploring the boundary of each, and eventually finds the destination on the boundary of the last face. While exploring a face boundary, face routing protocols identify the next face to traverse. The face routing protocol then performs a face change; what was the next face becomes the current face, and the procedure repeats, exploring each new face until the destination is reached. A maze-solving rule says that any 2-dimensional maze can always be exited (and solved assuming the finish/goal is on the boundary of the maze) if upon entering the maze, one places her right hand on the right wall of the entrance corridor, and always when confronted by an intersection, turns right, always keeping the right hand on the wall. [O'B89]. This right-hand rule can also assist in exploring the boundary of a face in a planar graph. Refer back to the maze analogy of our topology graph, and assume that the immediate goal is not to reach the maze end, but rather to travel around a face in the matching planar graph. If one places a right hand on the right wall when entering a corridor and always takes immediate right turns, keeping the right hand on the wall, eventually they will always arrive back where they began, having traced the boundary of a face in the equivalent planar graph. In Figure 2.6, for example, if one were at node $a$ and entering the corridor toward $b$, and if they placed their right hand on the right wall and walked always keeping it there, they would visit nodes $b$, $c$, $d$, $e$, $f$, and finally arrive back at node $a$. The path traces the boundary of face $F1$. The boundary of any face in a static planar graph can be traced in this manner. Alternatively, the left hand could instead be used. and immediate left turns taken, allowing a face boundary to be traced in a counterclockwise direction. Various face routing methods utilize the right-hand and/or left-hand rules to traverse the faces of a planar graph and arrive at the destination. The general concept is to have a packet travel around one face until some face change condition is met. The algorithm then changes faces, and repeats the process, exploring the boundary of the new face. This continues until the destination is found on the boundary of the last face explored. Various face change conditions exist, and each must guarantee that progress is made toward the destination. If a face boundary is being explored starting from a certain starting point, one basic face change condition is to change faces once a boundary edge is reached that intersects the line segment from the starting point to the destination at a point closer to the destination than the starting point. Another face change condition is to change faces at the node on the boundary nearest the destination. If this face change condition is used, the topology graph must also be a unit disk graph. If the topology graph is static, planar, and node locations are accurate, face routing guarantees that one of two things will eventually happen: (1) the destination is encountered while exploring a face boundary, in which case the packet can be delivered, or (2) an entire face boundary is explored and the face change condition is never met. In the second case, the destination is always unreachable given the assumptions above. However, the above assumptions are not always easy to guarantee [FS10] as mobile networks almost never naturally have a static or planar topology, and techniques used to planarize topology graphs often encounter errors in real-world networks. Face routing has two main disadvantages. First, the requirement that the graph be planar complicates the topology formation, and also generally leads to a less-optimal topology, since non-planar edges must be removed. Second, when exploring the boundary of a face, face routing can proceed in a clockwise or counterclockwise direction. A poor choice of direction could lead to a much longer path.

#### 2.1.6.4 Greedy-Face-Greedy Routing

Greedy and face routing are often combined producing so-called hybrid greedy-face routing protocols, or greedy-face-greedy protocols. These protocols start routing greedily until a local minimum is reached, at which point face routing is used to route around the face between the local minimum and the destination. Once some condition is met, greedy routing takes back over, efficiently routing the packet until another dead end is reached. Figure 2.10 shows an example of Greedy-Face-Greedy Routing. The packet is greedily routed from node $s$ to $z$, which is a local minimum. The packet then changes to face routing mode, in an attempt to route around the void between $z$ and $t$. The right-hand rule is used to enable the packet to travel around the boundary of the face in a clock-wise manner. Upon reaching $x$ the packet has made progress toward $t$ and is now closer to $t$ than when it started face routing. The packet reverts to greedy mode, and is greedily forwarded until it reaches $t$. Many geographic routing protocols employ this greedy-face-greedy cycle but in a variety of different ways [KK00], [KWZ03b], [BMSU99]. Several of these protocols are described in Section 2.2.2.3.

## 2.2 Related Works

### 2.2.1 Topology Control

As described in Section 2.1.4.3, many different Topology Control algorithms exist. In this section, a few of the most applicable algorithms to this dissertation are described.

Local Minimum Spanning Tree (LMST) [LHS03] is a topology control algorithm that builds a Minimum Spanning Tree-like (MST-like) topology in a fully distributed manner. The algorithm requires that

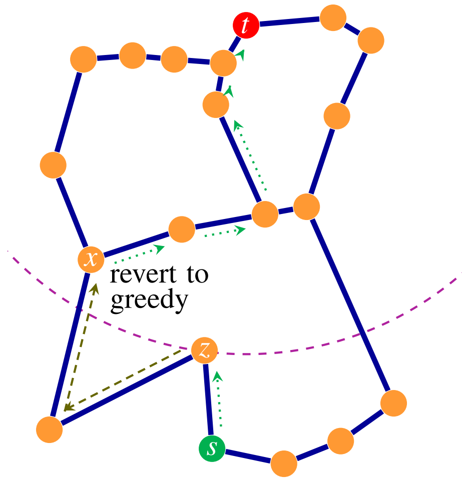


**Figure 2.10: A Greedy-Face-Greedy routing example.**

each node must know the positions of nodes within transmission range. This position information is called *local information* since it can be efficiently obtained by each node by simply periodically exchanging positions information with neighboring nodes. Each node then, using only this local position information, independently builds a local MST, spanning all nodes in its neighborhood. Each node then adjusts its transmission power such that only those nodes to which it is directly connected in the resulting graph are within transmission range. Similarly, for networks using directional links, each node adjusts where its links are pointing such that it is connected to only those nodes to which it is directly connected in the resulting graph. The outcome is a topology graph that has a low nodal degree, short link lengths, and is connected (unless physically impossible given the maximum range). Communication graphs with small nodal degrees are beneficial because they minimize interference, allow efficient bandwidth utilization, and reduce access contention in the network. LMST guarantees a nodal degree that is less than six. The link length is the distance between a transmitting node and its directly connected neighbor. Short link lengths are desirable to maintain low levels of packet loss. Finally, a connected network is necessary to enable any pair of nodes in the network to communicate.

Since LMST generates a global topology that is MST-like, the result is a topology that is not very robust as there is generally only a single path between any two nodes in the topology. When a link goes down there is often no alternate path in place, and thus the reliability of the network is decreased. Local Tree-based Reliable Topology (LTRT) [MNAK09] uses LMST as a subroutine to generate a more "reliable" topology. LTRT and its predecessor, TRT [ACK04] (that requires global information), are able to maintain a k-connected topology, where at least k paths exist between every node in the network. LTRT has been shown to perform well even in large dense networks [TNAK10] and performs better than several other similar protocols in terms of average nodal degree and average link length. However, LTRT does not allow the degree of nodes to be arbitrarily limited, which is desirable for our application since a node may bear fewer than six directional links.

Neither LMST or LTRT are designed for use in networks connected with directional antennas. Many of the standard topology control protocols cater to sensor networks, where power efficiency is a prominent concern, and where accurate position information may be unavailable. Some Topology control research has been conducted using directional antennas, but often sectorized directional antennas are assumed, as in [HSSJ02], where a Cone-Based Topology Control (CBTC) method is detailed for use by switched directional antennas. The basic idea of CBTC is to divide up a circular area centered at a node into equal slices called cones. Each node then selects a transmission power such that every cone around the node contains at least one neighbor reachable at that power.

### 2.2.2 Geographic (Position-Based) Routing

A large number of geographic routing protocols exist, each with their own strengths and weaknesses. The algorithms can be split into three classes: Greedy Forwarding Methods, Face Perimeter Forwarding Methods, and Hybrid Methods. Note that herein $\overline{xy}$ represents the line between points $x$ and $y$, $|\overline{xy}|$ represents the distance between points $x$ and $y$, $\vec{y}$ represents the vector extending from the origin to the point $y$, $\measuredangle xyz$ represents the measure of $\angle xyz$, and $\angle xyz$ is the angle at vertex $y$ created by the two rays that extend from $y$ through points $x$ and $z$.

#### 2.2.2.1 Greedy Forwarding Methods

Greedy forwarding methods attempt to advance a packet toward the destination by making a greedy choice at each hop. Several different greedy forwarding variants exist, depending on the metric used to determine the greedy choice. A packet could be forwarded, for example, to the neighbor that is nearest the destination (the one that maximizes the advance toward the destination). Instead, however, it could be forwarded to the neighbor that minimizes the angular separation between two rays; one extending from the node toward the final destination and the other extending toward the neighbor. These and other variants are described below. Some of the variants ensure loop-free operation, while others cannot.

**2.2.2.1.1 Greedy Routing (GR) (Closest Distance, maximum advance)** The most commonly used forwarding method is the basic greedy routing method [Fin87], where a packet is forwarded to whichever neighbor is nearest the destination. If a packet is destined for a node at point $t$ and is at a node with two directly connected neighbors at points $a$ and $b$, the packet would be forward next to the node at point $a$, if $|\overline{at}| < |\overline{bt}|$. Minimizing the distance to the destination is equivalent to maximizing the gain toward the destination or advance (where advance is calculated as $|\overline{pt}| - |\overline{at}|$). If a packet arrives at a node that is the closer to the destination than any of its directly connected neighbors, the packet has reached a dead end (local minimum) and must be dropped. Greedy Routing is loop free, but cannot guarantee delivery in arbitrary networks. In the example shown in Figure 2.11, $p$ is deciding to which of its directly connected neighbors ($c$, $g$, and $f$) a packet, destined for $t$, should be forwarded. When greedy forwarding using this variant, the packet would be forwarded to node $g$ because it is the nearest of all $p$'s neighbors to $t$. Choosing $g$ yields the maximum advance toward the destination.

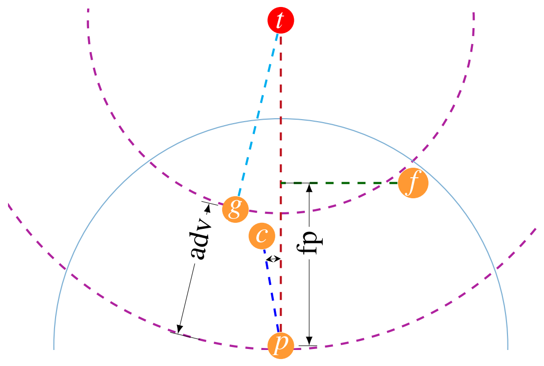


**Figure 2.11: An example showing how a packet at $p$ may be forwarded to various different neighbors depending on the variant of greedy forwarding used.**

**2.2.2.1.2 Compass Routing (DIR) (Closest Angle)** In compass routing [KSU99] the direction of the destination, relative to the current node, is determined. Then, the packet is forwarded to whichever neighbor has the relative direction nearest that of the destination. More specifically, a packet at a node at point $p$ destined for a node at point $t$ with two directly connected neighbors at points $a$ and $b$ would forward the packet to the node at point $a$ if $\measuredangle tpa < \measuredangle tpb$. A packet being forwarded from node $p$ would next visit node $c$ in the example shown in Figure 2.11 if Compass Routing were used. This is because the ray toward node $c$ from $p$ has a smaller angle relative to the ray from node $p$ toward the destination than the rays from $p$ toward any of the other neighbors. It should be noted, however, that compass routing does not guarantee loop-free operation and some complex graphs have been shown to contain loops.

**2.2.2.1.3 Most Forward within Radius (MFR) (Maximum Forward Progress)** In the Most Forward within Radius (MFR) method [MM96] a packet is forwarded to whichever node maximizes the forward progress of the packet. Forward progress differs from advance, in that it is measured along a line between the current node and the destination, where locations of nodes have been projected. A packet at a node at point $p$, destined for a node at point $t$, with two directly connected neighbors at points $a$ and $b$ would forward the packet to $a$ if $(\vec{a}-\vec{p})\cdot\frac{\vec{t}-\vec{p}}{|\vec{t}-\vec{p}|} < (\vec{b}-\vec{p})\cdot\frac{\vec{t}-\vec{p}}{|\vec{t}-\vec{p}|}$ (the projection of the vector from $p$ to $a$ onto the vector from $p$ to $t$ is less than the projection of the vector from $p$ to $b$ onto the same vector from $p$ to $t$). In Figure 2.11, a packet at $p$ would be forwarded to $f$ using this greedy forwarding variant since its progress along the line segment between $p$ and $t$ is greater than any other neighbor. Node $g$ is actually closer to the destination than $f$, but $f$ makes more progress along the line towards the destination. Like Greedy Routing, MFR is loop-free, but not as commonly used.

#### 2.2.2.2 Face Perimeter Forwarding Methods

Greedy forwarding methods cannot guarantee the delivery of a packet, but in some cases, face perimeter forwarding methods can. These methods seek to deliver a packet in a planar topology graph or subgraph by advancing across (traversing) a set of faces, forwarding the packet around the perimeter of each face. This section describes some of the many face routing variants.

**2.2.2.2.1 Face Routing (FR) (FACE-1)** The first and most basic Face Routing algorithm was initially published under the name Compass Routing II in a seminal work by Kranakis, Singh, and Urrutia [KSU99]. In this algorithm, to route a packet from $s$ to $t$, the face $F_0$ incident to $s$ and intersected by the line $\overline{st}$ would first be identified. The packet would then begin exploring the boundary of $F_0$ using the right-hand rule. As the boundary is explored, the point $p_0$ where $\overline{st}$ intersects an edge of the face is identified. For concave faces, this can happen multiple times, and the intersection point that minimizes the distance to $t$ is recorded in the packet. Upon returning to $s$, having traced the entire boundary of $F_0$, the packet again traces the face boundary until the packet is about to traverse the edge on which $p_0$ lies. At this point, $p_0$ takes the place of $s$ and we repeat the process, finding and exploring the boundary of $F_1$ incident to $p_0$ and intersected by $\overline{st}$. The packet continues traversing faces and advancing toward the destination until one of two events occur:

(1) the destination $t$ is encountered while exploring the boundary of the last face, in which case the packet is delivered, or (2) the packet explores the entire boundary of a face without finding an intersection point closer to the destination, indicating that the destination is unreachable, in which case the packet is dropped.

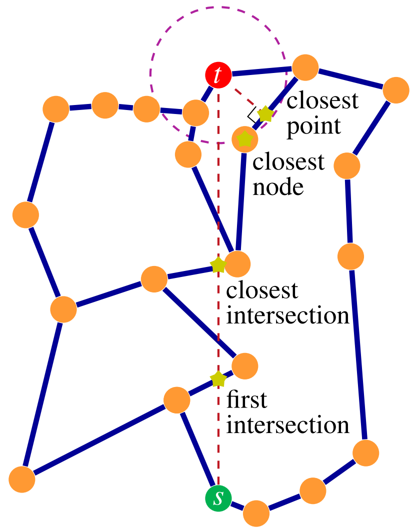


**Figure 2.12: An image comparing the various face change point options.**

Each Face routing technique employs a rule to determine at what point a packet should change faces. This basic face routing method uses the *closest intersection* rule. The intersection point of $\overline{st}$ with the boundary of the current face that is closest to $t$ is selected as the face change point. Figure 2.12 shows this and a variety of other face change point rules.

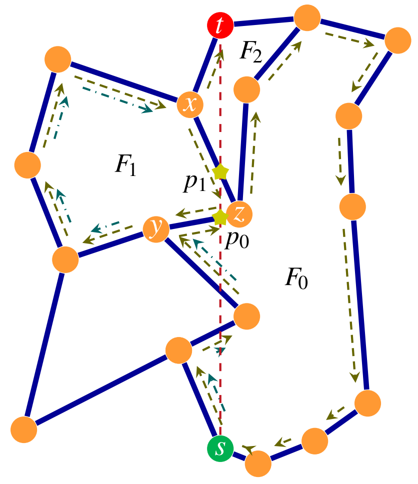


**Figure 2.13: A basic Face Routing example.**

Figure 2.13 shows an example of using this face routing method to forward a packet from node $s$ to node $t$ in a topology connecting several nodes. The packet begins at node $s$ and explores the entire boundary of face $F_0$, recording in the packet header the point, nearest $t$, where the line between $s$ and $t$ intersects the face boundary. In this case, the intersection point is $p_0$ on the line segment between nodes $y$ and $z$. Once this intersection point is found, however, the entire boundary must still be explored in case an even closer intersection point exists. Upon returning to node $s$ the packet is then forwarded until it reaches a node ($y$) attached to the edge containing $p_0$. At this point the packet transitions to tracing the face $F_1$, the face with $p_0$ on its boundary that intersects the line segment $\overline{p_0t}$. The procedure is repeated, exploring the entire boundary of $F_1$ and identifying $p_1$ as the intersection point. The packet is then forwarded around the boundary to node $x$, where it begins exploring face $F_2$. The destination is discovered on the boundary of this face and the packet is successfully delivered.

An optimization can be employed in this and similar face routing methods. The packet header may store information as it travels around the face about the number of hops before and after the chosen intersection point. When forwarding around the boundary back to the intersection point, an informed decision may then be made about whether it would be better to forward the packet in a counter-clockwise or clockwise direction. This optimization, for example, would have reduced the number of hops needed to forward the packet back to $p_1$ on face $F_1$ from four hops to a single hop.

**2.2.2.2.2 FACE-2** Bose et al. [BMSU99] propose FACE-2, a more efficient form of face routing where a face $F$ is traversed beginning at a point $p$ until reaching an edge $(u,v)$ that intersects the line $\overline{pt}$ at some point $p' \neq p$. Figure 2.14 shows an example of this face routing method. A packet begins forwarding around the boundary of face $F_0$ and upon reaching node $g$ determines that its next hop along the boundary would intersect $\overline{st}$ at $p_0$ (the first intersection point on the boundary of $F_0$). The packet then changes faces and begins exploring the boundary of $F_1$, until reaching node $y$ and changing faces back to $F_0$. $F_0$ is briefly explored finding $p_2$ and the packet then changes faces to $F_2$. The boundary of this face is explored until reaching node $x$, and once again the packet switches faces to $F_3$ and then delivers the packet to $t$ as it explores the boundary of this final face. The face change point rule used for FACE-2 is the *first intersection* rule, as depicted in Figure 2.12. This variant of face routing is important to the GFG method described in Section 2.2.2.3.1.

**2.2.2.2.3 Bounded Face Routing (BFR)** In [KWZ02] Kuhn et al. began their quest for a geographic routing protocol that could be analytically analyzed and proven to be asymptotically optimal and average-

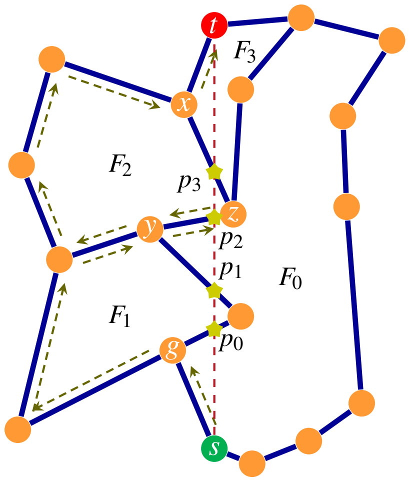


**Figure 2.14: A FACE-2 routing example.**

case efficient. Their initial algorithm builds upon Face Routing but attempts to overcome one of the main disadvantages of face routing; the possibility of tracing face boundaries in a non-optimal direction. Bounded Face Routing (BFR) employs an ellipse whose foci are the positions of a packet's source and destination. If an upper bound is assumed on the length of the shortest route from a source to a destination, then an ellipse can be drawn whose boundary is all points whose distances from the source and destination sum to that upper bound (by definition of an ellipse, the sum of the distances to the foci is constant for every point on the ellipse). It can be shown that a path through any point outside of the ellipse would have a length greater than the upper bound, thus guaranteeing that if the upper bound is correct, the shortest path lies either completely inside of or on the ellipse. BFR adds this bounding ellipse to face routing, causing the packet to turn around and explore a face boundary in the opposite direction if the next hop would have crossed the ellipse. If the ellipse boundary is reached while traveling around the face in the opposite direction, BFR fails, and there is no path to the destination within the ellipse (and no path shorter than the assumed upper bound).

Figure 2.15 shows an example of bounded face routing forwarding a packet through a slightly different topology than previous examples. The bounding ellipse with foci at $s$ ant $t$ is shown. A packet leaves $s$ and travels around the boundary of $F_0$ until reaching node $f$, recording the intersection point $p_0$. The packet cannot progress past $f$ because doing so would cause the packet to cross the bounding ellipse. The packet is the forwarded back past $s$ and on to node $c$ in a counter-clockwise direction. The packet once again would need to cross the bounding ellipse to progress further, and the packet's direction is reversed and it arrives

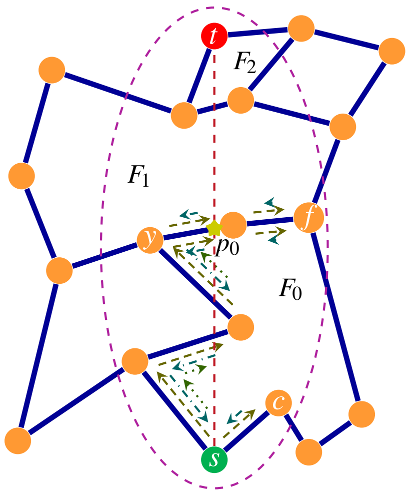


**Figure 2.15: A Bounded Face Routing example.**

at node $s$. Since an intersection point ($p_0$) was identified, the packet is forwarded around the face to node $y$, where the packet changes faces and begins exploring $F_1$. The packet attempts to route the packet in a clockwise direction, but doing so would cause it to cross the ellipse boundary, so it explores instead in a counter-clockwise direction until reaching node $f$, at which point it must reverse direction and returns to node $y$, since progressing further would require crossing the boundary. Bounding Face Routing has failed, and there is no path within the ellipse to the destination.

**2.2.2.2.4 Adaptive Face Routing (AFR)** Since the upper bound needed by BFR is not generally known, Adaptive Face Routing (AFR) [KWZ02] is used to adaptively determine an acceptable upper bound. AFR first selects an estimate for the upper bound (for example, twice the distance from source to destination), and then runs BFR. If BFR fails (i.e. there is no path to the destination within the ellipse), the upper bound (ellipse size) is increased (for example, doubled), and BFR is run again. This continues until eventually the ellipse becomes large enough that the packet is able to traverse the current face and move on to the next, or traces the entire boundary of the face (returning to the node where it began tracing the face) indicating that there is no path to the destination. Kuhn et al. prove that if the shortest path has cost $c$, AFR finds a route with cost $O(c^2)$. They further prove, using a worst case geometric construction, that since geometric routing algorithms don't store routing information at the nodes, any geometric routing protocol applied on a Unit Disk Graph has worst-case cost $\Omega(c^2)$. Therefore, AFR is asymptotically optimal.

**2.2.2.2.5 Other Face Routing (OFR)** In [KWZ03b] the same researchers who developed AFR introduce an alternative version of Face Routing called Other Face Routing (OFR). The only difference with this new algorithm is that when tracing the boundary of a face, rather than recording the point on $\overline{st}$ that intersects the face boundary nearest the destination, OFR instead finds the point anywhere on the boundary of the face that is nearest the destination. In Figure 2.12 this face change point rule is labeled *closest point*. Notice how using this rule in the example topology of the figure would only require the packet to route around the boundaries of two faces, rather than three or more. In the basic face routing method, the packet comes very near the destination while exploring $F_0$, but must backtrack to explore $F_2$. This minor change preserves the headway made by a packet toward the destination, but more importantly, it allows face routing to be combined with Greedy routing without losing its asymptotic optimality. This version of face routing may traverse a different set of faces than FR, and the face change points are no longer all on $\overline{st}$.

Kuhn et al. further note that when routing with OFR on a Gabriel Graph one may instead record the vertex (node) on the face boundary nearest to $t$ rather than the point nearest $t$ (in many cases they are the same). In Figure 2.12 this face change point rule is labeled *closest node*. Other Bounded Face Routing (OBFR) and Other Adaptive Face Routing (OAFR) are derived by applying this small change to BFR and AFR respectively. Use of this face change point rule, however, requires that the topology graph be a unit disk graph.

### 2.2.2.3 Hybrid Methods

Combining the strengths of both Greedy and Face methods has proven very effective, and this section describes just a handful of the most common hybrid greedy-face methods in the literature.

**2.2.2.3.1 Greedy Face Greedy (GFG)** Greedy Face Greedy (GFG) is briefly described in [BMSU99]. The notion is to support two modes while routing: a greedy mode and a face routing mode. Until a local minimum is reached, greedy routing is used. To overcome the local minimum FACE-2 is employed to route around a “hole” in the graph. Once the packet reaches a node that is strictly closer to the destination than the local minimum, the packet reverts to greedy mode. GFG combines the efficiency of greedy routing with the (conditional) delivery guarantees of face routing, yielding a very effective geographic routing protocol. Figure 2.10 shows an example of the Greedy Face Greedy protocol routing a packet on a sample topology.

**2.2.2.3.2 Greedy Perimeter Stateless Routing (GPSR)** The most frequently cited geographic routing protocol is Greedy Perimeter Stateless Routing (GPSR) [KK00]. GPSR differs only slightly from GFG. One difference is that GPSR introduces the notion of constructing and maintaining, at each node, a planar logical local topology subgraph. The Relative Neighborhood Graph (RNG) and the Gabriel Graph (GG) (see Section 2.1.4.2) are two methods for generating such a planar local topology given only local topology information. Use of this logical subgraph when face routing overcomes potential issues with crossing edges. Though a node may be able to communicate with many neighbors, when in face routing mode, a packet is only forwarded to nodes directly connected in the node's planar topology subgraph. The rules used to determine how to change from one face to the next and when to revert to greedy mode also differ between GPSR and GFG. Karp et al. also make a substantial contribution by presenting the results of several simulations demonstrating the effectiveness of the GPSR protocol in realistic mobile ad hoc networks, employing hundreds of mobile nodes in a high-fidelity simulation environment.

**2.2.2.3.3 Greedy Other Adaptive Face Routing (GOAFR)** Another geographic routing protocol is Greedy Other Adaptive Face Routing (GOAFR) [KWZ03b] (pronounced "gopher"). GOAFR adds Greedy Routing to OAFR (a geographic routing protocol that adds a small OFR change to AFR). A bounding ellipse is first initialized to contain all paths between $s$ and $t$ of length at most twice the distance from the source to the destination. Greedy steps are then taken (each forwarding to a node closer to the destination), until a local minimum is reached (or the destination is reached). While greedily forwarding, the bounding ellipse is expanded if the next greedy hop taken would intersect with the ellipse. When encountering a local minimum, OAFR is run to traverse just the next face, at which point the packet transitions back to Greedy mode, and the cycle repeats until the packet arrives at the destination, or the destination is deemed unreachable. Kuhn et al. show through analytical analysis, that GOAFR is asymptotically optimal, and through simulation, that it is average-case efficient. They also empirically show that GOAFR on average outperforms GPSR, producing shorter paths for all but the least dense scenarios and yielding lower latency and higher throughput.

**2.2.2.3.4 Greedy Other Adaptive Face Routing Plus (GOAFR+)** The last iteration of the GOAFR family of protocols is Greedy Other Adaptive Face Routing Plus (GOAFR+) [KWZZ03]. This version of the protocol encompasses two main improvements. First, an early fallback technique is introduced, that enables changing back to efficient greedy routing as soon as possible while in the face routing mode. Second, a circle centered at the destination is used instead of an ellipse to bound the search. To enable early fallback without losing asymptotic optimality, GOAFR+ adds two counters to each packet. The first counter, $p$, records, as the face boundary is explored, the number of nodes visited that are located closer to the destination than the last local minimum. Similarly, $q$ records the number of nodes visited that are located further from the destination than the last local minimum. Once a packet has visited more nodes closer to the destination than the product of a constant factor ($\sigma$) and the number of nodes further from the destination ($p < \sigma q$), it forwards the packet to the visited node that is closest to the destination and reverts to greedy routing.

The geographic routing protocol designed as a part of this dissertation borrows the bounding circle concept from GOAFR+. However, unlike GOAFR+, the circle applies only in the face routing mode, and the circle's size is never reduced. However, the initial circle size is intelligently selected when face routing begins. The GOAFR protocols assume routing to take place much faster than node movement. The purpose of this assumption is to ignore movement and topology changes, imagining instead that the algorithm is executing on temporarily stationary nodes. This is an unrealistic assumption, however, as eventually some packet will be in flight when there is a change in the network due to mobility. The geographic routing protocol described in Chapter 4 overcomes the need for this assumption.

### 2.2.3 Cross-link Detection Protocol (CLDP)

Many of the issues encountered when trying to utilize geographic routing in the real world, where topology graphs rarely match perfect Unit Disk Graphs, are classified in [KGKS05b]. In particular, planarization errors, and the causes of incorrect face changes are investigated. In real-world networks, the existence of radio-opaque obstacles may inhibit the communication between two nodes that are otherwise within communication range of one another. These radio-opaque obstacles in the environment may lead to topologies that violate the unit disk graph assumption. Methods that rely on the distributed generation of a planar subgraph, generally assume the network topology graph is a unit disk graph, and are affected adversely by these violations, leading to topology graphs with disconnected nodes and crossing edges.

Kim et al. also identify four different face change rules used by geographic routing protocols. When the *best intersection* rule is used, the intersection points of the face boundary and a line between the source and destination are determined, and the intersection point nearest the destination is used as the face change point (as in FR). Alternatively, the first point encountered while exploring a face boundary where the face boundary intersects a line between the source and destination can be used as the face change point. This is referred to as *first intersection* (used by FACE-2) and differs from the best intersection for concave faces. Another option is choosing the closest node on the face boundary as the face change point, called *closest-node other face routing* and used by OFR and GOAFR+ when routing on a Gabriel Graph. Finally, *closest-point other face routing* selects the point (not necessarily a node) on the boundary that is nearest the destination (also can be used by OFR and GOAFR+). Examples are given that show how the *first intersection* and *closest-node other face routing* rules do not always work correctly in real-world wireless networks. Storing a representation of a point in space in the packet header requires a fair number of bits, and leads to extra per-packet overhead. The routing protocol described in Chapter 4 overcomes the issues described above without needing to store intersection points in the packet header, while still enabling short efficient paths through the network. Motivated by the issues described above, Kim et al. introduce the Cross-link Detection Protocol (CLDP) in [KGKS05a]. In CLDP the faces incident at each node are probed to determine if they include crossing links (planarity violations). Messages are then exchanged between nodes and violating links are removed from consideration for use when face routing. CLDP involves extra overhead to probe and negotiate with neighbor nodes. It also has the possibility of introducing errors when nodes move and probed information becomes stale. The topology management protocol described in Chapter 3 solves these planarity violation issues by explicitly controlling the topology of a network of mobile nodes in a distributed fashion, and ensuring that it is planar.

### 2.2.4 Greedy Path Vector Face Routing (GPVFR)

One way to improve the performance of geographic routing is to give nodes more information about the network topology. In [LML05] Leong et al. seek to do just that with Greedy Path Vector Face Routing (GPVFR). The idea of GPVFR is to allow nodes to exchange Path Vector information, and use this information to make better geographic routing decisions. Path Vector Exchange Protocol (PVEX) is introduced to allow nodes to share path vector information with one another, such that every node knows the nodes on the boundary of each face it is incident to (up to some maximum face size). GPVFR begins routing a packet in greedy mode. Upon encountering a local minimum, GPVFR switches to Oblivious Path Vector Face Routing (OPVFR) mode that uses the path vector information in an attempt to find a node on one of the current node's face boundaries that is closer to the destination. If such a node is found, it is recorded as the OPVFR target and the packet is forwarded along the associated face boundary. If no such node is found, the protocol reverts to a version of face routing that uses the path vector information to select a direction in which to trace the face boundary. In either case, once a neighboring node is found that is closer to the destination than the position of the last local minimum, the packet reverts to greedy mode.

The geographic routing protocol described in Chapter 4 also utilizes topology information to improve the geographic routing efficiency. It similarly searches for the connected node nearest the destination and seeks to route toward it, using the topology information to select an appropriate direction for exploring the face boundary. Unlike, GPVFR, however, it does not use extra overhead to exchange the topology information, and its view can extend beyond the currently incident faces. GPVFR may not function well in highly-dynamic networks if stale path vector information causes packets to be routed incorrectly, and the paper assumes the network graph is "quasi-static", and that "it does not change for the duration of a particular routing activity"; an assumption that will not be true in our networks.

### 2.2.5 Airborne Network Management and Routing

#### 2.2.5.1 Using Flight Plans

In [TGK$^+$09], Tiwari et al. consider the use of flight plans to better manage a network of connected aircraft (an airborne network). Regardless of whether the network is commercial or military, the paths of most airborne nodes are pre-planned. The authors describe the advantages of using mission plans to improve communication planning, with airborne networks as their motivating example. They also describe FPE, a protocol for exchanging flight path information. The same group of authors detail MARP/MDP, a Mobility Aware Routing Protocol designed for airborne networks in [TGS$^+$08]. Their protocol routes packets based on a time-dependent topology pre-computed using the expected flight paths of all nodes. A time-indexed routing table that can route packets given different sets of rules at different times is then pre-computed. In addition, they introduce a protocol to disseminate information on deviations from pre-planned behavior. Simulations [KST$^+$10] indicate that their protocol performs better than Optimized Link State Routing (OLSR). This solution, however, assumes there will not be large variations from the planned routes in terms of both the path followed and the timing. This solution, also, doesn't appear to explicitly handle topology control. Further, this paper only considers the elliptical flight paths ("racetrack orbits") of small military networks, whereas the protocols developed in this dissertation seek to accommodate thousands of nodes generally flying point-to-point routes.

### 2.2.5.2 MAToC

MAToC, a system for topology management in airborne networks, is justified and outlined in [KGT$^+$09]. In MAToC, each node generates an optimized time-dependent topology from flight path information shared among nodes. The topology includes the assignment of optimal power, channel, and pointing direction for each antenna such that the worst case Signal to Interference and Noise Ratio (SINR) is maximized. MAToC then interacts directly with existing routing protocols (such as OLSR and AODV) to reduce overhead. A reactive component additionally handles links failures and forms backup links. The authors report the results of simulating their protocol in a simple scenario. Unfortunately, many of the details of the system are left out of the paper, and it is unclear, for example, how the flight plans are shared or utilized. The system does utilize directional antennas but considers only a small number of nodes flying military elliptical flight paths ("racetrack orbits").

### 2.2.5.3 AeroRP

Researchers at the University of Kansas developed the ANTP (airborne network and transport protocols) suite to enable reliable transmission of data in highly-dynamic military airborne networks [CRJ$^+$12]. The suite's routing protocol is called AeroRP [JS09][NGUS12], and it is designed to support aircraft flying close to one anther at Mach 3.5. It is a proactive routing protocol that uses location information and limited updates to build a forwarding table. Omnidirectional links are assumed, and nodes determine their neighbors, and in some modes their neighbors' positions, by snooping.

### 2.2.5.4 Free-Space Optics

Epstein and Mehta simulate an airborne network supported by Free-space Optical communication links (FSO) in [EM04]. They compare two standard routing protocols, Open Shortest Path First (OSPF) and Enhanced Interior Gateway Routing Protocol (EIGRP), in an airborne network where two separated ground stations communicate through a network of airborne nodes. The importance of fast response to link outages is emphasized, and OSPF is shown to not perform well due to its limited maximum update frequency of once per second. These routing protocols, however, would not function in dynamic large-scale commercial airborne networks.

#### 2.2.5.5 Geographic Load Share Routing (GLSR)

Medina studied connecting the aircraft flying in the North Atlantic Corridor into an airborne mesh network in [Med11][MHRR12][MHRR10]. The result of his efforts was Geographic Load Share Routing (GLSR). While Medina solves a slightly different problem than this dissertation, many of the principles and concepts are relevant. GLSR is essentially a greedy forwarding strategy, but instead of focusing solely on the greedy path, the transmission queue lengths are also considered. The node seeks to simultaneously maximize the packet's advance toward the destination and minimize the packet's queuing delay when determining where to forward each packet. The chosen next hop is said to maximize a packet's speed of advance (taking into account the possible delay on the transmission queue). This tends to load balance the link use and helps avoid bottleneck links. Long-range Omnidirectional links are assumed by Medina, and the density and traffic patterns of nodes in the network are assumed to be such that encountering local minima (or communications voids) while greedy routing is "extremely unlikely". A similar load sharing optimization could likely be applied to the routing protocol described in Chapter 4.

# CHAPTER 3: DISTRIBUTED EXPLICIT TOPOLOGY MANAGEMENT

In this chapter a new distributed topology management framework is described that is capable of managing a degree-constrained topology of an extremely large-scale network of mobile nodes while requiring no overhead or control messages except for the exchange of position information between nodes that are nearby one another. The chapter starts by introducing some terminology and explaining the expected system architecture. Next, the requirements of the topology management framework and the global topologies it generates are summarized. The details of the topology management framework are then given, followed by the details of several topology generation algorithms that can be used within the framework. Some potential pitfalls to avoid in designing further algorithms are described next. Finally, a detailed comparison of the topology generation algorithms and their attributes is presented.

## 3.1 Terminology

The description of this work is more clear with the addition of a few concepts and terms. In this section, these terms are defined and described.

### 3.1.1 Nodes and Links

A *node* is a single network entity. For airborne network applications, a node is an airborne platform (such as a manned fixed-wing aircraft, a drone, or a balloon). Each node bears a static number of *directional data links*, sometimes referred to just as *links*. These directional data links include a directional antenna that is mounted on a precision gimbal. Each directional data link can *point*, focusing its transmission energy in a specific direction. A link (a transceiver attached to a node) should not be confused with a *connection* that is formed when two directional data links point at and synchronize with one another such that data can be exchanged. Because of the unique qualities of directional data links, a few new terms are useful to more efficiently describe the relationships between nodes. These terms are described in the following sections.

#### 3.1.1.1 Neighborhood and Neighbors

The *neighborhood* of a node is the space around a node within which the node is able to form a direct connection with another node. The maximum distance over which two nodes can connect or remain connected is called the *link range*, or *maximum transmission range*, and is assumed in this work to be the same for every pair of network nodes. For simplicity, all distances can be normalized by this maximum range such that the normalized maximum transmission range between any pair of nodes is 1 unit. In this work, it is assumed that any pair of nodes separated by a normalized distance less than 1 unit can directly connect and communicate. On land, where radio-opaque obstacles are prevalent, this assumption is not very realistic. However, at the cruising altitude for commercial aircraft (above even most clouds) this assumption is more realistic. The neighborhood of a node, then, is assumed to be a unit disk (with radius 1 unit) centered at the node, in 2-dimensions (see Section 2.1.4.1), or a unit sphere (with radius 1 unit) centered at the node in 3-dimensions.

Naturally, any other node in a node's neighborhood is called a *neighbor*. Any node $x$ that is within the link range of node $t$, is considered a neighbor of $t$. If the topology of the network is explicitly controlled only some of the potentially many neighbors of $t$ are actually directly connected to $t$. For this reason, a separate designation is needed. A *directly connected neighbor* is a neighbor of a node to which it has a direct connection (i.e. their links are pointed at one another).

#### 3.1.1.2 Community and Community Members

Another aspect of this work is the exchange of position information between nearby nodes. To better conceptualize this, the new term *community* is used. The community of a node is the space within which the node is able to exchange position information with other nodes. Just as in a city a neighborhood is a part of a larger community, so also in this definition a community generally includes more nodes than the neighborhood. The distance over which position information may be exchanged is called the *position exchange range* or *community range*. The position exchange range will generally be longer than the link range because a different transmission scheme using an omnidirectional antenna and a lower bandwidth is assumed. As with the neighborhood, the community is approximated as a circle in 2-dimensions, though in reality, the shape will differ depending on what omnidirectional antenna is used. Lastly, any other node that is in a node's community is called a *community member*.

Figure 3.1 illustrates these relationships and those discussed in the previous section. The smaller circle represents the link range for node $t$. Nodes $x$ and $w$ are neighbors of node $t$. In addition node $x$ is currently a directly connected neighbor of node $t$, as indicated by the edge between them in the figure. The dotted outer circle represents the boundary of the community. Nodes $y$, $x$, and $w$ are members of $t$'s community but $y$ is not its neighbor. Node $z$ is neither a member of $t$'s community nor a neighbor of $t$.

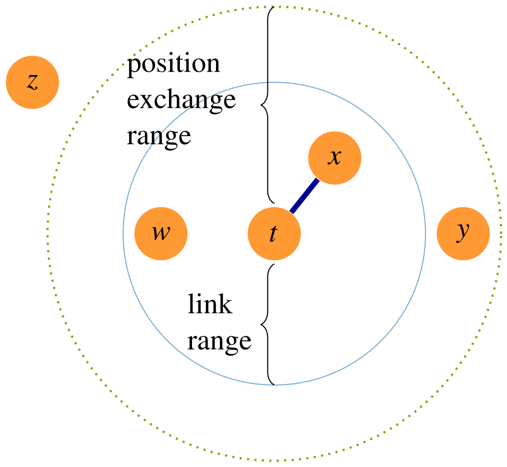


**Figure 3.1: An example showing how position exchange range, link range, the community, the neighborhood, and directly connected neighbors are defined.**

### 3.1.2 Network Density

One fundamental attribute of the network is its node density. The node density affects nearly every aspect of the network, including the network topology, the length of routing paths, whether nodes are able to remain connected, etc. The metric used in this work to describe the density of the network is the average number of *nodes per unit disk*. Remember that the neighborhood is approximated by a unit disk, where all distances have been normalized such that 1 unit is the maximum transmission range for all links. For a given snapshot of a topology, the nodes per unit disk metric, then, is the mean number of nodes present in the network in an area the size of the unit disk. In a network with a density of 10 nodes per unit disk, a node, on average will have 9 other nodes within its transmission range with which it could potentially form connections.

### 3.1.3 Network Topology and Connections

Using directional data links can lead to some unique situations and new terms. For example, a directional data link may be pointed at a neighbor, but that neighbor may not have (yet) reciprocated and pointed back.

This is labeled a *one-way connection* in this work.

A *topology* is a set of connections between nodes at a certain point in time. If only connections between nodes within a node's community are included in the topology, and it was computed using only information already available at each node, it is a *local topology*, whereas the *global topology* includes all of the connections in the network at a specific point in time. If the topology among a set of nodes changes over time, it is a *dynamic topology*. A *topology graph* is a representation of a topology where the connections are viewed as the edges between vertices in a 2-dimensional plane representing the nodes at the ends of the connections. As described in Section 2.1.4.2, a two-dimensional topology graph whose edges intersect only at vertices is a *planar graph*. A topology graph is *connected* if there is a path between every pair of vertices. If a topology graph is not connected, it is made up of a set of *connected components*, where each component is a connected subgraph of the topology graph.

In the network topology, the *maximum degree* of a node refers to the number of data links it has, or in other words, the number of connections it can be a part of at any one time. Similarly, the *degree* of a node or vertex in a graph is the number of edges currently incident at the node in the graph.

The set of *potential connections* for a node is the set of connections to each neighbor. As described in Section 2.1.4.1, the topology of a wireless network is often theoretically modeled as a *unit disk graph* (UDG), where edges exist only between all pairs of vertices whose separation distance is less than some threshold (i.e. the maximum transmission range). Figure 3.2 shows a unit disk graph or potential connections graph for a set of nodes that will be used to compare several topologies in the remainder of this chapter.

## 3.2 Proposed Architecture

Before diving into topology management, this section describes the architecture of the proposed network that motivates this work and that will be simulated for testing the protocols designed.

Tens of thousands of airborne platforms could simultaneously participate in a network. These airborne platforms could consist of fixed-wing aircraft, drones, balloons, or any mixture of those platforms. These platforms will be connected together with high-capacity connections into a mesh network, where each platform serves as both a user of the network and as a network router. This airborne network could also be tied to ground station gateways, connecting it to the Internet or other networks. Some nodes would use one of their links to point t a stationary ground station rather than another node. Figure 3.3 shows a high-level

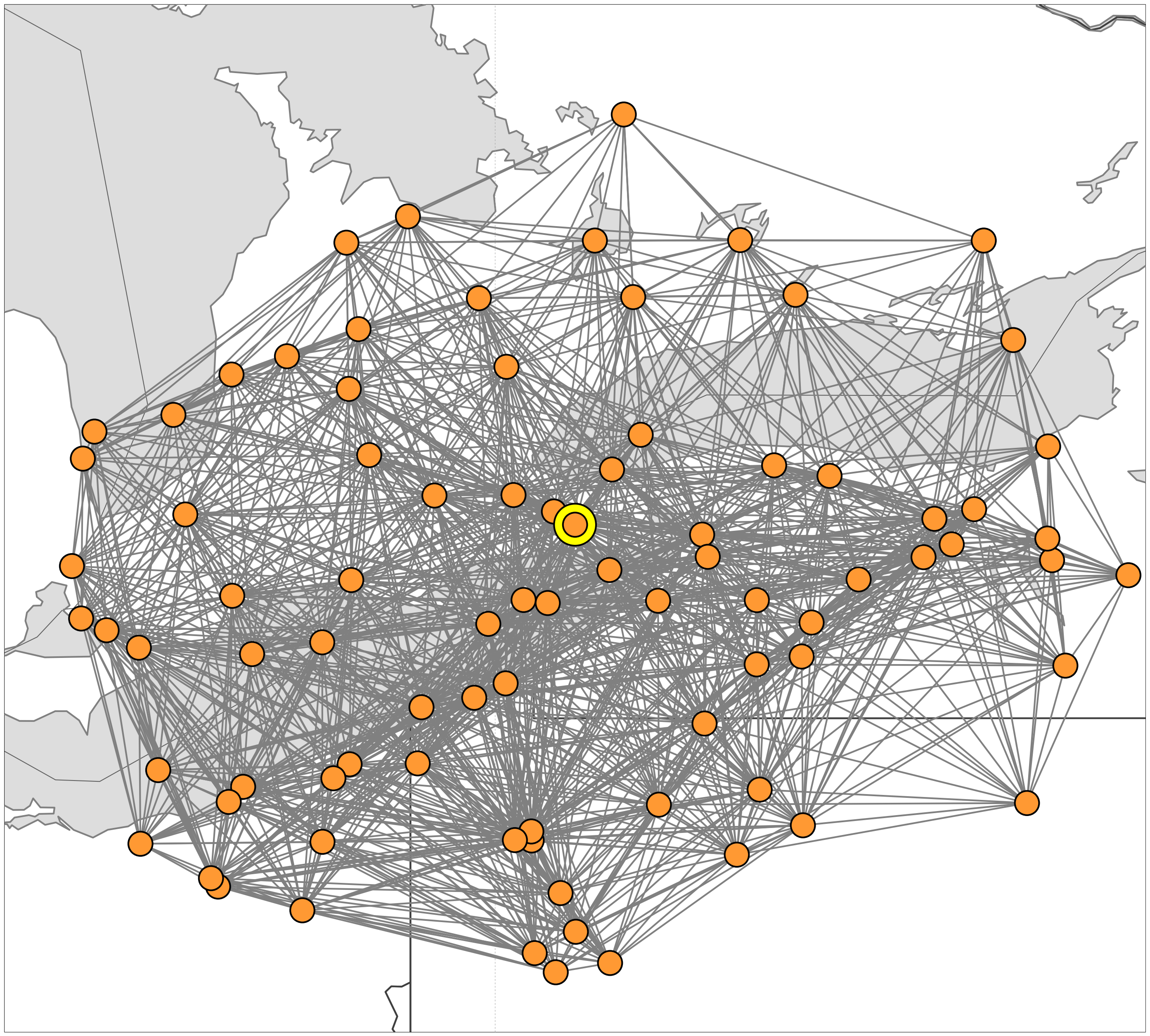


**Figure 3.2: The unit disk graph among a set of aircraft positions.**

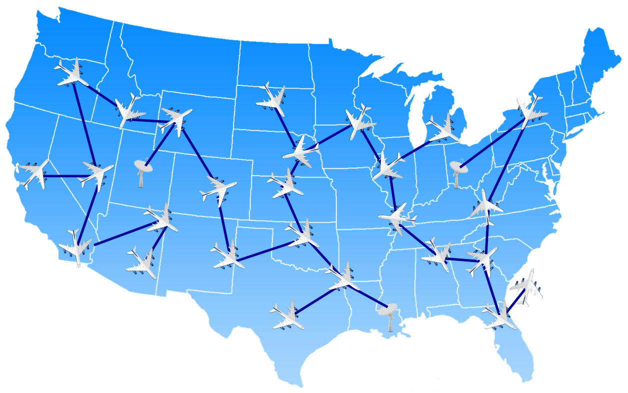


**Figure 3.3: A pictorial representation of the connection of aircraft into a mesh network and to ground station gateways.**

pictorial view of such a network, but obviously on a much smaller scale in terms of numbers of nodes.

For this work, the simulated nodes are commercial aircraft, but the algorithms and protocols are adaptable to other platform types or a mixture of platform types. Rather than implementing prototype networks with actual data links and platforms, the focus of this work is to design and simulate protocols for these large-scale airborne networks. Understanding the expected architecture of the network nodes, however, is beneficial. The architecture of each network node (airborne platform) can be divided into five different systems: a directional communication system, a position exchange system, a ground user communication system, an avionics system, and optionally an on-board communication system. Figure 3.4 shows each of these sub-systems and their connections. The following sections will detail each of these systems.

### 3.2.1 Directional Communication System

The Directional Communication System is responsible for forming the mesh network and routing data within it. The protocols and algorithms developed in this work support this system. The other systems provide needed data to this system or use the network it forms. The Directional Communication System consists of a node controller, a router, and some number of directional data links. Each of these components is described below.

#### 3.2.1.1 Node Controller and Router

The node controller consists of the processor and software needed to control the directional communication system on a node. The controller receives the positions of nearby nodes from the Position Exchange

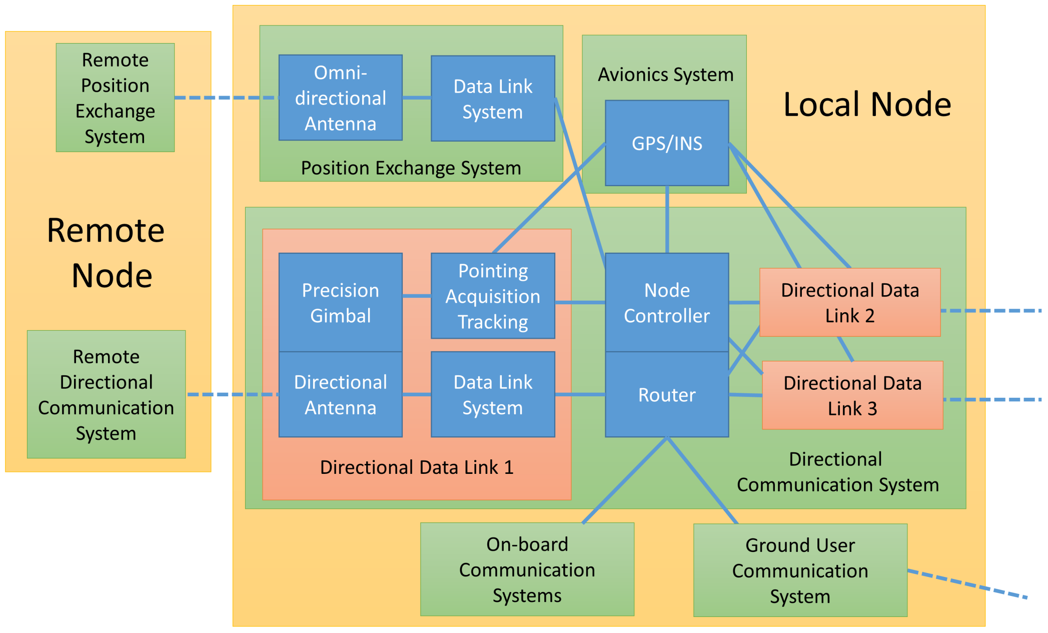


**Figure 3.4: A diagram of the system-level architecture for each airborne platform (node).**

System, and also receives this node's position information from the Avionics System. Using this position information the node controller can make informed decisions about which nodes it should connect with. The node controller explicitly manages the edges of the network topology incident at that node. When changes to theses incident edges in the network topology are desired, the controller communicates with the directional data links on the node, commanding them to change their pointing direction. The node controller is also tightly coupled with the router. As packets of data arrive from the data links, the router and the node controller work together to correctly route the packets. Each packet is either forwarded out another directional data link on its way to its destination or delivered to one of the other systems that connect to the router.

#### 3.2.1.2 Directional Data Link

Since high data rate communications are desired between nodes separated by long distances, some sort of directional data link is needed. The use of directional data links is a fundamental assumption of this work and motivates the need for the topology management framework. This work also assumes the links are mechanically steerable, though other Smart antennas (such as phased arrays) could be considered. The directional data links could be radio frequency (RF) data links, free-space optical communication (FSO) links, or hybrid FSO/RF data links (see Section 2.1.3). Each node is assumed to be outfitted with some number of these directional data links.

Each directional data link has a directional antenna. This is the physical component, mounted on the exterior of the airborne platform, that ultimately sends and receives the communication signals. In the case of RF directional data links, the directional antenna may be a small parabolic dish, similar to a satellite dish. In the case of FSO data links, the antenna would be an optical antenna, that is essentially a laser combined with a telescope and a specialized detector.

Each antenna is mounted on a precision gimbal that is able to, with an extremely high level of accuracy, point the directional data link at a target. The United States military has done this type of pointing for years with directional RF and FSO data links [CFG$^+$09].

Each gimbal is controlled by an integrated Position Acquisition and Tracking (PAT) system. The PAT system connects with the air platform's Inertial Navigation System (INS) (in the Avionics System) so that it can move to counteract the motion of the platform and constantly point at the target. The PAT system allows the antenna to (1) point at a new target based on input from the node controller, (2) acquire a connection with a remote node (that is pointing back), and (3) once connected, make minor corrections to the antenna pointing. In this way the remote node is tracked by the link, ensuring a consistent connection.

Each directional data link also has the necessary hardware and software to receive the stream of output from the antenna and translate it back into network packets. This is the role if the Data Link System component. This component would handle all of the modulation/demodulation, coding, amplification, processing, etc. The Data Link System is also responsible for exchanging signal strength values with the remote data link and constantly making minor corrections to "zero-in" on the pointing direction that yields the best connection.

For simplicity, it is assumed that each node has the same number and same type of directional data links. The topology management protocol, however, can support nodes with different numbers and types of directional data links. To support this, either a database associating each node's capabilities with its unique node ID would need to be available at each node, or this information would need to be exchanged with nearby nodes in the same fashion as position information is exchanged.

For most of the simulations and experiments conducted for this work, three directional data links per node are assumed. If each node had only two data links, the network topology would essentially be one long "snake" that interconnected the nodes. In such a topology, packets sent between the head and tail nodes would have to visit every node in the network before reaching their destination (though routing would be greatly simplified). If each node is instead equipped with three data links, network topologies become much more interesting. A complex topology interconnecting all the nodes can generally be formed assuming an adequate node density.

Equipping the airborne platforms with more than three directional links can lead to diminishing returns in terms of link utilization. The extra weight, expense, and power needed for more data links is difficult to justify. For these reasons, the simulations in this work have been restricted to the use of only three directional data links per node.

### 3.2.2 Position Exchange System

Each node is assumed to have a separate system that can exchange positions with nearby nodes. On commercial aircraft, the existing ADS-B system fills this role. Essentially the Position Exchange system consists of an omnidirectional antenna and a data link system. The omnidirectional antenna is able to broadcast ADS-B messages to nearby nodes in any direction. Similarly, ADS-B messages from other nearby aircraft (within about 288km) are received by the antenna. On other platform types, some system must enable the exchange of position information with nearby nodes. The Data Link System of the Position Exchange System connects to the antenna, and handles all the processing to obtain the position data from the transmissions. The Data Link System also shares the position information with the Node Controller.

### 3.2.3 Ground User Communication System

The Ground User Communication System is a separate system, the purpose of which is to provide data services to users on the ground. This System would connect to the Directional Communication System router to gain access to the backbone network. Once the backbone network is connected to ground station gateways and the Internet, the Internet is then reachable from the Ground User Communication System as well. The design and simulation of this system is not the goal of this work, and it is instead seen as a "customer" of the high-bandwidth communication link provided by the Directional Communication System. One can imagine various designs for such a system. It could use proprietary communication protocols and communicate with special hardware attached to homes or buildings, or it could operate much like a cell tower, providing 4G LTE connectivity to users devices. Companies like Facebook, Google and SpaceX are actively working on designing these types of systems.

### 3.2.4 Avionics System

The avionics system allows the Directional Communication System access to the Node's positioning and navigation systems. The Global Positioning System (GPS) and the Inertial Navigation System (INS) provide updated position and orientation information needed to maintain the correct pointing of the directional links. They also allow the node controller to know the node's current position and allow it to more accurately predict the node's future positions.

### 3.2.5 On-board Communication Systems

Lastly, the On-board Communication Systems are all other systems on the airborne platform that could use a high-bandwidth connection to other airborne platforms or to the Internet. Wireless routers to which passengers connect while on-board an aircraft are one example. Any other system on the airborne platform that needs to communicate command, control, or status information could also use this data path. On commercial aircraft, for example, non-emergency engine status information could be sent using the airborne network. Flight attendant and pilot communications with the ground or with other aircraft (even those out of radio range) could also be facilitated within the airborne network as an on-board communication system.

## 3.3 Topology Management Requirements

With new terms defined and the architecture described, the topology management framework can be explored. First, the goals of the global topology and this management framework are detailed.

### 3.3.1 Global Topology Requirements

As mentioned in Section 3.1.3, the global topology of a network is the topology at a specific point in time that includes all of the connections between any of the nodes in the network. With free-space optical links or the mechanically steerable RF links, it is assumed that each link can only connect with one other node. Because of this, and because each node is fitted with a small number of directional data links, the degree of each node (or vertex) in the network topology must necessarily be constrained in the global topology graph. In other words, if an aircraft has only three lasers to point and communicate with, it can have at most 3 incident edges in the global topology graph. Therefore, one of the most important requirements of any candidate global topology is that it must be degree constrained such that no vertex in the topology graph has more incident edges than the number of data links at that node (assuming one connection per link).

Next, two nodes separated by a greater distance than the maximum transmission range, will be unable to connect and communicate. Therefore, an obvious requirement of the global topology graph is that no edge be longer than the maximum transmission range. If distances are normalized by the maximum transmission range, and all links have a uniform range, this requirement can be satisfied by ensuring that the global topology graph is a subgraph of the unit disk graph (UDG) of all nodes in the network.

Also, the global topology graph connecting nodes in the network should connect as many nodes as possible into a single connected component. This objective will help guard against partitions in the network. Limiting network partitions will help ensure that any node is reachable from any other node and that data can be routed efficiently between nodes.

Lastly, it is desirable that the global topology be planar, ensuring that there are no crossing edges in the topology graph. This optional requirement is desirable to benefit the routing protocol described in Chapter 4.

Ensuring that a topology meets all these requirements is tricky, and doing so in a distributed manner is even more onerous.

### 3.3.2 Requirements of Topology Management

Once a topology has been generated meeting the requirements described above, it must be actively managed over time as nodes join the network, leave the network, and move about. This section describes the requirements of this management of the topology.

First, the topology management framework must be scalable. It is estimated that Google would need at least 6,000 balloons, perfectly positioned in the stratosphere, just to provide coverage to the entire continental United States. The number of aircraft in the skies above the United states is even greater, at around 7,000 aircraft, including general aviation traffic. However, the United States is just a portion of the potential worldwide coverage desired by such a network. The network would need to expand to tens or even eventually hundreds of thousands of airborne nodes. One hundred to four hundred thousand nodes would not be unheard of. Whatever system is used to manage the topology of this vast network and its ground stations would need to be highly scalable.

To enable this scalability, the topology management should be distributed, avoiding the bottlenecks and headaches of having a single centralized manager of the topology. Airborne networks that use a centralized scheme for managing the physical topology of the network are not able to react as quickly to changes in the network. Nodes would need to report their status to the centralized controller, and then wait for commands from the controller before acting. Also, these centralized topology management schemes would either require a separate channel for communicating control messages or would suffer from the possibility that unplanned changes to the network could cause control messages to fail to reach their destinations. Further, these control messages would consume greater amounts of the bandwidth provided by the connections they are managing as the network scaled to support larger numbers of nodes.

If instead, a distributed topology management scheme were employed, each node could make its own decisions about the connections it forms in the topology. Such a management scheme would ensure agility, but with low (or no) control message overhead at the cost of leaving some data links unconnected.

Another requirement relates to the required computational resources of such a management scheme. While the computational resources available at the nodes are anticipated to be much greater than those of most sensor network nodes, they are still limited. Therefore, the topology management scheme must be computationally efficient. If, for example, the management framework were designed to recompute a topology once a second, the algorithm must obviously be able to compute a topology in less than one second for the expected node densities.

Lastly, it would be desirable and impressive if the management scheme could operate with absolutely no communication overhead on the high-capacity links between the nodes. If the topology could be managed with no control messages, and no extra information exchanged between the nodes, it would leave the entire link bandwidth for customer communications. To accomplish this, a node would need to use only the information already available locally to manage the network topology. In the following sections, we will detail a topology management scheme that meets all of these requirements, even this one.

## 3.4 Topology Management Framework

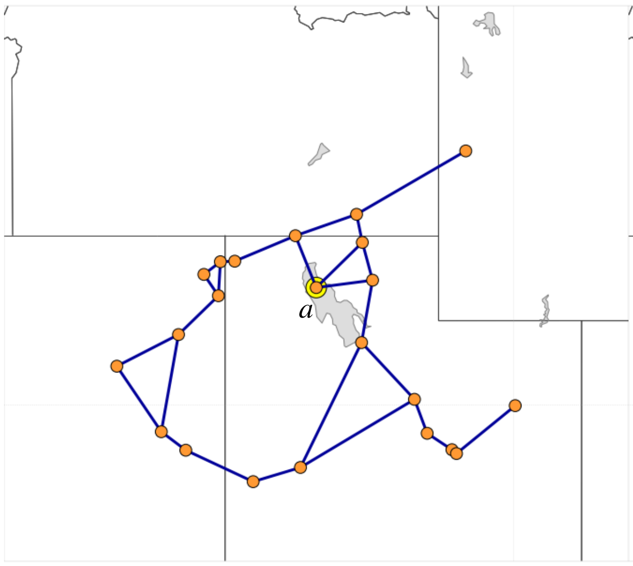
This section describes a framework sufficient to manage the topology while satisfying the requirements described above. The framework operates in the following manner. First, nodes periodically broadcast their position information (via a separate omnidirectional system) such that each node knows the position of every other node with some position exchange range of itself. Next, each node periodically uses a topology generation algorithm to compute a topology connecting the nodes for which it has positions. Finally, each node attempts to form its incident connections in the generated topology. The remainder of the section gives more details on this simple procedure.

As mentioned above, to facilitate the topology management each node regularly exchanges position information with every other node in its community (every node within a given range). This is the only communication overhead of the proposed topology management method. This does not violate our goal of operating with no extra communication overhead in applications where the position information is already exchanged by nearby nodes. In our example application of commercial aircraft, the FAA's new ADS-B systems already takes care of exchanging aircraft position information between nearby aircraft.

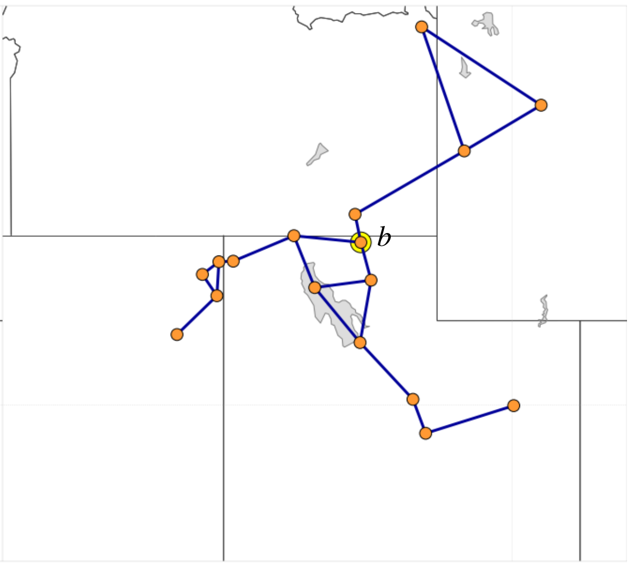
With this position information, each node periodically and independently uses a specific topology generation algorithm to compute a local topology that connects the nodes in its community into a network. This local topology is essentially an "opinion" of what the local node feels would be the best topology among the nodes in its community, given its limited view of the world. Next, each node in the network seeks to form the connections of which it is a part, in its local "opinion" topology.

If nodes run the same topology generation algorithm, in theory, nodes located geographically close to one another will share nearly the same set of community members. Therefore, there is a high probability that their computed topologies (their opinions) will be very similar, if not identical. If one node believes it should connect to the other, the other will likely reciprocate. Thus, the connections can be established without any handshake or extra communication overhead.

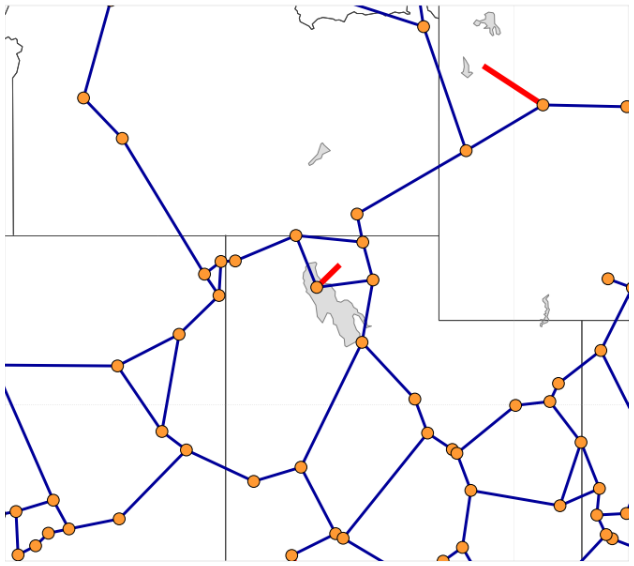
However, if nodes are further apart, it is more likely that differences in the communities of the two nodes will lead to a difference of opinion as to what the local topology should look like. In these situations it is possible for a local node to attempt to establish a connection, while the remote node intentionally points at different neighbors (as specified by its local topology), yielding a one-way connection. Figure 3.5 shows an example of a one-way connection in a topology graph connecting aircraft flying over the Northern Utah area of the United States on July 9, 2015, at 11:00 a.m. CT. Figures 3.5a and 3.5b each show the local topology among the community nodes for the node highlighted with a second (yellow) circle. Node *a* is flying over the Great Salt Lake (lake shown in gray), and node *b* is flying just northeast of there. Notice that there is disagreement between these two nodes about the direct connection they could share. Node *a*'s local topology includes the connection, while node *b*'s does not. Because of this disagreement, node *a* points one of its links at node *b* but a connection isn't formed since node *b* is pointing its three links elsewhere. Figure 3.5c shows the resulting global topology for the same area. One-way connections are displayed as thick (red) edges that span only half of the distance between the nodes. As second one-way connection can be seen in the upper-right of Figure 3.5c. Even though nodes do not explicitly communicate to determine which



**(a) Local Topology at *a*.**



**(b) Local Topology at *b*.**



**(c) Resulting Global Topology.**

**Figure 3.5: One-Way Connection Example.**

connections to form, there is generally good agreement between all the nodes, and a minimum number of one-way connections result.

These errors are the ‘cost’ of the distributed topology management. The choice of topology generation algorithm affects how often these errors occur, and how often nodes are disconnected from the network.

Our results will show that if an appropriate topology generation algorithm is employed, and the density of nodes is high enough, the result is an acceptable global topology. Most importantly, this method of topology management is scalable, easily coping with tens of thousands of nodes. It is also adaptable; able to rapidly respond to joining, leaving, or fast moving nodes. Other non-distributed topology management schemes would likely be less agile since communication latency and synchronization would likely delay a node's response.

To know when to compute the next topology, nodes are assumed to have a synchronized time source, such as that provided by a GPS. Each node computes its local topology periodically at about the same time as all other network nodes. In this way, nodes will simultaneously break old connections and form new connections. For the simulations in this work, the topology computation interval is generally assumed to be one second.

## 3.5 Topology Generation Algorithms

The effectiveness of the topology management method outlined above hinges on the selection or design of an appropriate topology generation algorithm. The topology generation algorithm takes a set of nodes and their positions and generates a set of connections between the nodes. A candidate topology generation algorithm, when used with the distributed topology management framework, must yield a global topology graph that satisfies the requirements described above. The global topology graph is not directly computed, but rather is the topology resulting from each node forming the local connections specified by its local topology graph. In the global topology graph, the number of incident edges at any vertex must not be greater than the number of directional data links on the node it represents. It is also desirable that the global topology graph be planar such that edges meet only at vertices. Ensuring that a node computing its local topologies never violates its own degree constraint is sufficient to ensure that the global graph will always be degree-constrained. However, requiring that each local topology be planar is not sufficient to ensure that the global topology is planar (see Section 3.6.1).

Remember also that in addition to these requirements, there are also several desirable properties of a candidate topology generation algorithm. The topology generation algorithm should strive for high inclusion, connecting as many of the nodes as possible in the global topology into a single connected component. In general, when each local topology connects all of the node's community members, the global graph will have high inclusion. However, in rare circumstances, it is possible for a node to be disconnected from the global topology, though every node produced a local topology including every community member.

In addition, a suitable candidate local topology generation algorithm should induce a global topology that allows for short routing paths through the network, minimizing the number of hops between every pair of nodes. This is a difficult desire to satisfy given the degree and planarity constraints.

Finally, to the extent possible, the algorithm should form robust topologies where multiple paths exist between pairs of nodes, where one-way connections are few, and where link utilization is high. The node density can impact the performance of a topology generation algorithm, but the algorithm itself can also control to an extent these desired attributes. One algorithm, for example, may prioritize using all of the available links, but this could also lead to a larger number of one-way connections.

As mentioned before, designing an algorithm that meets all these constraints can be tricky. One may initially consider using Kruskal's or Primm's algorithm to compute a minimum spanning tree (MST) among the nodes, that would lead to a high level of inclusion, but the lowest level of robustness, as removing any edge would disconnect the graph. More importantly, the traditional minimum spanning tree problem is not degree-constrained, and adding a degree constraint makes it NP-hard [GJ79]. Tree-based reliable Topology (TRT) [ACK04] is a topology generation algorithm that preserves connectivity and robustness, but unfortunately, it also does not constrain the degree. In general, effective topology generation algorithms must balance the trade-offs among these characteristics. For example, including long edges in the topology graph results in topologies with lower average hop counts when routing in the network, but long edges also lead to a higher occurrence of one-way connections. Similarly, better connectivity could sometimes be obtained but at the cost of an increase in computational complexity. The goodness of a topology has an impact on the efficiency of routing packets through it, and seemingly insignificant changes to an algorithm can transform a network, with relatively short paths between pairs of nodes, into one where paths don't even exist between most pairs of nodes.

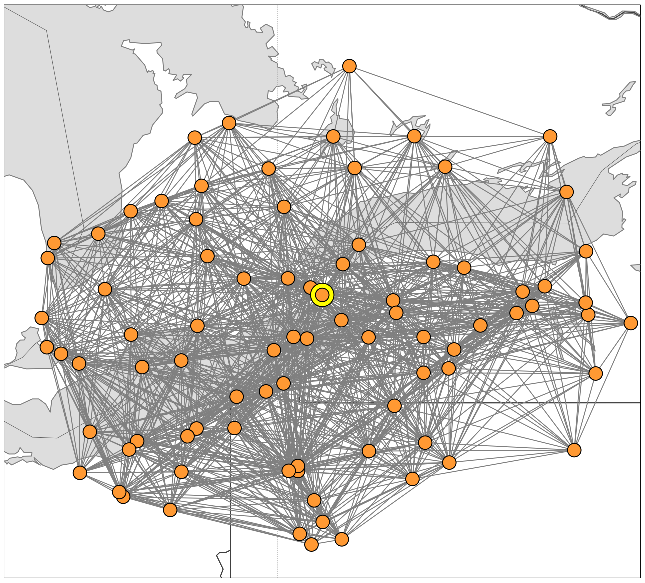
We have designed and experimented with several topology generation algorithms that meet the requirements described above. Each candidate algorithm will be discussed below. The main input to each of these algorithms is a potential connections graph or unit disk graph. In potential connections graphs, connections exist between any two nodes whose separation distance is less than some maximum range. Figure 3.6a shows such a graph for a set of nodes that will be used to study the topology generation algorithms. The positions for these nodes are obtained from an FAA Aircraft Situation Display to Industry (ASDI) dataset and represent the positions of actual aircraft on July 9, 2015, at 13:45 CT flying over the great lakes region of the northern United States. The connected nodes are all within 288 km (the position exchange or community range) of the node that is calculating the topology for this set of nodes.

### 3.5.1 Degree-Constrained Planar (DCP) Kruskal

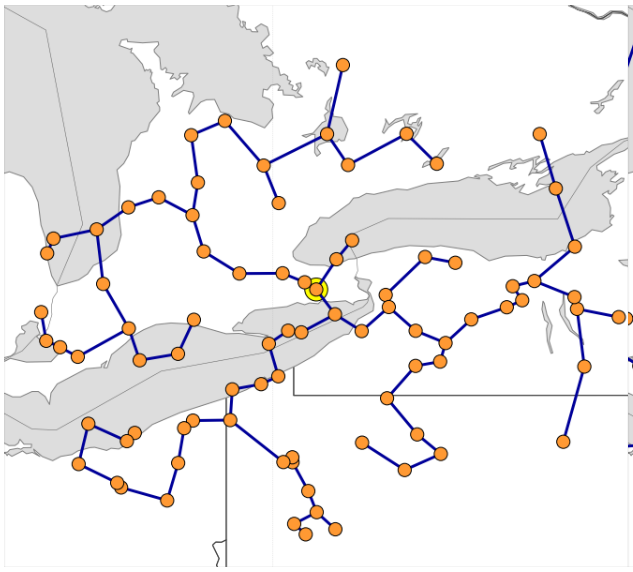
The first topology generation algorithm described is called Degree-Constrained Planar Kruskal (DCP-KRUSKAL). Figure 3.6b shows the result of using DCP-KRUSKAL to form a topology among the given set of nodes. DCP-KRUSKAL is not a great topology candidate by itself because of its lack of robustness to link failure. Since the global topology is much like a spanning tree a single link failure can in some cases partition the network or at least lead to very long paths stretching around large faces. DCP-KRUSKAL is, however, a useful subroutine for the other candidate algorithms, where, for example, it can be used to find a degree-constrained spanning tree within a planar subgraph of the potential connections. This algorithm is an augmented version of Kruskal's original Minimum Spanning Tree algorithm. It has been modified so that when edges are added to the resulting graph they do not violate planarity or degree constraints. To the original algorithm are added conditions to ensure that the resulting graph is planar (no crossing edges) and degree-constrained (number of incident edges limited). Initially, a vertex set is created for each vertex in the input graph, where each set initially contains a single item, called the associated vertex. In addition, an array is created to hold the number of incident edges (degree) at each vertex, and each value is initialized to 0. The input to the algorithm is a graph, and the output is expected to be a subgraph of the input graph. The edges of the input graph are first sorted by length (or weight), such that the smallest are first. For each edge in the sorted list, if (1) the vertices it connects are not already in the same set, (2) both vertices it connects have fewer than the maximum number of incident edges, and (3) the edge being considered does not intersect any edge in the current list of edges in the resulting graph, then the edge is in the DCP-KRUSKAL graph. Specifically, the following happens for each edge meeting the above conditions (1) the edge is added to the list of edges in the resulting graph, (2) the sets of both vertices are combined, and (3) the number of incident edges at the vertices is incremented.

This seemingly minor change of honoring planarity and degree constraints is detrimental to the properties of the minimum spanning tree, and the resulting graph no longer always spans a connected input graph, nor is the resulting graph always the minimum (in terms of the total sum of edge lengths) of the graph's spanning trees. Despite these facts, we find DCP-KRUSKAL to be a helpful and computationally efficient subroutine used by other topology generation algorithms.

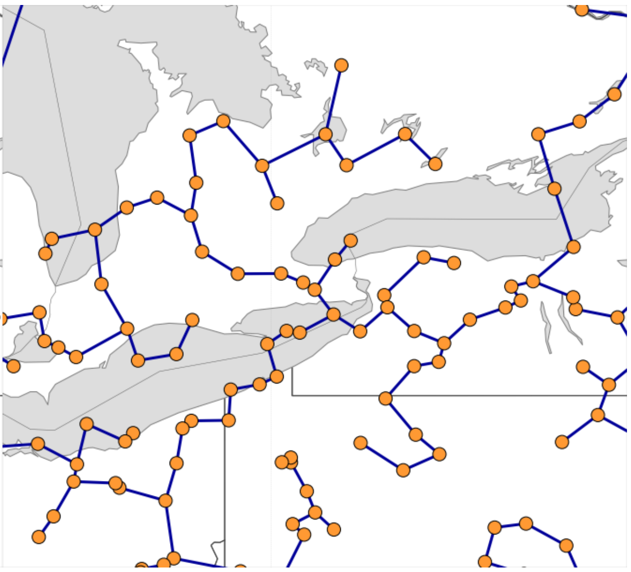
The attributes of each of the algorithms will be analyzed and compared in Section 3.7.



**(a) Unit disk graph**



**(b) Local DCP-KRUSKAL**



**(c) Global DCP-KRUSKAL**

**Figure 3.6: The unit disk graph and the local and global graphs resulting from running the DCP-KRUSKAL algorithm.**

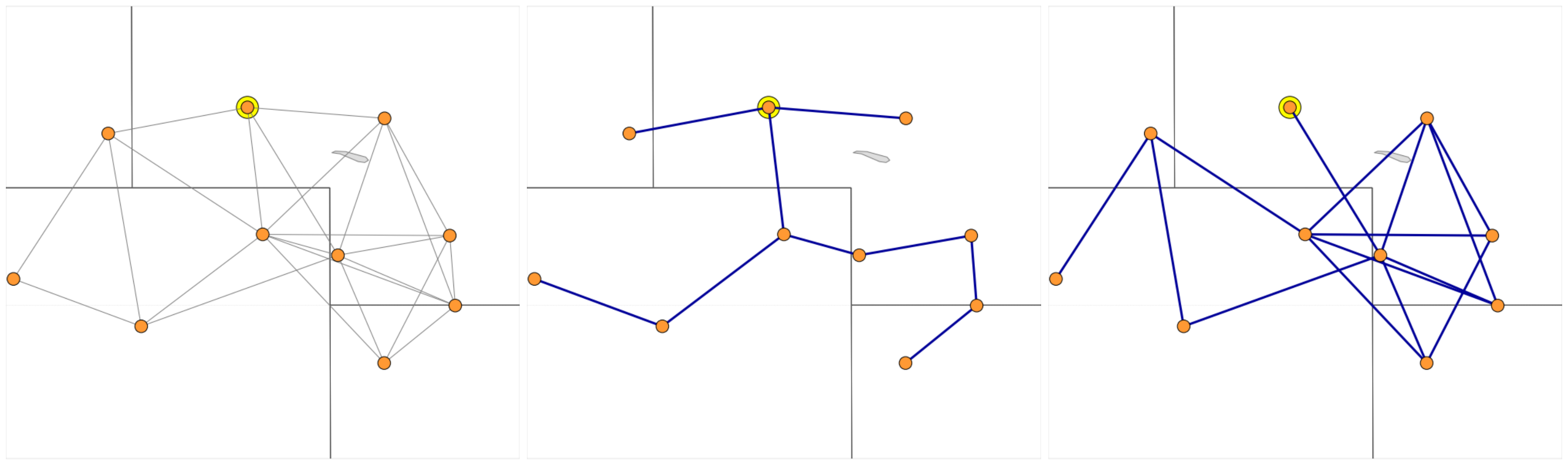
### 3.5.2 Degree-Constrained Planar Tree-based Reliable Topology

The Degree-Constrained Planar Tree-based Reliable Topology (DCP-TRT) algorithm is an adapted version of the Local Tree-based Reliable Topology (LTRT) algorithm [MNAK09]. The algorithm has been modified to ensure degree and planarity constraints are honored. The result is DCP-TRT, that forms robust topologies by carefully concatenating two spanning trees. An initial spanning tree is constructed, and then the second spanning tree is constructed such that it does not use any edge that is a part of the initial spanning tree. This forces the second spanning tree to introduce spanning edges not considered by the first, thereby increasing the robustness of the resulting combined topology.

Figure 3.7 shows a simple example of computing a DCP-TRT topology among 10 nodes. First DCP-KRUSKAL is called on the input potential connections graph or unit disk graph G, producing a degree-constrained spanning graph T. Next, T's edges are removed from the input graph (yielding G-T), and with these edges removed, the graph is passed to DCP-KRUSKAL again. The result is another degree-constrained spanning tree U. Notice that in this example since shorter edges are added to the spanning tree first, there is no way to connect to the highlighted local node in U. Finally, each edge of U is then added to T, longest first, unless adding the edge would cause a vertex to have more than the prescribed number of incident edges (violating the degree constraint), or the edge would cross one of the edges already in T (violating the planarity constraint).

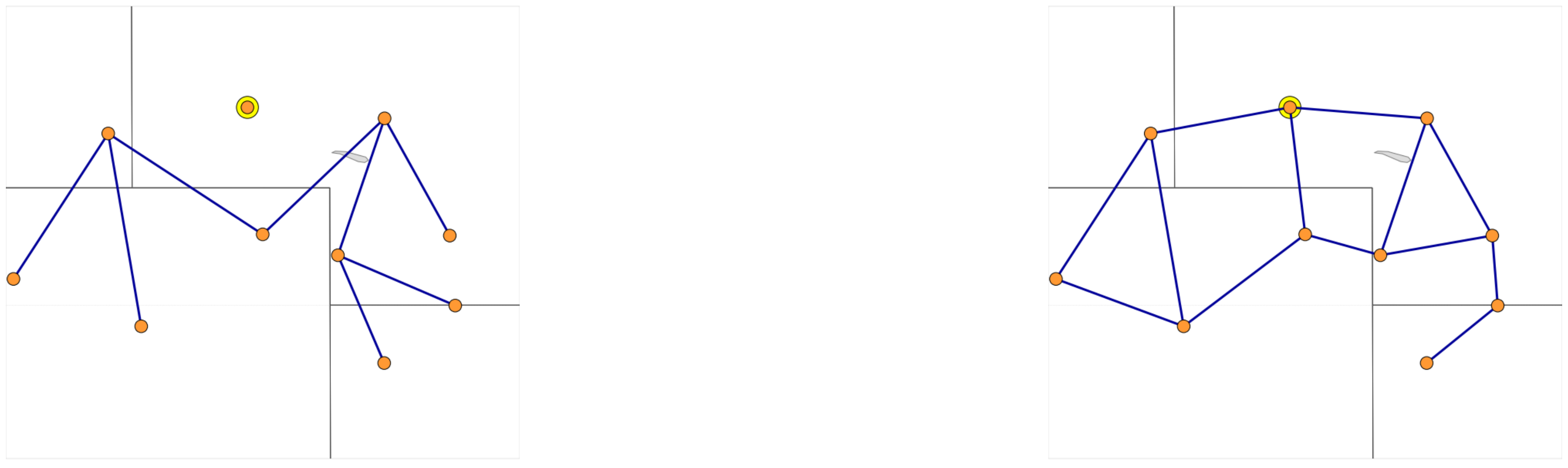
Note that if a vertex in the resulting topology graph has a degree less than the maximum degree, either the topology generation algorithm didn't use all the viable edges, or it is impossible to add an edge to the node without the edge (1) raising another node's degree above the maximum degree, or (2) crossing an edge already in the graph (making the graph non-planar). Unfortunately, there are rare situations where the use of this topology generation algorithm can produce a globally non-planar topology, though every local topology is planar. This is undesirable when a geographic routing protocol is used in the network, since, as explained in Section 4.4, a topology that is not globally non-planar can cause packets to get stuck in routing loops.

Figure 3.8 shows a more complicated example where DCP-TRT is used to construct a topology among the same set of aircraft flying over the great lakes region seen in the previous example. Remember that a planar graph divides the plane into a set of faces, each bounded by the edges of the graph. Notice that this algorithm generates a topology that has multiple paths between most pairs of nodes, but generally creates a planar graph with large faces.


(a) G

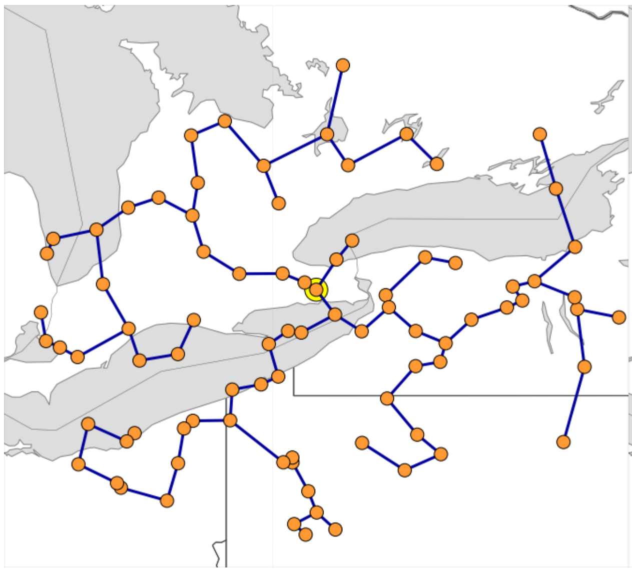
(b) T

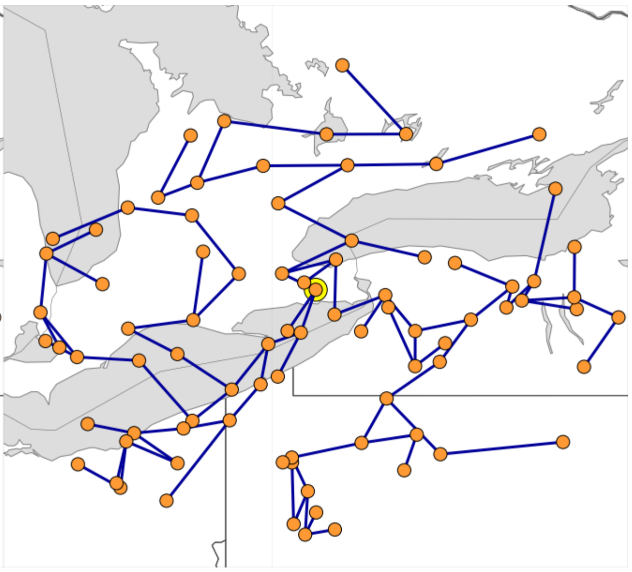
(c) G-T


(d) U

(e) T+U (DCP)

Figure 3.7: The steps of computing DCP-TRT.


**(a) DCP-KRUSKAL Graph**


**(b) DCP-KRUSKAL of result of edge removal**

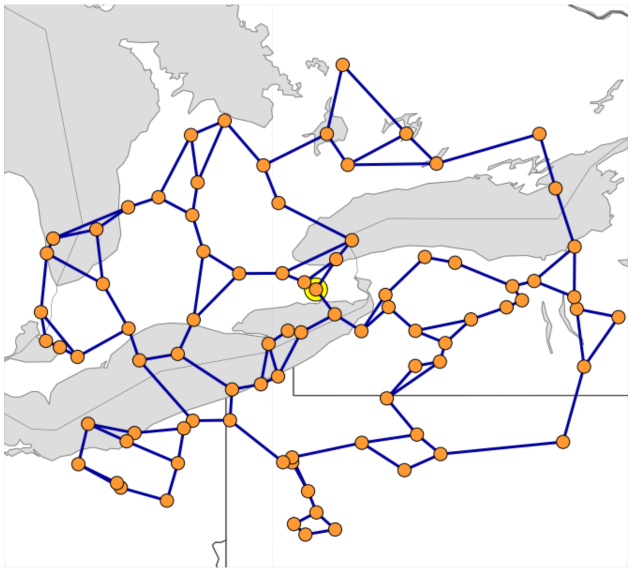

**(c) Final Local DCP-TRT Graph**

**Figure 3.8: The DCP-TRT topology among the sample set of nodes.**

Figure 3.9 shows the final global topology obtained by having each node compute its local topology and then having each node attempt to form the connections represented by the edges incident to the node in the resulting graph. Thick (red) lines represent one-way connections in the figure.

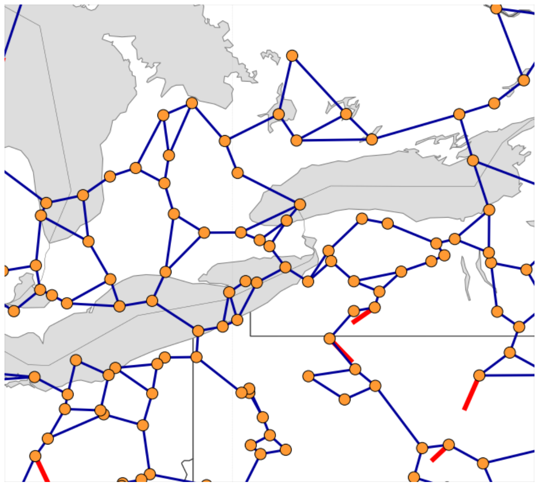


**Figure 3.9: The resulting global DCP-TRT topology.**

### 3.5.3 Degree-Constrained Gabriel Graph

As described in Section 2.1.4.2 the Gabriel Graph can be constructed for a set of points in the Euclidean plane. In a Gabriel Graph, an edge exists between a pair of points (*a*,*b*) if a disk, centered at the midpoint between *a* and *b*, with points *a* and *b* on its perimeter, contains no other points in the set. In other words, to determine the Gabriel Graph for an input set of points, for each pair of points $(a,b)$, determine the circle whose diameter is the same as the length of the line segment *ab*, where *a* and *b* are on the circle, and check if any other point in the input set is inside of or on the boundary of that circle. If so, the edge should not appear in the graph, otherwise, it should.

The Gabriel Graph can be shown to be planar by the following reasoning. According to Thales' theorem, the diameter of a circle always subtends a right angle to any point on a circle. Therefore, for any point, *c*, on the circle of which *ab* is a diameter, the measure of the angle $\angle acb$ is always 90°. By extension, if a vertex *c* is inside of the circle, the $\angle acb$ is always obtuse (greater than 90°). Similarly, if a vertex, *c* lies outside the circle, the measure of the angle $\angle acb$ is always acute. We can prove that any Gabriel Graph is planar, by first assuming that a Gabriel Graph contains two edges *ab* and *cd* that do intersect. Note that the sum of the angles $\angle acb$, $\angle cbd$, $\angle bda$, and $\angle dac$ must equal 360° according to the quadrilateral interior angle sum formula. This implies that at least one of those angles, say $\angle acb$, for example, is at least 90°. If that is true, however, the edge *ab* should not exist in the graph since *c* is either on the boundary of or inside of the circle for which *ab* is a diameter given the measure of the angle. Therefore, the edge *ab* should not be in the Gabriel Graph, a contradiction. As a consequence, no two edges of the Gabriel Graph may intersect, and thus every Gabriel Graph is planar.

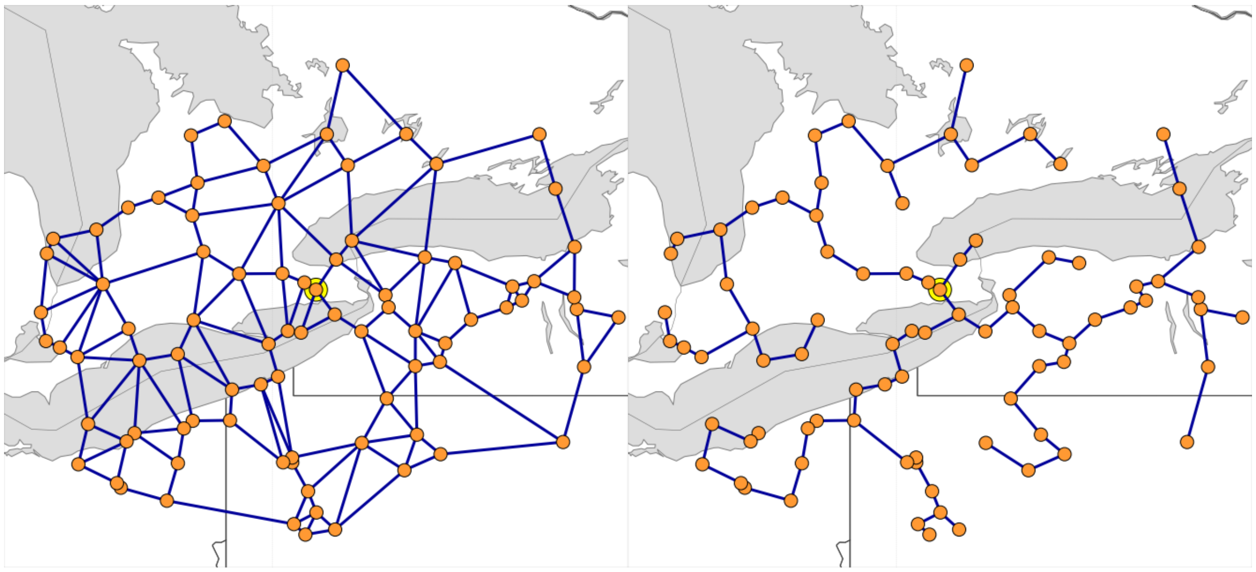
While Gabriel Graphs are planar, they are not degree constrained (see Figure 3.10a for an example). DC-GG is, therefore, introduced as a new degree-constrained version of the Gabriel Graph algorithm that works well for distributed topology management. Figure 3.10 shows the steps followed in computing DC-GG among the same set of aircraft seen in previous examples. Note that rather than starting with a set of nodes (points), the DC-GG actually starts with the potential connections or unit disk graph for the community nodes, and generates a subgraph of this input graph. The Gabriel Graph of the potential connections graph can also be called the Unit Gabriel Graph, as it includes all edges in the standard Gabriel Graph except those that are longer than the maximum transmission range. The Gabriel Graph is computed from the local node's potential connection graph, yielding the unit Gabriel Graph shown as GG in Figure 3.10a). Next, the DCP-KRUSKAL algorithm is used to find a degree-constrained spanning tree (K in Figure 3.10b) within the Gabriel Graph. Doing this for all nodes ensures that whenever possible, the final combined graph will span the nodes in the network. Finally, the unused edges in the original Gabriel Graph are transferred to the spanning tree graph, longest first, until no more edges can be added without violating the degree constraints. The result is the Degree-Constrained Gabriel Graph which can be seen in Figure 3.10c. Adding the longest edges first tends to create a topology where paths have shorter numbers of hops, a desirable quality for routing purposes.

Figure 3.11 shows the global topology resulting from each node independently computing their local topologies, and forming connections accordingly.

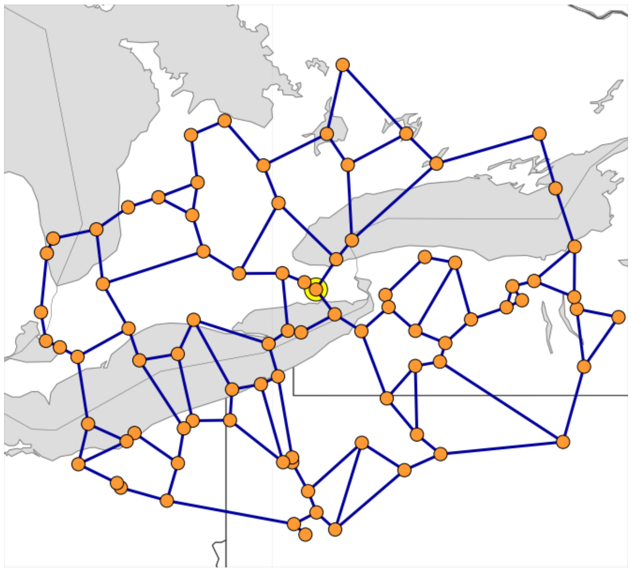
### 3.5.4 Degree-Constrained Delaunay Triangulation

The Delaunay Triangulation can be similarly employed as a subroutine in the generation of a network topology. As described in Section 2.1.4.2, Delaunay Triangulation is a triangulation of a set of points such that a triangle exists in the graph if and only if the candidate triangle's circumcircle (that passes through all the triangle's vertices) contains no other point in the set of input points. The Gabriel Graph is a subgraph of Delaunay Triangulation and Delaunay Triangulation is the dual graph of the Voronoi diagram.

The Degree-Constrained Delaunay Triangulation (DC-DT) is produced by first generating the Delaunay



**(a) Unit Gabriel Graph (GG)** **(b) DCP-Kruskal (K)**



**(c) DC-GG**

**Figure 3.10: The steps to computing the local DC-GG.**

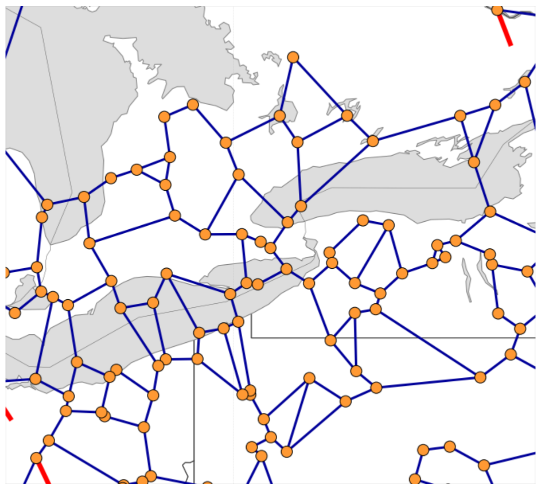


**Figure 3.11: The resulting global DC-GG topology.**

Triangulation (DT) of the local node's community nodes. Next any edges in the triangulation that are not in the Unit Disk Graph (edges longer than the maximum range) are removed, yielding the Unit Delaunay Triangulation (UDT). Next, DCP-KRUSKAL produces a spanning tree (K) within the triangulation, ensuring the resulting graph will span all the nodes. To this graph are transferred additional edges, longest first, from the Delaunay Triangulation, until each node's degree constraint is met (where possible). Figure 3.12a shows the Delaunay Triangulation, Figure 3.12b the Unit Delaunay Triangulation, and Figure 3.12d resulting local DC-DT topology for the set of nodes.

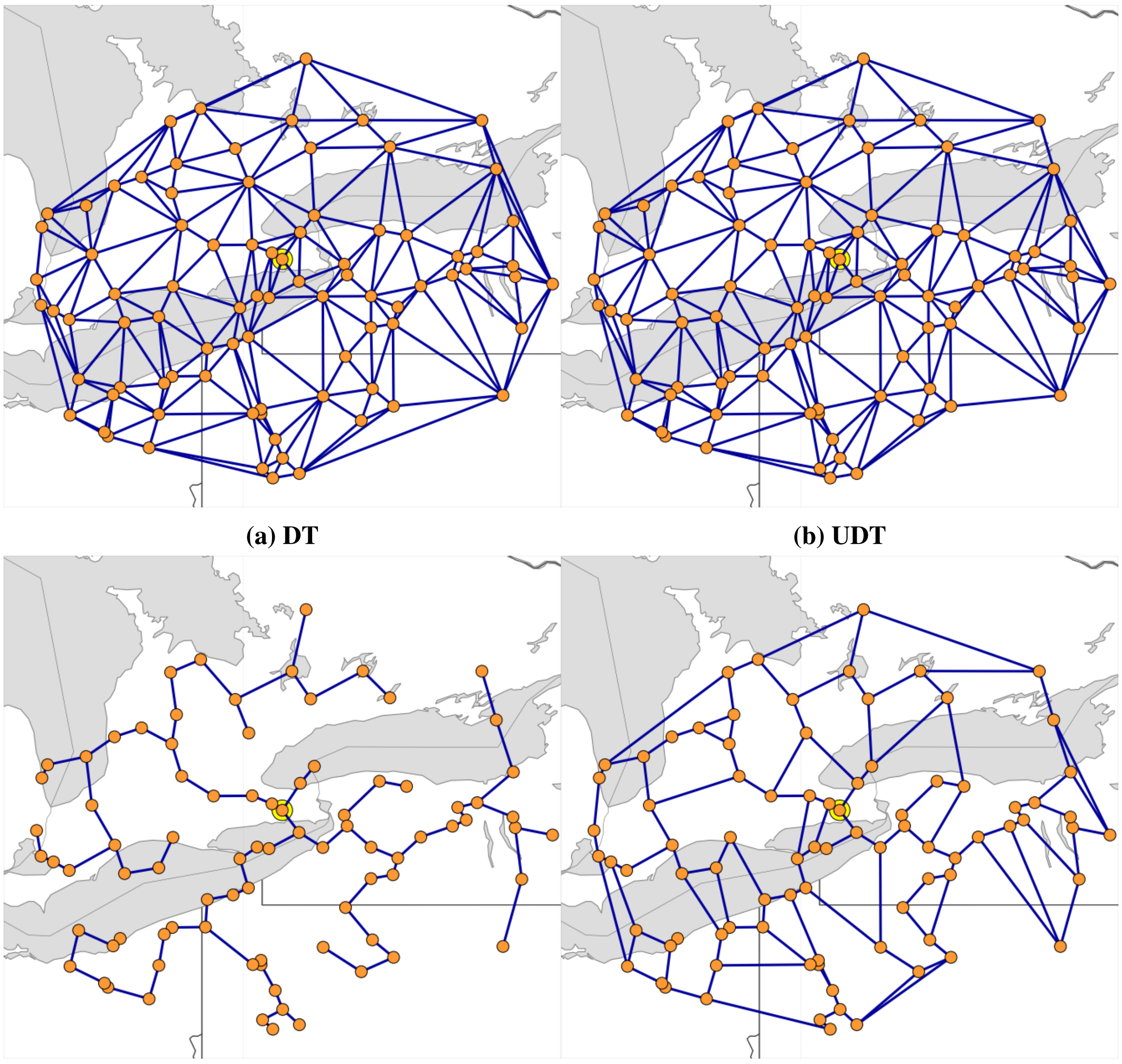
Figure 3.13 shows the resulting global topology resulting from each node independently computing its local topology using DC-DT and forming its incident connections accordingly.

## 3.6 Pitfalls

The algorithms described are effective at generating acceptable global topologies. There are many other candidate algorithms, however, that fail to form an acceptable topology using the topology generation framework. This section describes some of the pitfalls to avoid when designing the algorithms.

### 3.6.1 Adding Longest Edges

Some topology generation algorithms do not fully utilize the directional data links in the network. They do not form viable connections and instead leave links idle. For example, DCP-KRUSKAL forms a tree-like topology for which most nodes use only one or two of their three links. These algorithms form a topology but leave, unused, data links that are within range of other nodes that also have available links. The degree



**(a) DT** **(b) UDT**

**(c) K** **(d) Local DC-DT**

**Figure 3.12: The unit disk graph and the local and global graphs resulting from running the DCP-KRUSKAL algorithm**

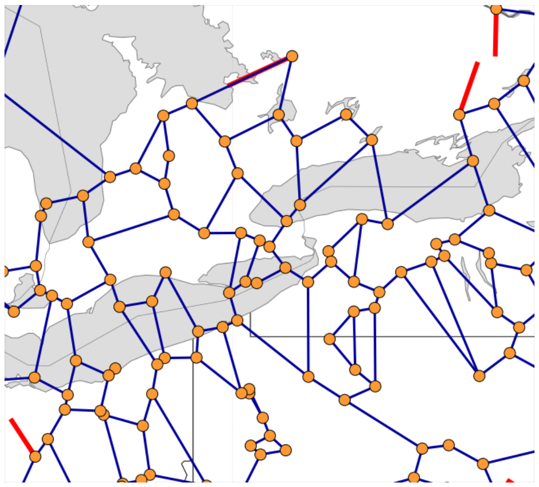


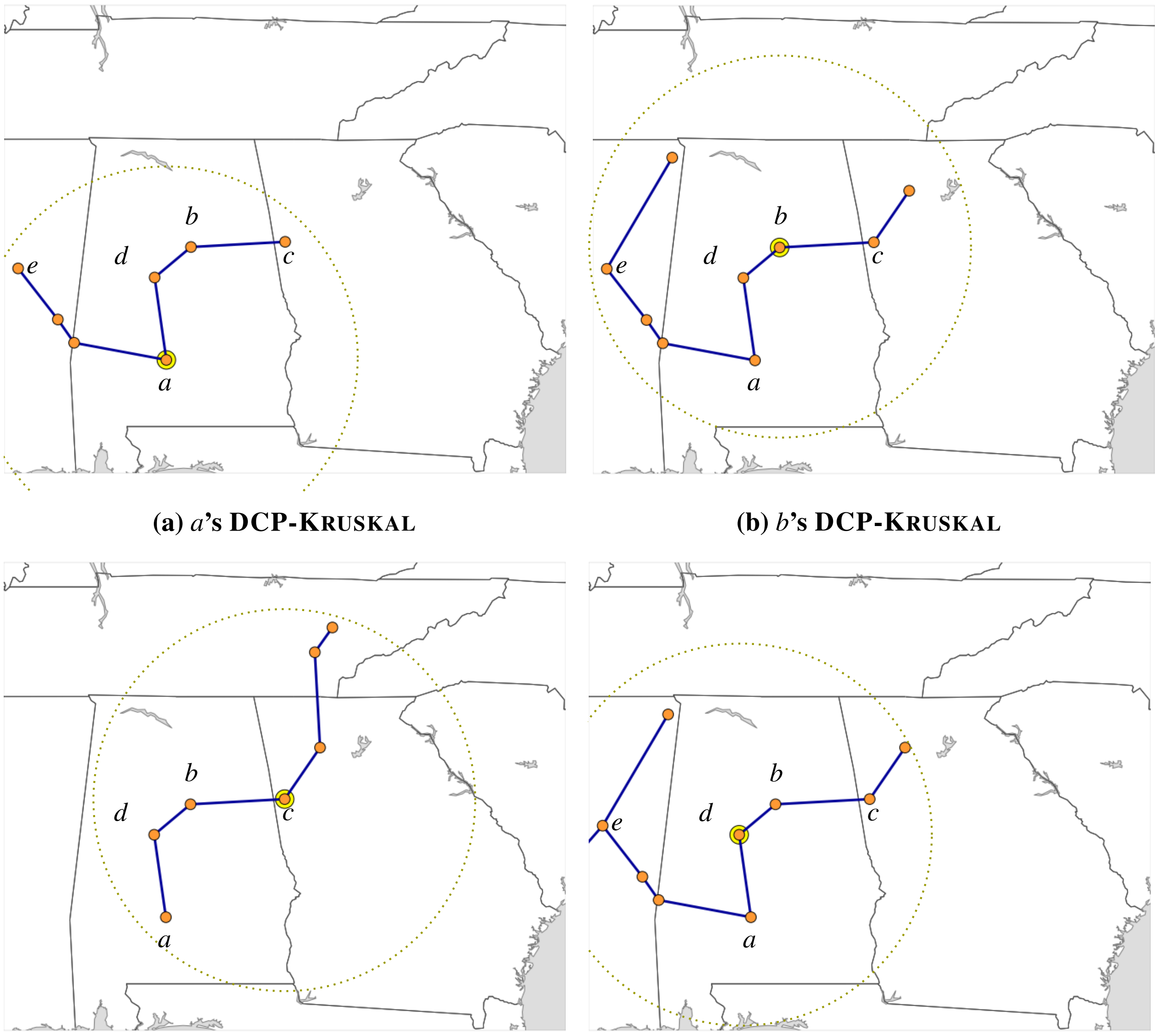
**Figure 3.13: The global DC-DT Topology.**

of both nodes in such situations is less than the number of links on the nodes. In these cases, it is tempting to augment the topology generation algorithm with a final pass that will add edges to the topology.

One such algorithm is Degree-Constrained Kruskal Plus Long (DCP-Kruskal+L). This algorithm first forms a planar degree-constrained minimum spanning tree (DCP-Kruskal) among the nodes, and then adds additional edges, longest first (+L), to the topology as long as each edge doesn't violate the degree or local planarity constraints. The spanning tree tends to ensure that all nodes in the network stay connected to one another, while the additional long edges enable shorter paths to the destination.

Unfortunately, this algorithm cannot guarantee a globally planar network. Even though each local topology is guaranteed to be planar, the global network topology, implicitly generated by the distributed topology management framework, is not guaranteed to be planar. Figures 3.14-3.16 gives an informal proof of this fact using a simple example where a topology is generated connecting simulated aircraft retracing the routes of actual aircraft whose trajectories over the Southern United States were recorded.

Figure 3.14a-d shows the local topology calculated independently by four different nodes using the DCP-KRUSKAL algorithm to connect the nodes in their community. In each case the node computing the topology is highlighted with a second (yellow) circle around the node. In each image only the nodes in the community of the node computing the local topology are plotted. For example, node *e* does not appear in Figure 3.14c, since it is not in node *c*'s community. Each node has a slightly different set of nodes (community members) of which it is aware. To this base topology, a topology generation algorithm could now add as many edges as possible (starting with the longest) until no more edges can be added without



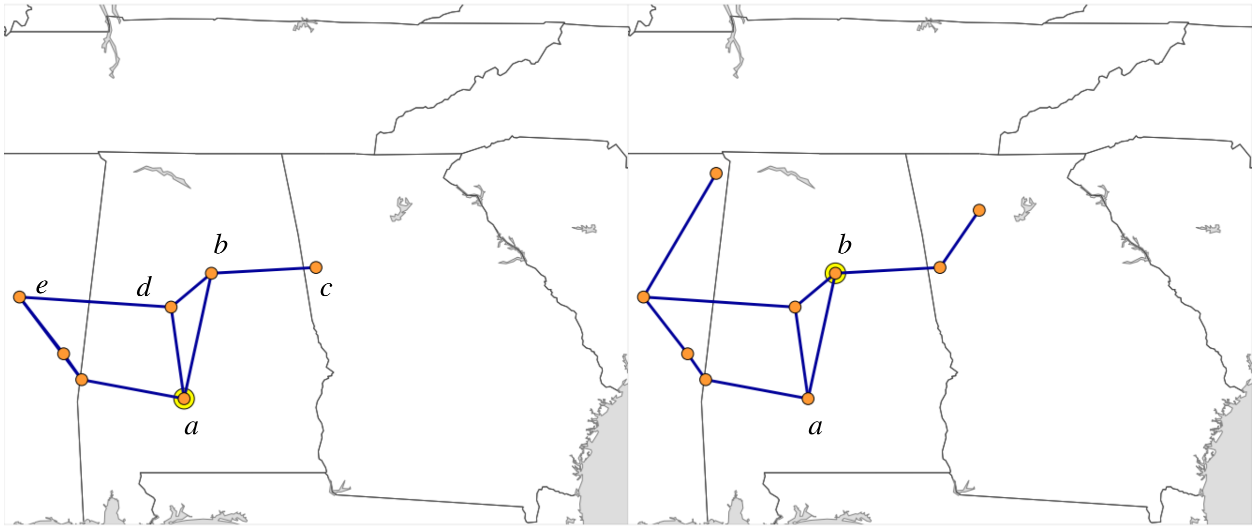
**(a)** *a*'s **DCP-KRUSKAL** **(b)** *b*'s **DCP-KRUSKAL**

**(c)** *c*'s **DCP-KRUSKAL** **(d)** *d*'s **DCP-KRUSKAL**

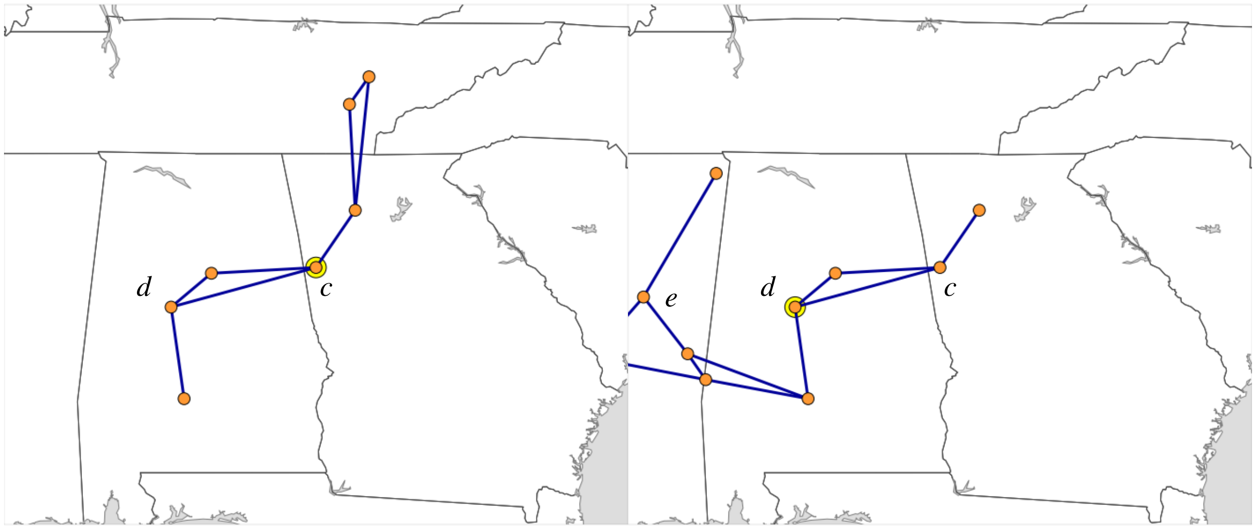
**Figure 3.14: The result nodes** *a*, *b*, *c*, **and** *d* **independently generating a local topology using the DCP-KRUSKAL algorithm given each node's community members. The dashed circle shows the extent of the community from each node's point of view.**

violating the degree or local planarity constraint. The result of adding these longest edges to each local topology are shown in Figures 3.15a-d. From node *a*'s point of view, the edge between nodes *e* and *d* is the longest potential connection in the community not already included in the DCP-KRUSKAL graph that doesn't violate degree constraints. It is added to the topology first. Notice also that assuming there are three links per node, node *d* is now using all its links in the topology. The edge between nodes *c* and *d* is the next longest, but it cannot be added because node *d* is already at maximum degree. Therefore, the edge between *a* and *b* is added instead, such that *a* will point one of its links at *b*. From node *b*'s point of view, nearly the same conditions exist, and *b* chooses to point one of its links at *a*, forming a connection.

Node *c*, however, is not aware of node *e* and therefore does not add the edge from *e* to *d* to its topology. This leaves node *d* with an available link. The longest potential connection not already in the graph is between *d* and *c*. Finally, even tough *d* knows about node *e*, its DCP-KRUSKAL graph is such that node *e* is already using all three of its links. Therefore the connection between *d* and *c* is instead established.



**(a) plus longest edges *a*** **(b) plus longest edges *b***



**(c) plus longest edges *c*** **(d) plus longest edges *d***

**Figure 3.15: The result of these four nodes adding additional edges, longest first, to the topology graph.**

Figure 3.16 shows the resulting global topology that is implicitly generated by each node computing its local topology and attempting to form its local connections as specified by the topology. The outcome is a graph that is non-planar even though every local topology was planar.

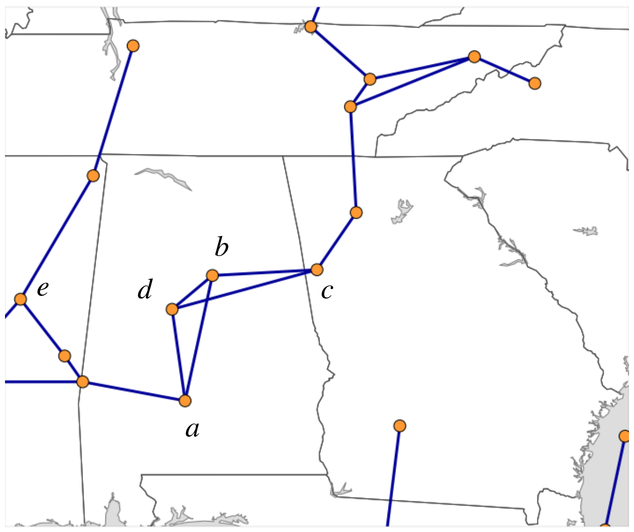
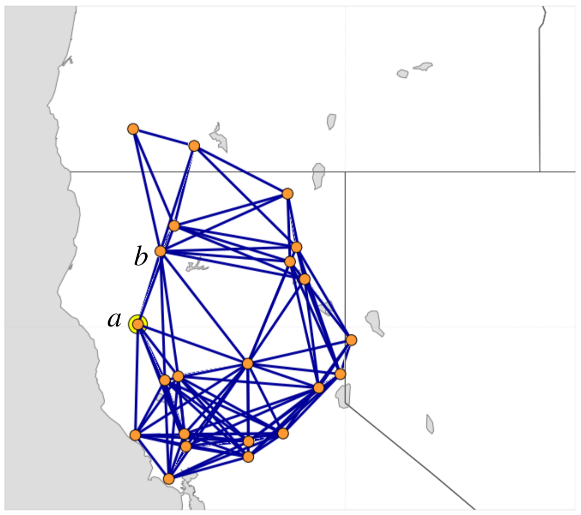


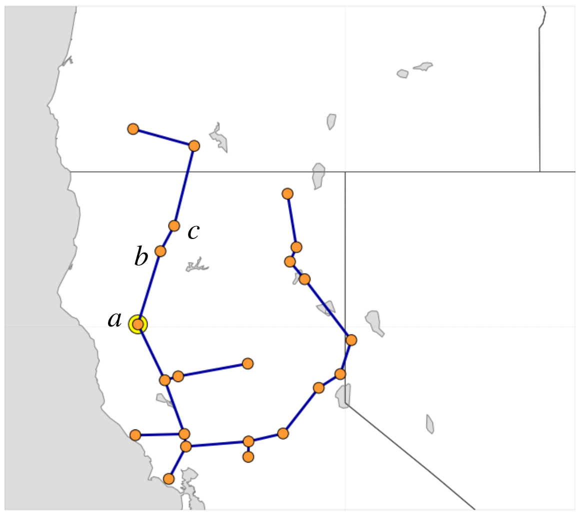
**Figure 3.16: The resulting global non-planar topology.**

### 3.6.2 Locally Planar, Globally Non-planar

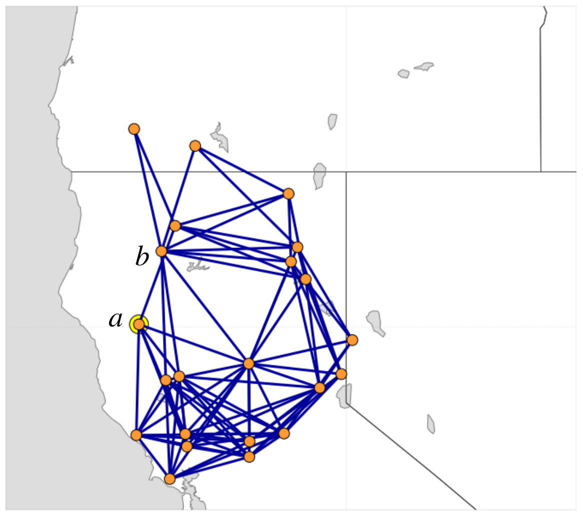
Even without adding the longest edges to a topology graph, it is possible to end up with a graph that is locally planar at every node, but globally non-planar. This pitfall is exemplified by the Degree-constrained Planar Tree-Based Reliable topology generation algorithm (DCP-TRT). As described in Section 3.5.2, DCP-TRT is a topology generation algorithm that forms a more robust topology by making multiple calls to DCP-KRUSKAL. Figure 3.17a-e shows the steps taken in computing the local topology given node *a*'s local view. First, DCP-KRUSKAL is run on the potential connections (or unit disk) graph shown in Figure 3.17a to form a degree-constrained spanning tree among the nodes in that graph (Figure 3.17b). Next, these edges are removed from the original potential connections graph, leaving the subgraph shown in Figure 3.17c. None of the edges in Figure 3.17b appear in the graph in Figure 3.17c (though some edges are hard to distinguish from other edges between nearly collinear neighbors). Next, DCP-KRUSKAL is called again on this resulting subgraph, yielding a slightly more awkward-looking degree-constrained spanning tree as seen in Figure 3.17d. Finally, the edges in Figures 3.17b and 3.17d are combined into the single local graph shown in Figure 3.17e. Notice that the final local graph includes an edge between *a* and *c*. Figure 3.18a-e



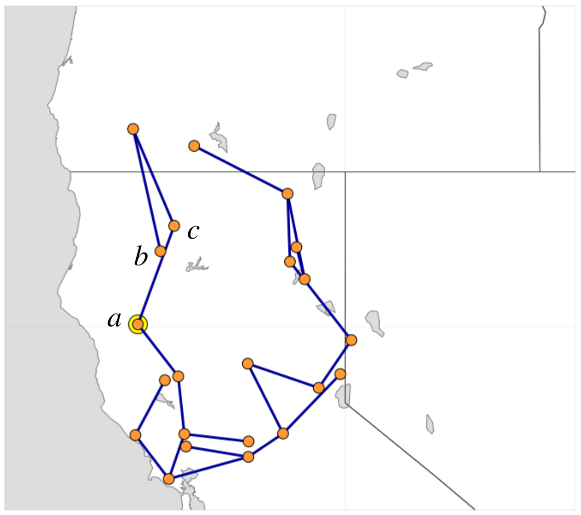
(a) Unit Disk Graph



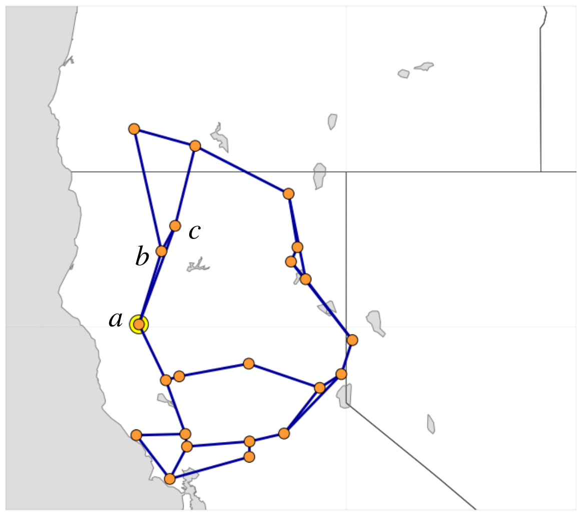
(b) DCP-KRUSKAL 1



(c) edges removed

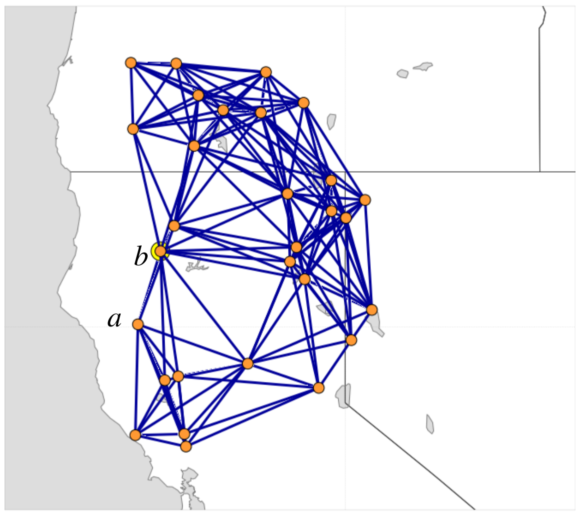


(d) DCP-KRUSKAL 2

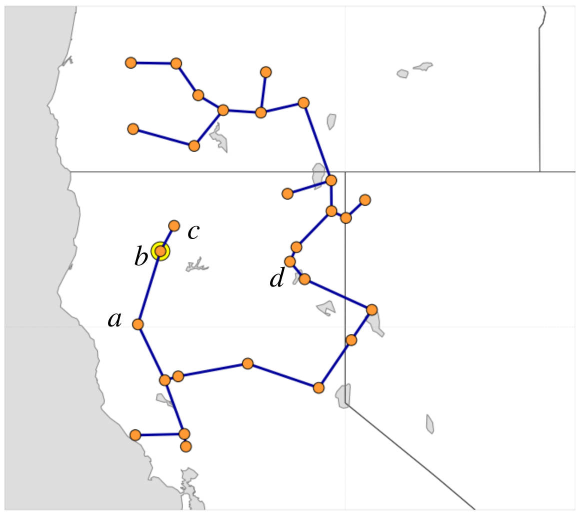


(e) local DCP-TRT

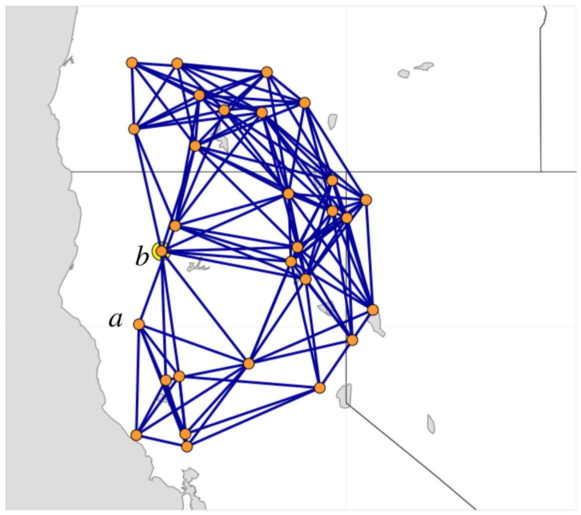
**Figure 3.17: Computation of the local DCP-TRT topology at node $a$.**



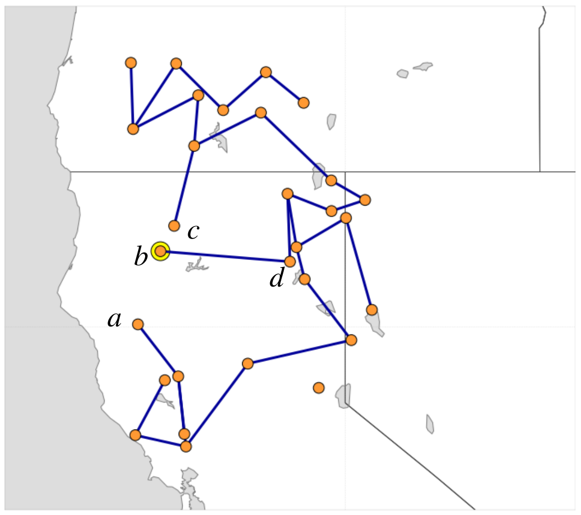
(a) Unit Disk Graph



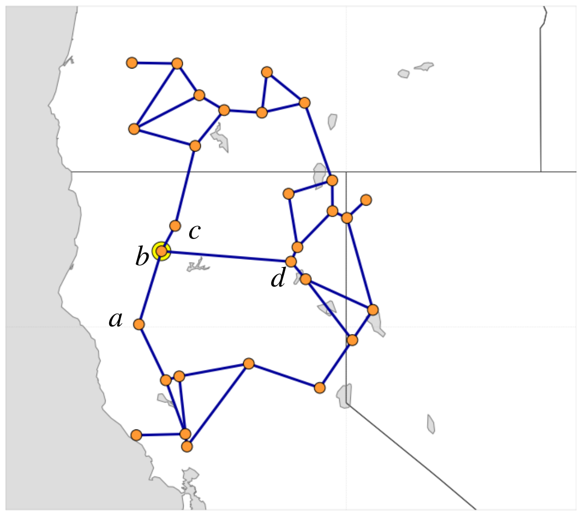
(b) DCP-KRUSKAL 1



(c) edges removed



(d) DCP-KRUSKAL 2



(e) local DCP-TRT

Figure 3.18: The computation of the local DCP-TRT topology at node *b*.

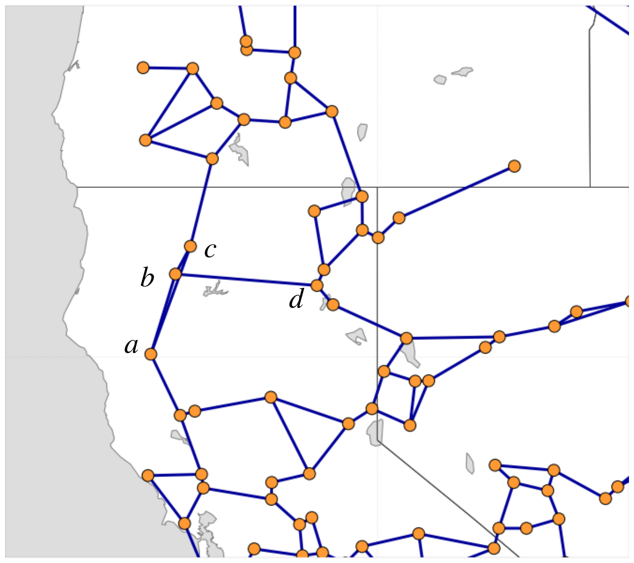


**Figure 3.19: The resulting global non-planar topology using DCP-TRT.**

shows a similar progression in the computation of the DCP-KRUSKAL topology graph from node *b*'s perspective. Notice that since the original spanning tree (3.18b) includes edges *ab* and *bc*, and these cannot be a part of the spanning tree in Figure 3.18d. The shortest edge that can be added to incorporate *b* into the degree-constrained spanning tree is the edge *bd*. Note, however, that this edge crosses the edge *ac* that is included in the local topologies for both *a* and *c*. This leads to the global non-planar topology shown in Figure 3.19 (edges *ac* and *bd* cross).

## 3.7 Topology Generation Algorithm Comparisons

### 3.7.1 Aircraft Trajectory Data Processing

The bulk of the simulations and experiments for this work use real aircraft trajectory data. Before this data can be used some filtering and processing is necessary. The raw data is a list of entries that include a call sign (such as DAL213 for Delta Airlines flight number 213), a timestamp, and position information. First, these entries must be grouped into flights. This can be tricky, since there are cases where data is missing from a specific call sign for a couple of minutes, and there are also cases where only a couple of minutes separate the landing of a flight with a certain call sign, and the takeoff of a different aircraft at a different airport with the same call sign.

For most simulations in this work, we constrain the nodes to be within the boundary shown in Figure 3.20. The boundary is approximately a 200km offset from the border of the continental United States.

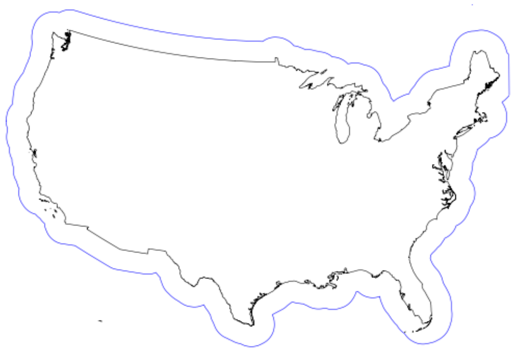


**Figure 3.20: A boundary ˜200 km offset from the continental United States border and coastline.**

Constraining the trajectories in this way enables simulations to be concentrated on a well-known geographic region, while avoiding situations where an aircraft would need to join and leave the network multiple times as it flew along a coastline.

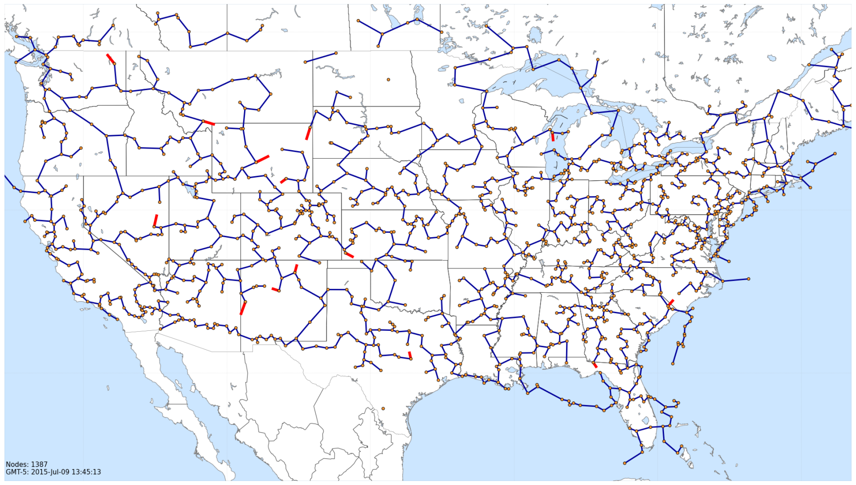
In this section, the topology alogrithms introduced above are compared both visually, empirically.

### 3.7.2 Visual Comparison

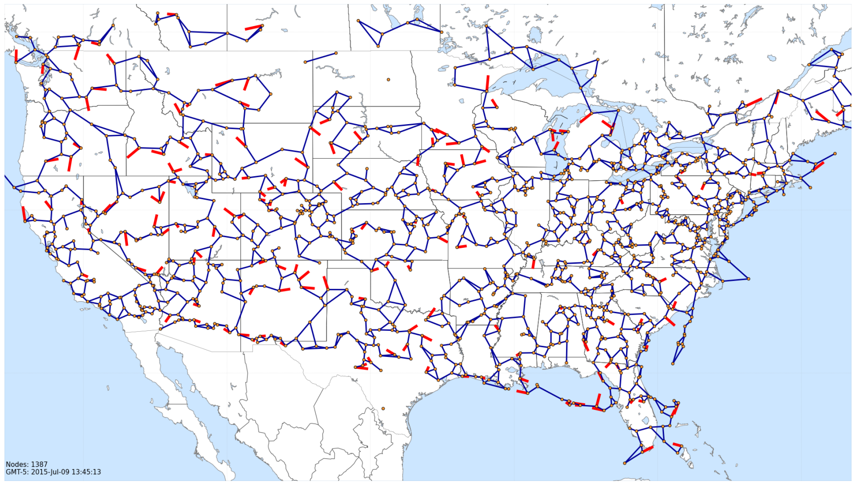
This section presents a visual comparison of the global topologies of a continental-scale networks. Figure 3.21 displays the result of applying each topology generation algorithm on a set of 1387 aircraft spread across the United States at 13:45 CT on July 9, 2015.

Notice that for DCP-KRUSKAL (Figure 3.21a), although every local topology is a tree, the global topology is not a tree but contains several very large faces. This happens because each node has a limited view of the world when determining its local topology, and cannot guarantee the global topology is a tree. Notice also, that DCP-KRUSKAL has fewer edges in its topology compared with the other topologies, and relatively few one-way connections (shown as thick (red) edges that terminate at the midpoint). The topology also has many nodes that only have a single active link. These leaf nodes act like "dead ends" when the topology is viewed as a maze that the packet is trying to solve to get to the destination. A greater number of "dead ends" can make geographic routing more difficult. Finally, DCP-KRUSKAL generates a topology that is not very robust to link outages, as there are few paths between any two points.

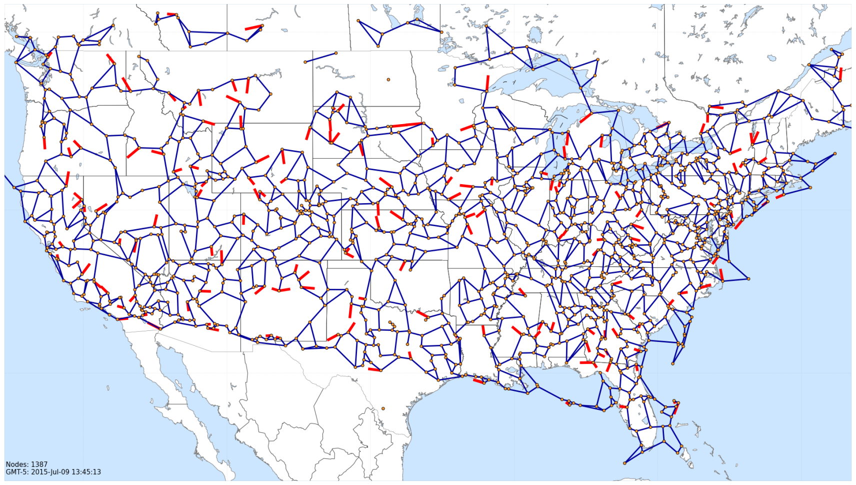
As seen in Figure 3.21b, DCP-TRT has more redundancy, more faces, and more one-way connections than DCP-KRUSKAL. This algorithm tends to form topologies containing many groups of nodes that are well connected, with multiple paths between one another, and longer connections that join these groups together. DCP-TRT is more robust to connection failures, and there are multiple paths between most pairs



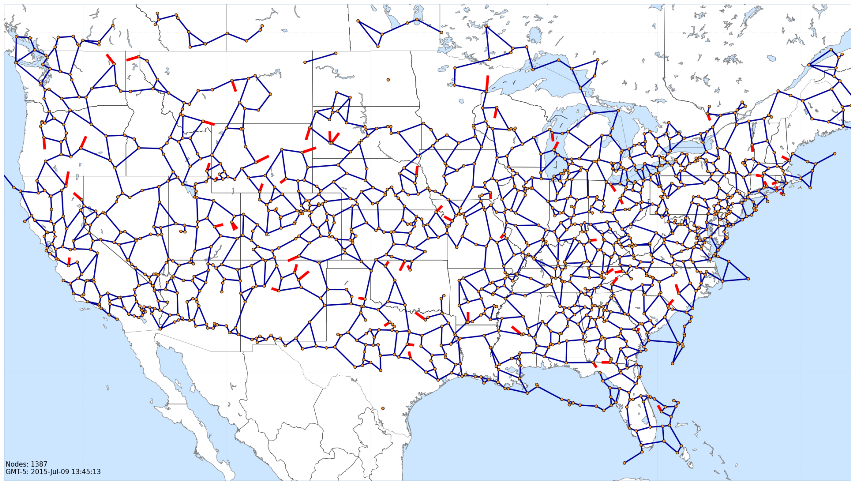
**(a) Large Global DCP-Kruskal**



**(b) Large Global DCP-TRT**



**(c) Large Global DC-DT**



**(d) Large Global DC-GG**

**Figure 3.21: A visual comparison of the global continetal-scale topology resulting from each algorithm.**

of nodes. Notice also that the one-way connections are more prevalent in the areas with lower densities (more in the west than the east, for example).

Figure 3.21c displays the global topology resulting from using the DC-DT local topology generation algorithm. Visual inspection reveals that DC-DT generates a topology with more faces than DCP-KRUSKAL or DCP-TRT. Further, it can be seen that there are a greater number of paths between any two nodes in the topology generated by DC-DT than in the topology generated by DCP-KURSKAL (more robustness), but DC-DT's topology has a higher number of one-way connections.

Finally, Figure 3.21d shows the result of using the DC-GG algorithm on the same set of nodes. Like DC-DT, visual inspection indicates that DC-GG produces a topology with many smaller faces and in general more path options (robustness) than DCP-KRUSKAL. It appears to have, however, fewer one-way connections than DC-DT.

Note also, that none of the topologies are able to connect the node flying over central North Dakota (top middle of the figures). This is not a deficiency of the topology generation algorithms, but rather a physical limitation. This node is further than the maximum transmission range (200 km) from any other node and is, therefore, unable to connect to the network regardless of the topology generation algorithm. Similarly, the two nodes flying near the Canadian border in northern North Dakota and Montana are physically unable to connect to any other nodes due to link range restrictions. They are, however, able to connect to one another, as specified by every topology. Nodes near the boundary of the network or in areas with lower densities of nodes are more prone to being disconnected from the largest connected component in the topology. In a real-world network, having ground stations at these lower density locations would alleviate this problem.

### 3.7.3 Attribute Comparisons

To gain an understanding of the performance of these algorithms a detailed empirical analysis of several key properties or attributes of the topology graphs for many sets of nodes of various densities is now presented. For each experiment, a date and time are selected at random within the 31-day time period of July 2015. Aircraft (excluding general aviation flights) flying above 10,000 feet and within ˜200km of the continental United States at the selected time are then randomly selected until a target mean density value is reached. If there are not enough aircraft (nodes) available at the chosen time, a new random time is selected and the process repeats. Each node (in the set of potentially thousands of nodes) then determines which nodes are in its community (within 288 km) and the potential connections graph (unit disk graph) for this set of nodes is determined. The nodes then independently run each of the candidate topology generation algorithms on the potential connections graph to obtain a local topology graph at this node for each algorithm. Once all of the nodes have computed their local topology graph using a given algorithm, all the local graphs are combined together, simulating each node forming the connections of which it is a part in its local topology graph. The result is a global topology graph for each candidate algorithm, that was generated by local decisions at each node.

Continuing with the experiment, after the global topology is determined, 100 pairs of nodes are randomly selected, and the shortest path (in hops) between them (if a path exists) is measured. Finally, the positions of the same set of nodes one second later are determined and these positions are used to re-compute all the topologies and analyze another 100 shortest paths. We run this experiment 775 times (25 random samples per day) so that each point is the mean of 1550 topologies of that density (2 topologies, 1 second apart per experiment) or 155,000 paths through topologies of that density.

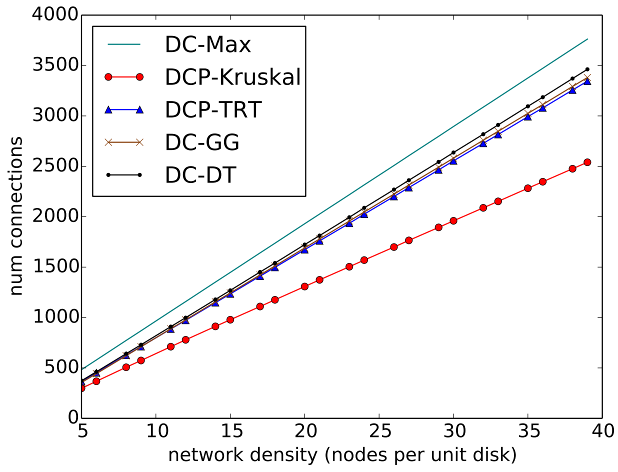
The following sections analyze several attributes of the computed topologies and paths.

#### 3.7.3.1 Number of Connections or Edges

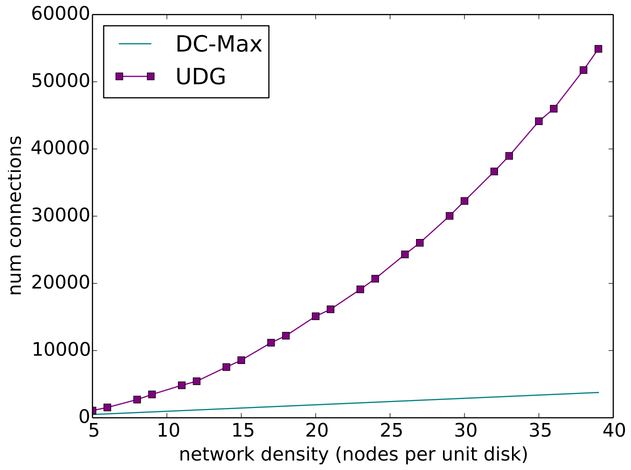
First, the number of connections in each algorithm's topologies are analyzed. Figure 3.22a shows how the number of connections increases as the density increases. In the figure, the x-axis represents the density of the simulated networks in nodes per unit disk (see Section 3.1.2). For these various densities, the mean number of connections (edges in the topology graph) included in the global topologies for the various algorithms are plotted. Also, for comparison, a line representing the theoretical maximum number of connections given the degree constraint is also included (DC-Max). This is the theoretical maximum number of connections possible if every node was able to connect each of its directional links to another node that also has an unused directional link. The planarity constraint restricts link usage to some even lower upper-bound.

Remember that a density of value of 10 nodes per unit disk (disk with a 200 km radius) in the continental United States equates to about 643 nodes, while a density value of 35 nodes per unit disk equates to about 2250 nodes. As the number of nodes increases, there is a linear increase in the number of edges in the topologies. DC-DT consistently produces topologies with the most edges. DCP-Kruskal, as would be expected, yields topologies with the fewest edges because of its spanning tree-like topologies. On average DC-GG includes slightly fewer edges in its topologies than DC-DT. This make sense, since the Gabriel Graph is a subgraph of Delaunay Triangulation. The number of edges in the topology graph is a helpful performance metric because it correlates with the number of hops needed to route a packet through the topology and with the robustness of the topology.

Figure 3.22b compares the number of connections in the unit disk graph with the degree-constrained maximum also shown in Figure 3.22a. Note how the degree constraint severely limits the number of connections, when compared to the actual number of potential connections. The UDG essentially represents the connectivity of an omnidirectional link with the same range as the directional link. The disparity in the number of connections helps to illustrate the limiting factor imposed by the use of directional links.



**(a) The mean number of connections (edges) in each topology.**



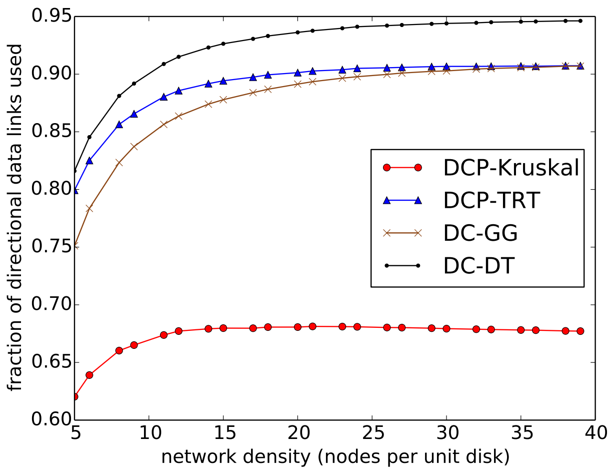
**(b) A comparison of the mean number of connections in the Unit Disk Graph with the degree constrained maximum number.**

**Figure 3.22: An analysis of the number of connections.**

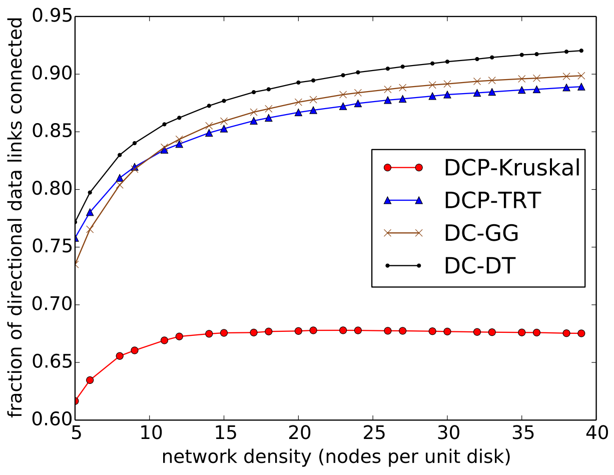
#### 3.7.3.2 Directional Data Link Usage

A good topology generation algorithm will generally try to avoid allowing links to go unused. However, an algorithm can make poor use of a link by just pointing it randomly at any neighbor. Two metrics, then, are important when considering link usage. First, the number of links used (commanded to point) by the topology generation algorithm, and second, the number of links actually connected by the algorithm (discounting one-way connections). The first metric allows one to see how many links an algorithm tries to use, and the second metric gives an indication of how many of those links are actually able to form connections. Figure 3.23a shows the fraction of directional data links used by each of the topology generation algorithms across networks of various densities. The numbers reported here include directional data links that are a part of a one-way connection. DCP-KRUSKAL uses less than 70% of the data links available, which is expected since it is only seeking to form a spanning tree among the nodes. The remaining algorithms all use more than 85% of the data links for networks with densities above about 10 nodes per unit disk. DC-DT makes use of the highest percentage of directional data links, however, it should be noted that DC-DT also creates the highest percentage of one-way connections (as detailed in the following section).

Figure 3.23b plots the same information, but without including one-way connections. In other words, it plots the fraction of directional data links actually connected by each algorithm. Notice that despite generating the highest percentage of one-way connections, DC-DT always connects the highest fraction of data links. Also, note that DC-GG outperforms DCP-TRT for all but the lowest densities when one-way connections are excluded.



**(a) The fraction of directional data links used by each algorithm.**



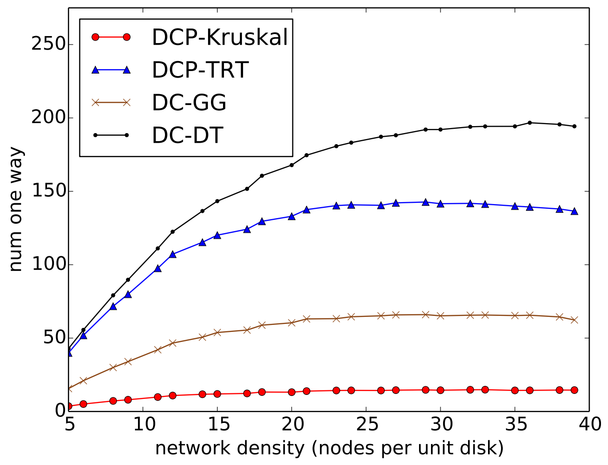
**(b) The fraction of directional data links actually connected by each algorithm.**

**Figure 3.23: An analysis of the data links used and connected by each algorithm.**

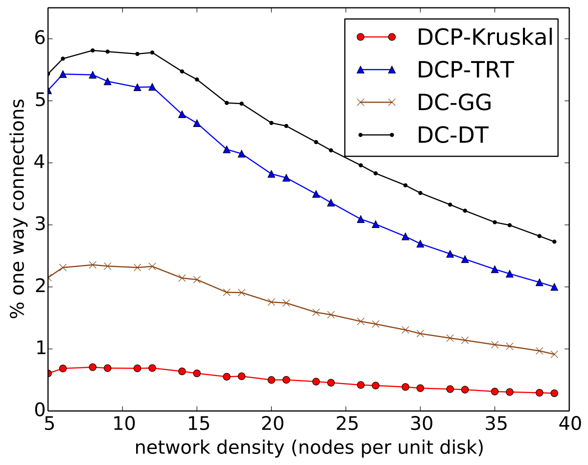
#### 3.7.3.3 One-Way Connections

Figure 3.24a shows the raw number of one-way connections in the topology graphs produced by the various algorithms. As the density increases, the number of one-way connections levels off, and in some cases even decreases. DC-GG has substantially fewer one-way connections that DC-DT. When DC-DT adds edges, longest first, it is essentially taking a risk. It hopes that the node at the other end of a long connection will have a view of the world close enough its own view, that it will come to the same conclusion about the need for the connection, and point back to form the connection. The benefit of these long edges is shorter paths, on average, but the cost is a higher incidence of one-way connections. DC-GG has fewer edges to consider for addition, and thus fewer long edges, yielding fewer one-way connections, but also longer average shortest paths than DC-DT.

Figure 3.24b shows the percentage of one-way connections among the directional data links "used" by the algorithm, where any directional data link that is commanded to point is considered "used" whether or not a connection is formed. At a density of 32 nodes per unit disk (~2090 aircraft above the United States), the occurrence of one-way connections is a little over three percent for DC-DT. This equates to a data link forming a one-way connection for about 2 minutes (119.74 seconds) of each hour. Compare that with 91.18 seconds for DCP-TRT, 42.29 seconds for DC-GG, and 12.70 seconds for DCP-KRUSKAL.



**(a) The number of one-way connections.**



**(b) The percentage of one-way connections among used data links.**

**Figure 3.24: An analysis of one-way connections.**

#### 3.7.3.4 Connection Lengths

Figure 3.25 plots the mean length of a connection using each of the algorithms across a variety of densities. For every algorithm except DC-DT, the mean length is always below half the maximum length of 200 kilometers, even for the lowest densities. DC-DT produces mean connection lengths just over half the maximum length for only the lowest density tested. All other mean connection lengths measured were also under 100 km (half the maximum transmission range). The mean connection length decreases with increasing density because maintaining the degree and planarity constraints requires that new nodes added to the network must generally join by breaking an existing, longer, connection, replacing it with two shorter connections, thus decreasing the mean connection length.

DC-DT produces the longest connections on average, followed by DC-GG and the DCP-TRT. Since Gabriel Graph is a subgraph of Delaunay Triangulation, DC-GG will, in general, have fewer edges to consider for addition to the spanning tree than DC-DT does. With fewer edges to consider it is natural to assume that when adding edges, longest first, DC-GG will add fewer long edges, on average, than DC-DT. DCP-Kruskal is an approximation of the minimum spanning tree, and is therefore near the lower bound for the mean connection length required to span all the nodes at each density.

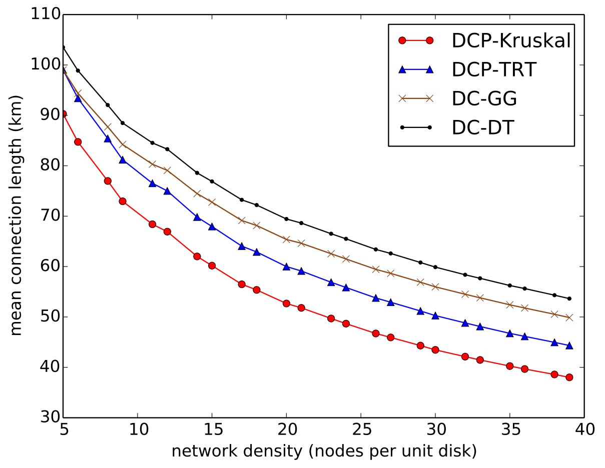
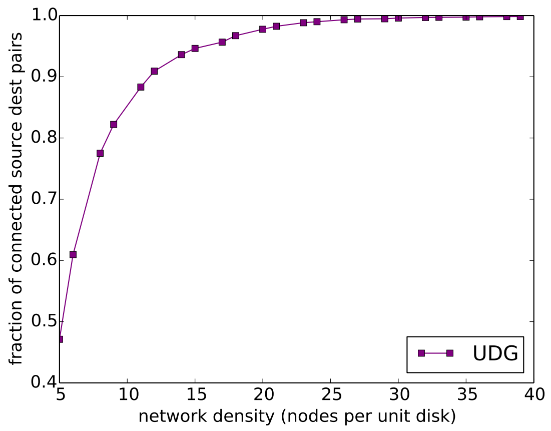


**Figure 3.25: The mean length of a connection across algorithms.**

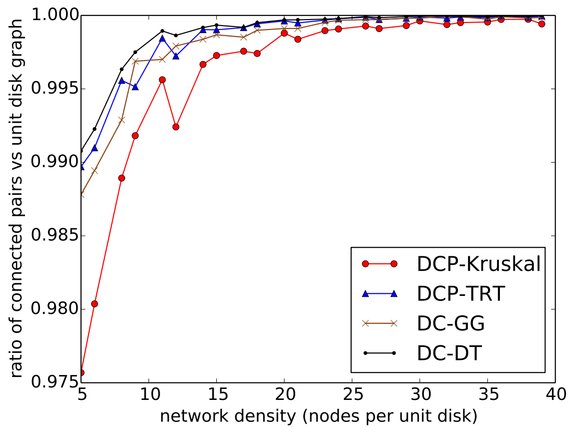
#### 3.7.3.5 Connected Pairs

Figure 3.26a shows the mean fraction of pairs of nodes tested, between which there exists a path in the unit disk graph. Note that on this plot the y-axis begins at 0.4. The curve closely approximates the upper-bound for the ratio of connected pairs over the given densities. No algorithm can connect pairs of nodes better than the unit disk graph without violating the maximum communication range constraint. For densities above 24 nodes per unit disk (~1543 aircraft across the United States) the ratio of connected pairs of nodes stays above .99 for the unit disk graph. All of the algorithms' curves match or come very close to this curve, such that it is impossible to compare them on this plot. Next, the nearness of the other algorithms' curves to that of the unit disk graph is analyzed.

Figure 3.26b essentially shows how close each algorithm comes to meeting the upper-bound curve shown in Figure 3.26a. Specifically, Figure 3.26b plots the mean ratio of the number of sampled node pairs connected with a path in the given topology to the number of node pairs for which a path exists in the unit disk graph. Notice that on this plot that y-axis begins at 0.975. A perfect topology generation algorithm would always provide a path when one exists (giving a mean ratio of 1.0), however, the added degree constraint and planarity constraint make this difficult (if not impossible) to achieve. Note that above a density of about 9 nodes per unit disk (˜579 aircraft), the number of nodes disconnected is less than 1% for all of the algorithms. DCP-KRUSKAL performs worst since its focus is on short edges and the resulting low robustness causes it to miss opportunities to maintain connectivity with nodes that are otherwise disconnected. DC-GG performs slightly worse than the DCP-TRT, and DC-DT is the obvious best choice for lower densities. DC-DT also performs consistently as good as, and often better than, the other algorithms at the higher densities as well, though it is difficult to see that in this plot.



**(a) The mean fraction of pairs of nodes connected by a path in the Unit Disk Graph.**



**(b) The mean ratio of the number of sampled node pairs connected with the given topology generation algorithm to the number connected in the unit disk graph.**

**Figure 3.26: An analysis of the connected node pair metrics.**

### 3.7.3.6 Computation Time

Figure 3.27 shows the mean computation time per node for each local topology computed. The mean times were derived by measuring the total time required to generate all of the local topologies for each network tested, and then dividing by the number of nodes in that network. As the density increases, the computation also increases substantially. The exact cause of the change in the trend for all algorithms at about a density of about 30 nodes per unit disk is unknown, but it is likely caused by the fact that as the density increases there are fewer input data files that have random samples of the required density. Note also, that the implementation of Gabriel Graph used here is a non-optimized version, whereas DC-DT uses an optimized library method to compute the Delaunay Triangulation. It is expected that the execution cost of DC-GG, once optimized, would more closely match that of DC-DT, since DC-GG is a sub-graph of Delaunay Triangulation, and can be quickly calculated given the Delaunay Triangulation of a set of nodes. For a density of 30 nodes per unit disk (˜1930 nodes across the United States), all of the algorithms are able to compute a topology in less than 22 milliseconds on average[1]. DC-DT has the fastest computation time. All of the computation times, however, are sufficiently low as to enable multiple current and future topologies to be computed during each iteration, if desired. Computing future topologies based on positions extrapolated several seconds into the future could enable topology changes to be predicted and anticipated (see Section6.3.1.1 for more on this future work).

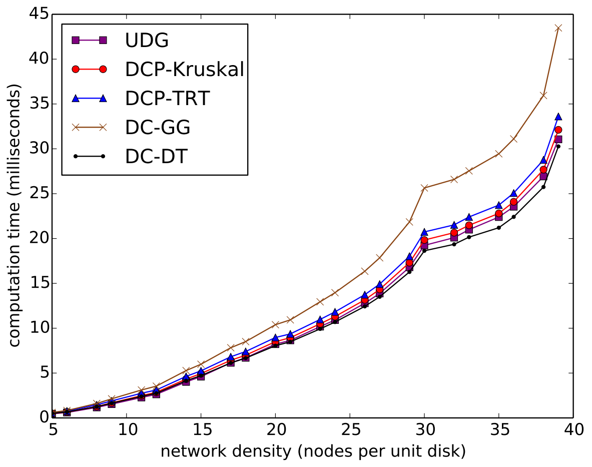


**Figure 3.27: The per-node computation times for the various algorithms.**

### 3.7.3.7 Dynamicity

Figure 3.28 plots the percentage of edges that change (are added or removed) when the topology is recomputed after 1 simulated second. We call this metric the dynamicity of the network topology. It is anticipated that this metric may be important once the time required to physically slew antenna and establish connections is accounted for in the simulation, whereas for the simulations in this work it is assumed that links can instantly re-point and connect. DCP-TRT has the highest level of dynamicity. This is another

[1] on an Intel Xeon CPU (E5-2637 v2) at 3.50GHz

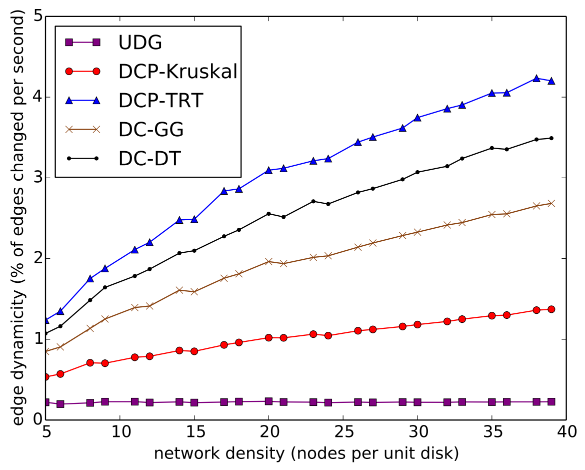


**Figure 3.28: The mean ratio of the number of edges added or removed in consecutive topologies, to the number of edges in the topology.**

reason to prefer the other algorithms over DCP-TRT. Notice also, DC-GG has lower edge dynamicity than DC-DT.

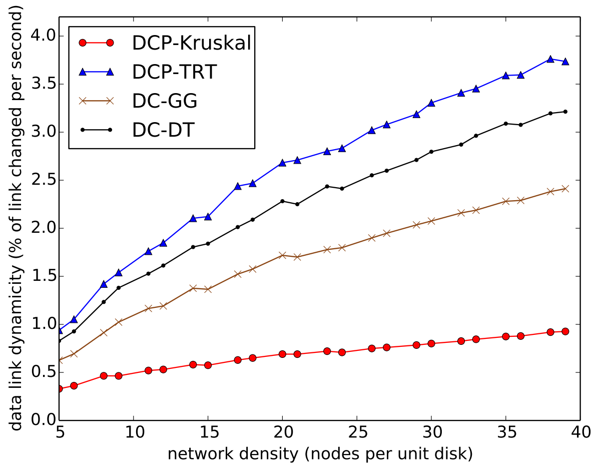


**Figure 3.29: An analysis of link dynamicity.**

In Figure 3.29 the data link dynamicity is plotted for each of the algorithms. The data link dynamicity is the percentage of data links in the network starting a new connection or terminating a connection during an average second. Notice that a data link that re-points is included twice in this metric; once for terminating a connection and once for starting a new connection. For DC-DT at a density of 30 nodes per unit disk (˜1930 aircraft) the data link dynamicity is 2.8%. This means that using this algorithm at this density, a data link would be expected to connect or disconnect approximately once every 36 seconds (1/.028). However, if good link utilization is assumed, then most link changes will be re-pointing commands, such that one connection will end and another will start, as opposed to one connection just ending and the link not being used during the next iteration. This assumption would nearly cut the percentage in half and extend the time between re-pointing events to over a minute. Notice that a small change in the dynamicity percentage can produce a large change in the estimated average link lifetimes. For DC-GG, at a density of 30 nodes per unit disk, the link dynamicity is 2.08% (.72% lower than DC-DT). Using this algorithm one would expect the time between re-pointing events to be about a minute and a half (2*(1/.0208) = 96 seconds), nearly 35% longer than when using DC-DT. If link establishment times were on the order of 1 second, these link lifetime numbers would seem reasonable. A link could form and then on average send data for a minute or more before re-connecting. Since the dynamicity is growing with density, however, this is likely one attribute that will ultimately limit the scalability of the system, since at extreme densities the may be too much churn in the link pointing.

#### 3.7.3.8 Shortest Path Hop Stretch

Figure 3.30 plots the ratio of the length (in hops) of the shortest path between the random pairs of nodes simulated for each algorithm to the shortest possible path (in the unit disk graph). From our experience the number of hops can have a significant impact on the routing efficiency of the network. Notice that DC-DT consistently forms the shortest paths in terms of numbers of hops. The intuition about why this ratio continues to rise as the density increases is that because density, node degree, and maximum transmission

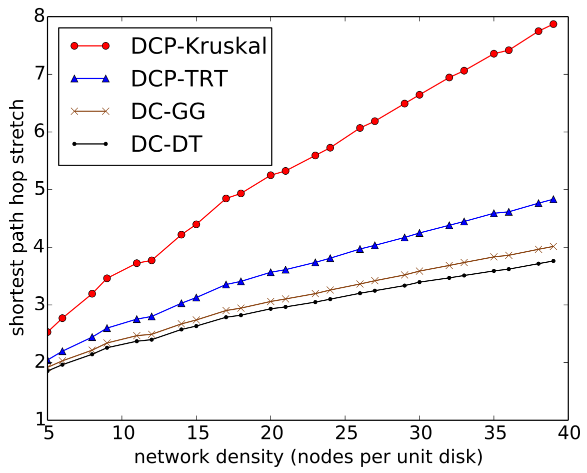


**Figure 3.30: The ratio of the number of hops in the shortest path for each algorithm's computed topology to the number of hops in the shortest path in the unit disk graph.**

range are held constant, new nodes that join the network often have no choice but to break an existing connection to join. This essentially adds an extra hop to any path that used the initial connection. Note also from the graph that DCP-KRUSKAL performs the worst given that it generates a tree-like global graph, with minimal, if any, redundancy.

#### 3.7.3.9 Overall Assessment

There is no absolute overall best algorithm. While DC-DT forms the highest numbers of connections, makes use of the most data links, has the longest links, the highest ratio of connected node pairs, and the fastest computation time, it also has a high dynamicity and produces a lot of one-way connections. DC-GG also performs relatively well for most of the attributes and has a smaller dynamicity. Some these differences are explained by the fact that DC-GG generally generates fewer edges than DC-DT. DC-TRT suffers from high dynamicity and longer shortest paths, though it has connectivity that approaches that of DC-DT.

## 3.8 Simulation Results and Evaluation

We now present the results of simulating each of the topology generation algorithms in a network of hundreds of aircraft flying real routes. Using ns-3 [RH10] we simulate 1000 simultaneous low-rate flows through our simulated airborne network that grows as aircraft join over the course of one hour, to over 500 aircraft. These nodes were selected to ensure that they were connectable throughout the simulation by all of the studied topology generation algorithms. For the simulation, we assume that links can instantaneously re-point, and that nodes have a synchronized time source. The Topology Aware Geographic Routing protocol (TAG) [NAJ16], that is able to route within these highly dynamic network topologies, is used to route the packets through the managed topology. A topology update interval, and position exchange interval of once per second is assumed. Further a 200 km maximum link transmission range and a 288 km position exchange range are used, and each node is assumed to have 3 directional data links. Low-rate UDP flows between 1000 pairs of nodes are simulated for an hour, transmitting a total of 1.87 Gigabytes of data.

**TABLE 3.1: The mean number of hops for 1,000 low-rate UDP flows**

| | mean hops | median hops |
|---|---|---|
| DC-DT | 24.039 | 21 |
| DC-GG | 23.291 | 20 |
| DCP-TRT | 31.309 | 26 |
| DCP-KRUSKAL | 56.585 | 42 |

Table 3.1 lists the mean and median numbers of hops traveled by the 3 million packets routed during each of the 1-hour simulations (one per algorithm). We find that surprisingly DC-GG performs best with the lowest number of hops. This is not consistent with our analysis above and has to do with the fact that the geographic routing protocol (TAG) does not always find the shortest path. Therefore DC-DT may generate the shortest paths, but the geographic routing protocol has slightly better performance when topologies are generated using DC-GG.

# CHAPTER 4: TOPOLOGY-AWARE GEOGRAPHIC ROUTING[1]

This chapter describes the Topology-Aware Geographic Routing Protocol (TAG), a new geographic routing protocol suitable for dependably routing packets through a large-scale, highly-dynamic network topology. First, an overview of the protocol is given. Next, the protocol's header and its contents are presented, and a base version of TAG is detailed. Optimizations of this base protocol are then presented as well as implementation details. Finally, the results of experiments conducted with the protocol in realistic airborne network simulations are presented and analyzed.

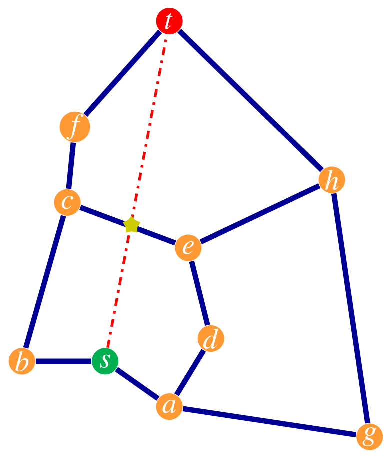
## 4.1 TAG Overview

This section gives a high-level overview of the Topology-Aware Geographic Routing Protocol (TAG). Details about each of the components and optimizations are included in the remaining sections.

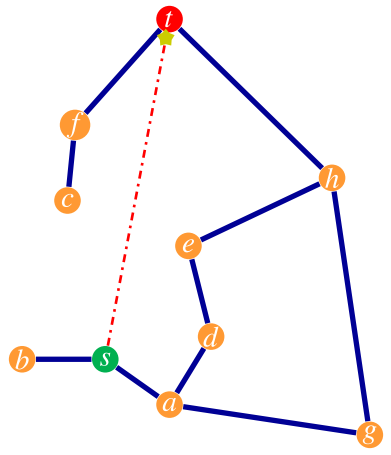
At its base, TAG is a greedy-face-greedy geographic routing protocol (see Section 2.1.6.4). As with most other geographic routing protocols, TAG requires that a small additional header containing minimal state information be added to each packet. Among other values, this extra header in each packet contains a mode field that specifies the routing mode. Each new packet introduced to the network starts in GREEDY mode. In this mode, each forwarding node attempts to greedily forward the packet to whichever directly connected neighbor is nearest the destination, in terms of linear (straight line) distance. This continues until either the packet is delivered, or a local minimum is encountered. A local minimum occurs at a node that has no directly connected neighbor that is closer to the destination than it is.

At every local minimum, there exists a face (internal or external) formed by the edges of the planar 2-dimensional network topology that is incident to the local minimum node and is intersected by the line segment between the local minimum node and the destination. Figure 4.1 shows two examples of this, one for an internal face and the other for an external face where the local minimum is the source node $s$ and the destination is $t$. In either case, the incident face occludes the packet's advance towards the destination.

[1]The contents of this chapter includes material from an article published in the proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). The citation of this related original source is as follows: B. Newton, J. Aikat, and K. Jeffay, "Geographic Routing in Extreme-Scale Highly-Dynamic Mobile Ad hoc Networks", in Proceedings of IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), London, September 2016.



**(a) If the destination ($t$) is reachable and the topology is a planar static graph, there is a point (not necessarily a node) on the boundary of the face incident to the face start node ($s$) and intersected by the line segment $st$, that is closer to the destination than $s$. This figure shows the case where the face is an internal face.**



**(b) This figure shows the case where the face intersected by the line segment ($st$) from the face start node to the destination intersects an external face, and the point closer to the destination on the boundary of that face is the destination itself.**

**Figure 4.1: An example of how face routing works with internal and external faces.**

However, if the destination is reachable, there must be a point (not necessarily at a node) on the boundary of that face that is closer to the destination than the local minimum node (or is the destination). Upon reaching a local minimum the packet switches to FACE mode and face routing is used to either find such a point or demonstrate that none exists. In face routing (see Section 2.1.6.3), the boundary of a face is explored using the right-hand rule. The packet will either encounter a point on the boundary of the face that is closer to the destination or it will arrive back at the local minimum, having explored the entire face. In the latter case, the packet is dropped because, as will be shown, there is currently no route to the destination. In the former case, when a node determines that a packet has now made progress towards the destination, it reverts the packet's mode to GREEDY and attempts to again greedily route the packet. If another local minimum has been reached, face routing is initialized again, and the boundary of the next face occluding the packet from advancing towards the destination is explored. This greedy-face-greedy cycle continues until the packet is delivered or is deemed undeliverable.

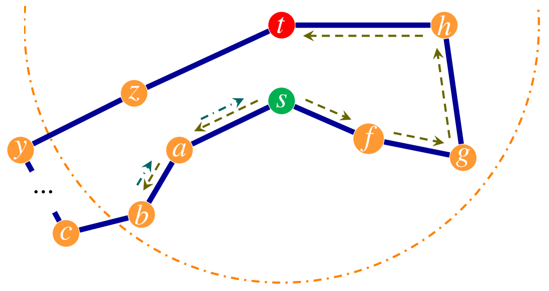


**Figure 4.2: An example of using an adaptive bounding circle to mitigate worst-case Face Routing situations, assuming there were many hops between $c$ and $y$.**

When a packet arrives at a local minimum it may choose to explore the face boundary in a clockwise direction, using the right-hand rule, or instead in a counter-clockwise direction, using the left-hand rule. In extreme cases, exploring a face in search of a point closer to the destination in one direction may require many more hops than exploring in the opposite direction. To mitigate these worst-case situations, an adaptive bounding circle (as in GOAFR+ [KWZZ03]) is added to the greedy-face-greedy foundation. This circular boundary ensures that when a packet attempts to explore one direction around a face, it only goes so far before retreating and attempting to advance in the opposite direction. If the packet is unable to reach a point closer to the destination within the bounding circle in either direction, the circle is expanded, and the process is repeated. Figure 4.2 shows an example of the bounding circle restricting the exploration of the face in a direction that would take more hops to reach the destination.

When in greedy mode, a packet is routed to whichever directly connected neighbor is nearest the destination. It is possible, when using this standard greedy algorithm, for a packet to be greedily routed down a dead-end path since a node does not use multi-hop connectivity information to make the forwarding decisions (it is generally assumed that a node using geographic routing does not know which nodes its directly connected neighbors are connected to). Note, however, that one by-product of the topology management protocol described in Chapter 3 is that each node has computed its “opinion” of what the topology should look like among the nodes of which it is aware. TAG introduces a modified greedy algorithm, called topology-aware greedy routing, that utilizes this local topology information to make better routing decisions when in greedy mode.

Similarly, nodes may use this local topology information to make better forwarding decisions when a local minimum is reached, and face routing is started. By analyzing the local topology, a node is able to suggest the best direction (clockwise or counter-clockwise) for a packet to begin exploring the boundary of the face. In addition, the radius of the circle bounding the exploration can be strategically set to avoid unnecessary exploration of the face boundary and expansion of the bounding circle. Note, however, that the local topology information is only an “opinion” of what the topology should be, and occasionally differs from the actual topology. This information, therefore, is only used to give hints of bounding circle size and exploration direction to the underlying algorithm.

If the topology changes while a packet is exploring the boundary of a face, the packet can get caught in a routing loop. Unlike most other geographic routing protocols, TAG is also designed to gracefully handle situations where the topology changes while a packet is exploring the boundary of a face. To enable this, nodes store the time-stamped previous connection for each data link. Whenever the topology changes, each affected node for each of its affected data links stores the ID of the node previously connected to and the time of the associated change in the topology. When packets reach a local minimum and begin exploring a face boundary, a time stamp indicating the time at which the packet began exploring the face, is added to the header. This time stamp combined with the stored previous connection information effectively enable a distributed ’snapshot’ to be taken of the face boundary at the time a packet commenced routing around a face. At each hop, nodes forward the packet around the face boundary as it existed at the time given in the header, not the current face boundary. Doing so avoids routing loops resulting from a dynamic topology, but requires special handling for situations where an edge of the original face no longer exists in the topology, and requires that nodes store a small amount of extra data for each of their data links (the ID of the node previously connected with and the time the connection ended).

The next several sections give more details about the TAG header, the TAG algorithms, and how each of the optimizations introduced here actually work.

## 4.2 Shim Header

Before delving into the algorithm details, this section describes the contents of the small header added by TAG to each packet in addition to the standard network headers. This "shim header" is similar to those used in Dynamic Source Routing (DSR) [JM96], and Multiprotocol Label Switching (MPLS) [RVC01]. In contrast to protocols such as DSR, however, the size of the header does not grow with the length of the path or the size of the network. As described in Section 2.1.2.1, commercial aircraft are already required by the FAA to exchange position information with nearby aircraft using the ADS-B system. Assuming such a system is used to exchange positions, and assuming that the destination position is known by the sender, the bytes in the TAG's headers are the only overhead bytes that must be transmitted by TAG. No control, update, or beacon messages are required. Table 4.1 lists and describes the contents of the header.

**TABLE 4.1: The fields in the additional packet header.**

| Symbol | Description | Size (bytes) low-resolution | Size (bytes) higher-resolution |
|---|---|---|---|
| $M$ | Mode: GREEDY, FACE-FWD, FACE-REV, or FACE-RETURN | 1 | 1 |
| $D$ | Destination Location (x, y) | 2x16 bits = 4 | 2x32 bits = 8 |
| $d$ | Destination Node ID | 1 | 2 |
| $s$ | Source Node ID | 1 | 2 |
| $f$ | Face Start Node ID | 1 | 2 |
| $e$ | first edge traversed (ID of 2nd node visited) | 1 | 2 |
| $fd$ | Distance from face start position to destination | 2 | 2 |
| $cw$ | Direction (clockwise or counter-clockwise) | 1 | 1 |
| $r$ | Radius of the bounding circle | 2 | 2 |
| $t_{\text{faceStart}}$ | time started face | 2 | 4 |
| | total size (bytes) | 16 | 26 |

The header first includes a mode flag, $M$. This value enables each packet to independently store which of four modes it is operating in (greedy mode or one of three face routing modes that will be discussed shortly). Next, the header includes the destination location $D$. Both greedy routing and face routing modes must know the approximate destination location to be able to route the packet correctly. Each node is assumed to have a single unique node identifier or node ID. This value could be the IP-address associated with one of its interfaces, or it could be any other node unique value. The header next includes four node IDs: a destination node ID, $d$, a source node ID, $s$, the ID of the node where the packet last began tracing the boundary of a face (face start node ID), $f$, and $e$, the node ID of the second node visited (or to be visited) while tracing the boundary of a face (the node visited directly after $f$). In conjunction with $f$, $e$ essentially encodes the first edge traversed on the current face. Next, the packet includes $fd$, the distance from the latest face start position to the destination position. Using this value a packet in face routing mode can determine when it has advanced towards the destination. The Boolean *cw* value facilitates the choice of tracing a face boundary in either a clockwise (right-hand rule) or a counter-clockwise (left-hand rule) direction. The radius of the packet's current bounding circle is stored in $r$. Finally, $t_{\mathrm{faceStart}}$ stores the time at which this packet began tracing the boundary of a face.

Only the first three values are needed for greedy routing, and the remainder of the values are used only for the face routing mode. The destination node ID is also optional, but helpful. When packets include the destination ID it is possible to update the destination location when the packet visits a node which knows the current position of the destination node (given the destination node ID). Also, the destination node ID can be used to confirm that the packet has indeed arrived at the expected destination (not some other node at that position).

The source node ID is only needed to support dynamic topologies or to support notifying the source node of errors. $f$ and $e$ must be stored in the packet in order to accurately detect when a packet has traced the entire boundary of the face separating the face start node from the destination. The $t_{\mathrm{faceStart}}$ value is also only needed to support dynamic topologies.

The current implementation of the header used for simulation also includes two additional values. The first value stores the node ID of the previous hop. It is assumed this information could be obtained in a real system by querying the data link associated with the incoming interface. It is, therefore, not included in Table 4.1. The second value stores the position of the face start node. It is needed only if the intersection test described in Section 4.12 is enabled (and even then, only in rare cases). The packet need not store this position for default operation when using the DC-GG topology generation algorithm, and it is not included in the header for the simulation results presented in Chapter 5.

Assuming 16-bit values suffice for the radius value and location components (3 16-bit values per location), and the time value also requires only 16 bits, 256 nodes could be supported with a 16-byte header, as indicated in the low-resolution size column in Table 4.1. With an additional 4 bytes (20 bytes total), 65,536 nodes can be supported. The simulations in this work use a higher-resolution version of the packet where

26 bytes are used (also shown in Table 4.1).

As will be explained later, TAG's shim header must sometimes become a stack of headers which is generally limited to a depth of 2 headers. The maximum size of the header for all situations is then 26 bytes for each of 2 headers plus a single 8-bit value to encode the number of headers in the stack. This gives a maximum size of 33 bytes for the low-resolution header and 53 bytes for the higher-resolution header. To overcome an implementation issue, the size of the stack of headers used in the simulations for this work is 73 bytes total. In addition to the 26 bytes described above, the implemented version of the header contains a 2-byte field storing the identifier of the previous hop. Further, each header in the implementation includes 8 bytes of unused space. The total size of each header is then 36 bytes, and the stack of headers then occupies 72 bytes plus an additional byte representing the number of headers, for a total of 73 bytes.

## 4.3 Greedy Routing

Greedy routing is the most basic component of TAG, therefore it is detailed first. Greedy routing requires only knowledge of the positions of directly connected neighbors, and the position of the final destination. As described in Section 2.1.6.2, the algorithm for determining a packet's next hop in standard greedy mode is simple; the packet is forwarded to the directly connected neighbor that maximizes the packet's advance towards the destination (minimizes the remaining distance to the destination). If the forwarding node is closer to the destination than all of its directly connected neighbors, the packet has encountered a local minimum and cannot make greedy progress. In this case, there is a void or hole in the topology that inhibits greedy routing. When relying solely on greedy routing, packets reaching a local minimum such as this must be dropped. If, however, other routing modes are supported (such as face routing), these modes must now be employed to route the packet around this void. The packet must backtrack as necessary and then progress until it reaches a point that is closer to the destination than where the local minimum was encountered. At this point, greedy routing can again be attempted.

Assuming $N$ represents the set of a node's directly connected neighbors, $t$ represents the target node and dist() is a method that returns the linear (straight line) distance between two nodes, the function, $g(N,t)$, computes the greedy choice for the next node visited and is defined as:

$$g(N,t) = \{ n \in N \mid \text{dist}(n,t) \, \text{is a minimum} \} \tag{4.1}$$

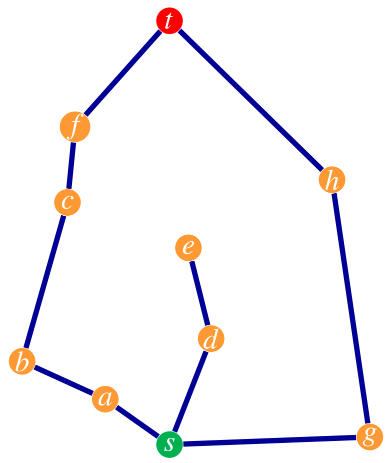
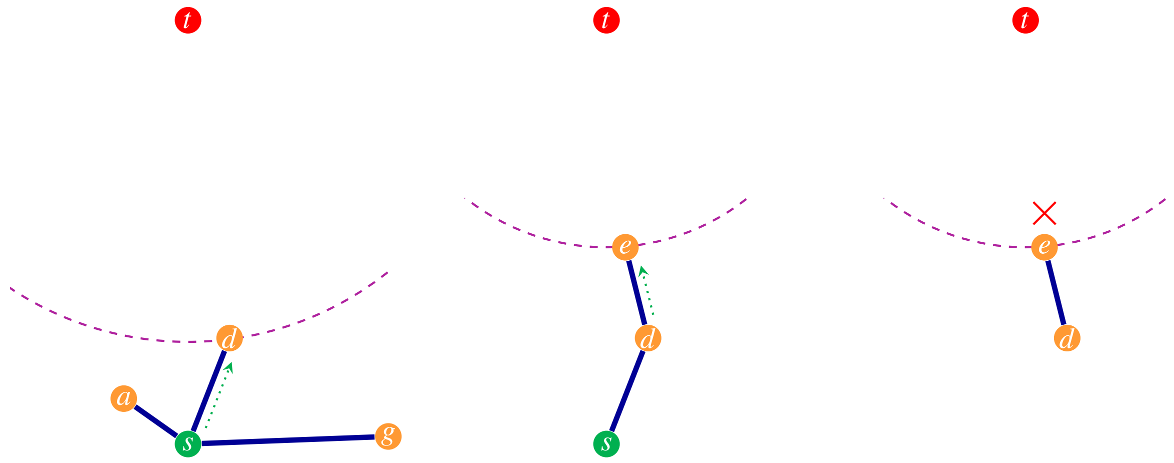


**Figure 4.3: An example topology used for several examples in this chapter.**

Figure 4.3 shows a topology that will be used to investigate various conditions of the geographic routing protocols. The topology is not necessarily derived from realistic aircraft positions, or produced with the topology management framework, but it is designed to compactly exhibit several characteristics that do occur in real topologies. In the figure, node $s$ is the source node for a flow of packets and $t$ is the target node or destination for these packets.



**(a) Greedy routing hop $s$ to $d$.** **(b) Greedy routing hop $d$ to $e$.** **(c) Greedy routing dead end.**

**Figure 4.4: An example of greedy routing in the example topology.**

Figure 4.4 shows an example of greedy routing in this example topology. Each image shows the decision process for a single hop as the packet makes its way toward the destination. Only the destination position and the local information available to the greedy algorithm are displayed in each step. This local information includes the local node's position and the positions of directly connected neighbors. This local information combined with the position of the destination (part of the state information contained in the special shim header) is enough to enable greedy routing. In 4.4a the dashed circle centered at *t* and intersecting *d* shows that *d* is clearly closer to *t* than *a* or *g*. Therefore the packet is forwarded to *d*. Notice that node *s* is essentially unaware of node *e* when deciding to forward the packet to node *d*. The new local view at *d* is shown in Figure 4.4b. Since *e* is the closest node to *t* that is directly connected *d*, the packet is forwarded to *e*. Again note that in this case node *d* need not know anything about node *g* to greedily route the packet to node *e*. After arriving at node *e* one could imagine that if *e* were directly connected to *t* the packet would reach the destination on the next hop. The more interesting example, however, is to assume that *e* is only directly connected to *d*, as shown in Figure 4.4c. In this case, the packet has reached a dead end or local minimum, and cannot make greedy progress.

As described in Section 2.1.6.2, greedy routing is extremely efficient, but cannot reliably, by itself, always deliver packets to their destinations. Greedy routing in a static topology requires only the destination position value from the shim header. In dynamic topologies, the destination node ID is also needed to ensure that a packet has reached the correct node and to update the destination position while the packet is en route.

### 4.3.1 Face Routing

Unlike greedy routing, face routing can generally deliver packets to their destinations reliably, but not as efficiently. As described in Section 2.1.6.4, when greedy routing and face routing are combined, a synergy is produced, resulting in a protocol that is both efficient and reliable. The basic face routing method described here seeks to forward a packet from a local minimum, around the boundary of a face in the planar topology graph, until a node closer to the destination is reached, or the packet explores the entire boundary of the face and returns to the local minimum node. The boundaries of the faces that are explored are implicit, such that an individual node need not be aware of the entire network topology that forms the boundaries of the faces to which the node is adjacent. It is sufficient for nodes to store the positions of directly connected neighbors, and for each packet to store minimal state information such as the node and position at which face routing started, and the direction in which the face should be explored.

The face boundary is explored using the right-hand rule. This rule says that upon reaching a node *y* having just visited a node *x* the next edge traversed should be the next edge sequentially counter-clockwise about *y* from edge *xy*. Another way to think about the right-hand rule is to imagine that the edges are roads and the vertices are traffic circles or roundabouts, where drivers drive on the right-hand side of the road. At each roundabout, the packet is forwarded as if it were a car that always takes the first right exit from every roundabout encountered.

The right-hand and left-hand rules can be formally defined as follows. Assuming $a$ and $b$ are directly connected neighbors and pos() returns the 2-dimensional position of a node, the bearing $\theta_{ab}$ (at $a$ toward $b$) is given by

$$\theta_{ab} = \pi + \text{atan2}(\text{pos}(b)_y - \text{pos}(a)_y, \text{pos}(b)_x - \text{pos}(a)_x) \tag{4.2}$$

The atan2 function is an arctangent function that takes two arguments ($y$, $x$) that specify a point in the plane and returns the measure of the angle between the positive x-axis and the given point. Unlike arctangent, atan2 is able to determine the correct quadrant for the result and is able to correctly produce a result of $\pm\pi/2$, where arctangent breaks down (requiring a divide by zero). Using atan2 the angle at $a$ toward $b$ can be computed. The result has $\pi$ added to it to yield a bearing angle with a range between 0 and $2\pi$.

Making use of equation 4.2, the difference in bearing angle at node k between incident edges from nodes $j$ and $l$ can be computed using the following formula:

$$\Delta\theta_{jkl} = \begin{cases} \theta_{kj} - \theta_{kl}, & \text{if } \theta_{kl} < \theta_{kj} \\ 2\pi + (\theta_{kj} - \theta_{kl}), & \text{otherwise} \end{cases} \tag{4.3}$$

This formula essentially computes the absolute value of the difference between two bearings. If the result of the subtraction would yield a negative number, $2\pi$ is added to the value.

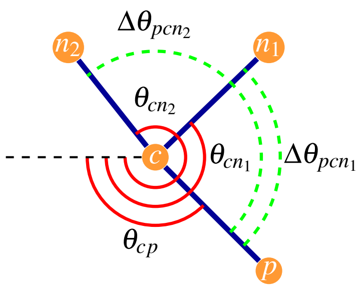


**Figure 4.5: A depiction of the bearing calculations.**

Figure 4.5 shows an example of the bearing calculations at node $c$ for neighbor nodes $p$ (the previously visited node), $n_1$, and $n_2$. The arcs labeled $\theta_{cp}$, $\theta_{cn_1}$, and $\theta_{cn_2}$ indicate the bearing angles relative to the negative x-axis. The arcs labeled $\Delta\theta_{pcn_1}$ and $\Delta\theta_{pcn_2}$ indicate the change in bearing between the given node pairs relative to node $c$.

Assuming N is the set of the forwarding node's directly connected neighbors, and that $\theta_{cp}$ is the incoming bearing (at the current node toward the previous), then the next node in face routing mode using the right-hand rule ($f_r(N, \theta_{cp})$) is given by:

$$f_r(N, \theta_{cp}) = \{n \in N \mid \Delta\theta_{pcn} \text{ is a minimum}\} \tag{4.4}$$

The right-hand rule essentially chooses the next hop that minimizes the change in bearing from the incoming edge. If the left-hand rule is instead used, the next node visited is given by:

$$f_l(N, \theta_{cp}) = \{n \in N \mid \Delta\theta_{pcn} \text{ is a maximum}\} \tag{4.5}$$

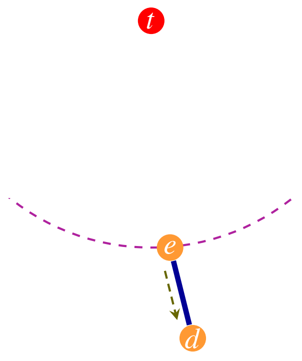
In the example shown in Figure 4.5, a packet arriving at node $c$ from node $p$ would proceed to node $n_1$ when using the right-hand rule.

An initial incoming bearing value must be determined before the first hop around a face boundary can be computed. Since the face that the packet will explore is the face between the local minimum node and the destination, an appropriate incoming bearing can be calculated by imagining that the packet just arrived from the direction of the destination node, and calculate the incoming bearing as usual. If $t$ is the destination node and $s$ is the source node, the initial incoming bearing ($\theta_{sp}$) can then be given by:

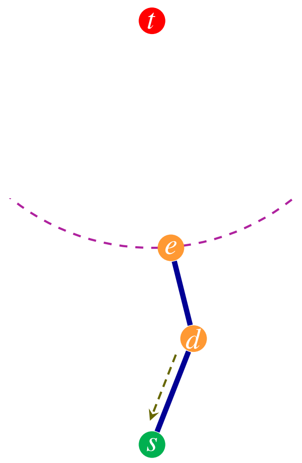
$$\theta_{sp} = \theta_{st} \tag{4.6}$$

Like greedy routing, face routing makes use of the destination position stored in the packet header. In addition, the distance between the destination and the node where the current face began being explored must be stored in the packet. This enables a packet to determine when it has reached a point closer to the destination than where it began routing around the face. The packet must also store the first edge visited on the face boundary. This information allows a packet to determine when it has explored the entire boundary of a face.

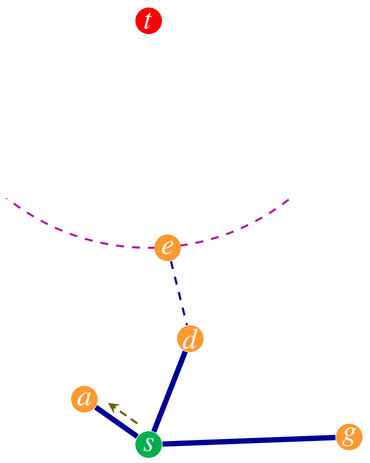
An example of face routing is shown in Figure 4.6. This figure shows the decision process at each hop as a packet is routed through the topology shown in Figure 4.3. This is the same topology used to demonstrate



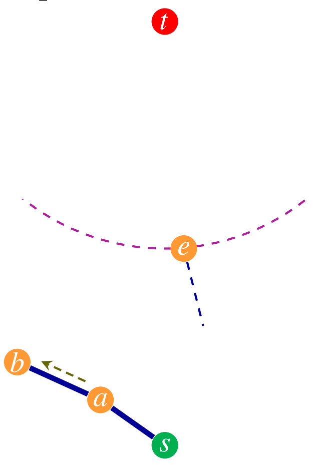
**(a) Face routing hop $e$ to $d$.**



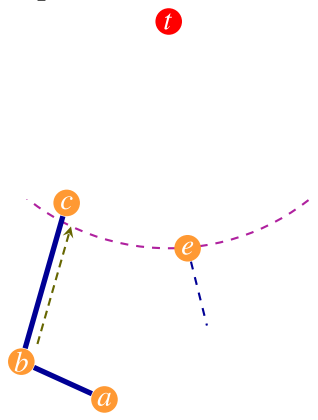
**(b) Face routing hop $d$ to $s$.**



**(c) Face routing hop $s$ to $a$.**



**(d) Face routing hop $a$ to $b$.**



**(e) Face routing hop $b$ to $c$.**

**Figure 4.6: An example of face routing in the example topology.**

greedy routing. In this example, the packet transitions to face routing mode at node $e$, where the greedy routing example encountered a local minimum. Note that each image shows only the local information and header information used to make each forwarding decision. The dashed curve is a portion of the circle centered at $t$. If this circle is crossed, progress has been made toward the destination.

In figure 4.6a, face routing commences for the packet. There is only one possible outgoing edge from node $e$, so determining the next hop, in this case, is trivial. In general, however, to determine which edge of the face a packet should visit first, imagine that the packet has just arrived from the direction of the destination node, and forward the packet using the same logic used to explore the remainder of the face boundary. That is, forward the packet over the next edge counter-clockwise about the current node from the incoming edge (where the incoming edge is an imaginary edge from the destination node to the current node, when starting face routing). At node $d$ the packet the proceeds across the only other adjacent edge to node $s$, as shown in Figure 4.6b.

At node $s$ the packet can either be forwarded to $a$ or to $g$ as shown in Figure 4.6c. Node $a$ is chosen as the next hop since it is the first edge encountered while sweeping in a counter-clockwise direction (from the perspective of the node) from the incoming edge. Another way to visualize this is to think of $s$ as an American-style roundabout, and the edges as roads. To follow the right-hand rule, upon entering the roundabout a packet should always take the first exit. In this case, that first exit, after entering the roundabout from the road to $d$, is the road (edge) that leads to $a$.

Note that the dashed line from $e$ to $d$ represents the first edge explored on this face boundary. This edge is implicitly stored in the packet header by including the node ID of nodes $e$ and $d$. The packet continues to $b$ and then $c$ as shown in Figures 4.6d and 4.6e. At each hop, the right-hand rule is followed.

The packet has now reached a node that is closer to the destination than the node at which this face began being explored. If face routing is combined with greedy routing, the packet can now attempt to revert to greedy routing mode.

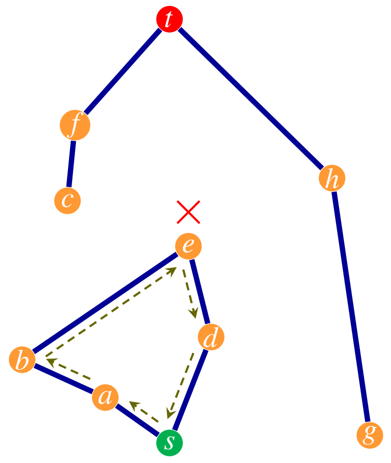


**Figure 4.7: An example of face routing when the destination is unreachable.**

Suppose that the topology was instead like that in Figure 4.7 such that $b$ was connected to $e$ instead of $c$, and there was no edge from $s$ to $g$, making the destination unreachable. In this case, if the left-hand rule were used instead, face routing would proceed exactly as the previous example except that the packet would proceed towards $e$ after visiting $b$. Upon returning to $e$ the packet would again apply the left-hand rule and determine that the next edge explored would be that toward $d$. Since this next edge matches the edge first explored by the packet (and stored in the header), the packet has certainly explored the entire face boundary, and it can be shown that there is currently no route to the destination. At this point, the packet is dropped. Note that if the right-hand rule were used, the result would be the same, but the nodes would be visited in the opposite order.

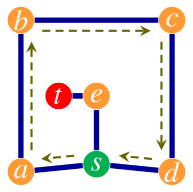


**Figure 4.8: An example of face routing motivating the need for the first edge to be included in the header.**

Knowing the node where face routing started is not sufficient to determine that the entire face boundary has been visited. For example, the topology shown in Figure 4.8 shows a situation where a packet enters the network at node $s$ and explores the boundary of the face visiting nodes $a$, $b$, $c$, and $d$, and then returning to $s$. If the packet only used the starting node (and not the first edge explored) to determine if the entire face boundary had been visited, the packet would be dropped upon returning to $s$, without visiting node $e$ and delivering the packet to node $t$. For this reason, each packet must also include the first edge visited. This will ensure that situations like that shown in Figure 4.8 are handled appropriately, and the entire boundary of a face is actually explored before the packet is ever dropped.

Using face routing as described here introduces a problem in topologies that are not unit disk graphs. In such graphs there may be cases where no node is closer to the destination than the local minimum, however the packet does reach a point on one of the edges that is closer to the destination. This situation will be analyzed and fixed in Section 4.12.

## 4.4 Non-planar Topologies

Why must a topology be planar to support face routing? This section attempts to answer that question by presenting a couple of pathological cases where a non-planar topology can cause face routing to fail to deliver a packet to a reachable destination. This is, however, not intended to be an exhaustive coverage of all pathologies.

Figure 4.9 shows a topology where the links between nodes $a$ and $e$ and nodes $b$ and $d$ cross, creating a non-planar physical topology. When any greedy-face-greedy protocol is used to route a packet starting at node $s$ and heading to node $t$, the packet would immediately switch to face routing mode, since it has no

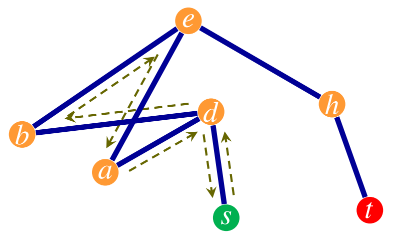


**Figure 4.9: An example of a non-planar topology that thwarts face routing. The arrows trace the path of a packet starting at $s$ and using the right-hand rule in an attempt to get to $t$.**

directly connected neighbor closer to the destination than itself. Using the right-hand rule (taking the first right exit at the roundabout at each vertex) the path traced would be $s$, $d$, $b$, $e$, $a$, $d$, $s$, as indicated by the arrows in the figure. Notice that although there is a path to $t$, face routing fails to find it, and drops the packet.

The network topology could be made planar by adding a single node at the point of intersection of the crossing connections. Face routing would succeed in this case since the packet would now advance from $d$ to the added node, and then on to $e$, $h$, and finally $t$.

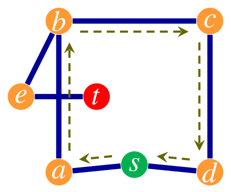


**Figure 4.10: An example of a non-planar topology where face routing fails to forward a packet to a reachable destination.**

In Figure 4.10 another pathological case is displayed. Here the destination is on the interior of a face but is not connected to the boundary of that face. The edge from $e$ to $t$ crosses the edge from $a$ to $b$, making the topology non-planar. Assume a packet starts at $s$ and has $t$ as its target or destination. Since $s$ is closer to the destination than either $a$ or $d$, a packet using any greedy-face-greedy routing scheme would immediately switch to face routing mode and would visit the nodes $s$, $a$, $b$, $c$, $d$, and then finally arrive back at node $s$. The packet would be dropped and it would be assumed that the destination was unreachable, but there is a path to the destination. Notice again, that the addition of a single vertex at the point of intersection of the crossing links would allow the packet to be easily delivered (via the path $s$, $a$, $t$).

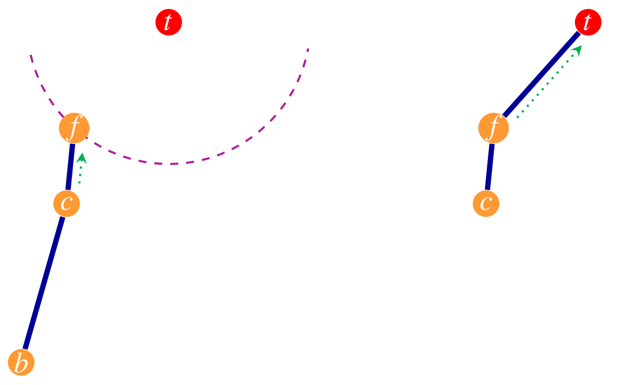
These examples emphasize the need for the topology used for face routing to be globally non-planar. If this is not the case packet drops can be expected when these and other pathologies are encountered.

## 4.5 Greedy-Face-Greedy Routing

Combining Greedy and Face routing together yields a protocol that is efficient and reliable, as described in Section 2.1.6.4. Efficient greedy routing is used whenever possible, and face routing is used to route around “holes” or voids in the network topology when local minima are reached. In order to support both greedy and face routing, a mode value is needed in the shim header.

The greedy-face-greedy algorithm works by first greedily advancing until either reaching the destination or a local minimum. If the destination is reached, the packet is delivered, and TAG’s job is done. Otherwise, upon reaching a local minimum, the packet changes to face routing mode and begins tracing, in a clockwise direction (and only clockwise in Greedy-Face-Greedy routing), the boundary of the first face encountered by a line segment extending from the local minimum node to the destination.

Greedy-face-greedy routing would follow the greedy path described in Figure 4.4, and then, by face routing, the path described in Figure 4.6. After reaching a node located closer to the destination than where face routing began, the packet would revert to the greedy routing mode. Figure 4.11 shows the final greedy steps resulting in delivery of the packet. In Figure 4.11a the packet is forwarded from $c$ to $f$, its directly connected neighbor that is closest to the destination. Then finally in Figure 4.11b, the packet is forwarded to the destination.



**(a) Greedy routing hop $c$ to $f$.** **(b) Greedy routing hop $f$ to $t$.**

**Figure 4.11: An example of Greedy-Face-Greedy Routing.**

## 4.6 Bounded Face Exploration

A face can be explored in either a clockwise direction, using the right-hand rule or a counter-clockwise direction, using the similar left-hand rule. However, for basic face routing packets are always routed in the same direction. With the addition of a flag in the header indicating the direction a face is being explored, packets can be routed in either direction. In general, however, there is no local information available to determine which direction would be more optimal.

In some cases, the choice of which direction to explore a face can have a significant impact on the efficiency of face routing. Take the topology in Figure 4.12, for example. In this topology the ellipsis indicates some large number of hops between nodes $c$ and $y$. Reaching the destination by exploring the face between the source and destination in a counter-clockwise direction would require only 4 hops, while exploring in a clockwise direction could require tens or even hundreds of hops. This is the worst case scenario for face routing. Given only the local information at node $s$, it is generally impossible to determine which direction will minimize the number of hops required to reach the destination.

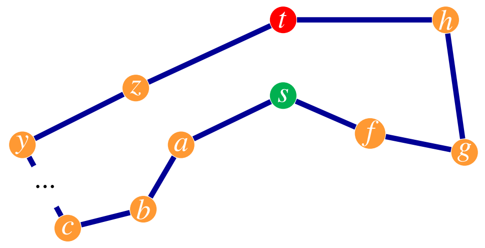
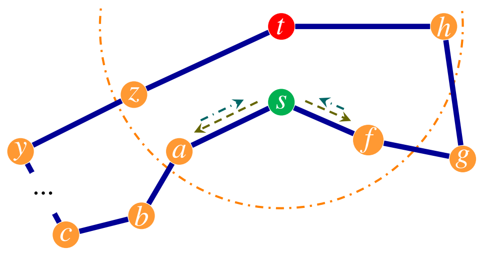


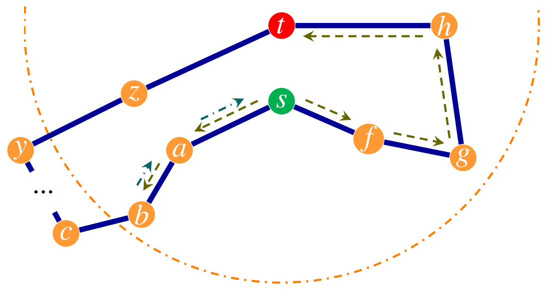
**Figure 4.12: An example of a topology where the face exploration direction makes a difference.**

As described in 2.2.2.2.4 Adaptive Face Routing minimizes the impact of these worst case scenarios. Adaptive Face Routing uses a boundary within which the packet will be face routed. If this boundary is crossed while exploring the face in one direction, the packet returns to the local minimum node and then explores the face in the opposite direction. If traveling around the face boundary in this direction would also cause the packet to cross the boundary, the boundary is too small. The packet returns again to the local minimum node and the boundary size is increased. This entire process is repeated until progress can be made toward the destination or the entire boundary is explored. Like GOAFR+, TAG uses a circle centered at the destination to bound the face exploration.

Figure 4.13 shows an example of adaptive face routing for the example worst case topology. The source node is itself a local minimum, thus packets switch to the face routing mode immediately. When face routing



**(a) The initial bounding circle radius does not allow progress in either direction.**



**(b) An expanded bounding circle allows the packet to reach the destination without having to visit all the nodes along the clockwise face boundary exploration path.**

**Figure 4.13: An example of Adaptive Face Routing.**

starts, a bounding circle is defined as indicated by the dashed line in Figure 4.13a, and the packet starts being forwarded in a clockwise direction. The packet is forwarded to node $a$, but at this node it is determined that continuing to explore the face boundary in this direction (toward $b$) would require crossing the bounding circle. The direction in which the packet is exploring the face boundary is flipped to counter-clockwise, and the packet returns to node $s$. The packet then continues to node $f$, where again exploration is stopped because the next hop would cross the bounding circle. The exploration direction is again flipped and the packet is returned to node $s$. The bounding circle is now expanded, as shown in Figure 4.13b, and the packet visits nodes $a$ and $b$ before encountering the new boundary. The direction is again flipped, and the packet makes its way back to $s$ and then to nodes $f$, $g$, $h$, and finally to the destination node $t$. Although this path required 8 hops more than the optimal path, it is much more efficient than exploring the numerous hops along the clockwise path.

To support adaptive face routing, the packet must now include a flag to indicate in which direction the face is currently being explored, and a value indicating the current radius of the bounding circle. In addition, the face routing mode must now be split into three face routing mode values: FACE-FWD, FACE-REV, and FACE-RET. Other parameters and formulas are needed to compute the initial radius and expanded radii of the bounding circles.

One way to think about the face routing mode, when adaptive face routing is employed, is to imagine that the packet is being routed around the face inside of an annulus (a ring-shaped object) centered at the destination. Figure 4.14 shows an example of an annulus and a topology. The space inside the inner circle of the annulus is closer to the destination than the local minimum node where this packet started (or re-started) face routing. The radius of this inner circle is equal to the distance from the local minimum position to the destination. The outer circle is the bounding circle whose radius was set when switching to face routing mode. In Figure 4.14, a packet could route around the face boundary within the annulus from node $a$ to $b$ to $c$ and then to node $d$. If the packet crosses the inner circle, however, and the packet is forwarded to $e$, it has made progress, and is now closer to the destination than when it began face routing. It may now safely attempt to switch back to greedy mode. If, on the other hand, the packet would cross the outer circle on its next hop, as is the case with node $f$, the exploration direction is flipped and the network nodes then forward the packet such that it backtracks and explores the boundary in the opposite direction.

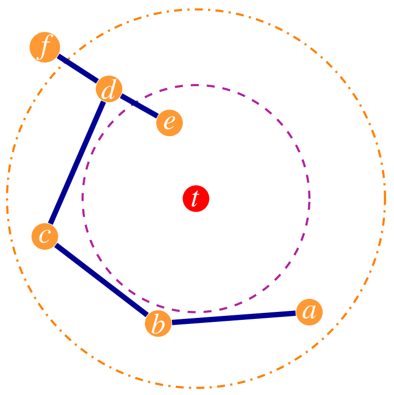


**Figure 4.14: A depiction of the notion of routing within an annulus centered at the destination.**

As mentioned above, additional mode values are required to support adaptive face routing. Initially, the packet begins face routing in the FACE-FWD (face forward) mode. If the boundary is encountered, the direction is reversed and the mode changes to FACE-REV (face reverse) mode. The packet eventually arrives at the node where it began face routing and continues tracing the face in the opposite direction. As before, if the packet crosses the inner circle of the annulus it transitions to GREEDY mode. However, if instead, on its next hop it would cross the outer bounding circle, the direction is again flipped, and the packet transitions to the FACE-RET (face return) mode. This will cause the packet to backtrack all the way to the node where it started face routing. Finally, the radius of the outer bounding circle is increased by multiplying it by a configurable bounding circle increase factor (generally a value ranging from 1.4 to 1.5). Finally, the packet begins the entire face routing procedure again, starting in the FACE-FWD mode.

While face routing, if the packet ever encounters the destination, it is delivered. If however, while in FACE-FWD mode, the packet detects that it has traced the entire boundary of a face, then there is no path to the destination and the packet is intentionally dropped. Dropping the packet at this point is actually a critical benefit of Face Routing, for if the packet were allowed to continue trying to reach the unreachable destination, it would waste valuable resources while looping indefinitely.

The following explanation gives some reasoning about why packets should be dropped after having explored the entire boundary of a face. If, while exploring the boundary of a face in a globally planar static graph, a packet returns to the node where it started routing around the face, and its next hop, according to the right-hand rule (or left-hand rule if exploring in the opposite direction) would take it to the first node visited while exploring the boundary (the entire boundary has been traced), and the packet never encountered any location (at a node or along an edge between nodes) that is closer to the destination than where it started routing around the face, then there is no path to the destination in the graph, and the packet should be dropped. Consider that in a planar graph the edges segment the 2-dimensional space into a set of faces, perhaps several internal faces and one external face which encompasses all other space in the plane. Face routing attempts to explore the boundary of the face incident to the face start node and intersecting the line segment from it to the destination. If the face whose boundary the packet is exploring is an internal face and none of the points on the boundary come closer to the destination than the face start node, then the destination must be inside of and disconnected from the face whose boundary was explored. If the destination node had been connected to the graph there would have to have been some point on the boundary of the face that came closer to the destination. The other possibility is that the face between the source and destination nodes is the external face. In this case, the packet would explore the entire outside boundary of the entire connected component. If the packet never advanced towards the destination around this external face, then it is clear that the destination lies outside of and disconnected from the entire connected component.

If the packet attempts to find a node (not just a point) that is closer to the destination than the face start node, this logic no longer always applies. In arbitrary planar graphs (such as graphs that are not unit disk graphs) there can be situations where no node on a face boundary is closer to the destination than the face start node. See Section 4.12 for more on how this situation can be handled.

Face routing assumes that the topology is static while the packet is in flight. When the topology changes, packets can get stuck in routing loops. TAG is designed to deal with dynamic topologies as described later in Section 4.11.

## 4.7 Base Protocol Implementation

This section includes the implementation details for the base version of TAG described thus far.

Before each packet begins its journey, the position of its intended destination must be determined. Depending on the type of network and destination node, the destination position may need to be obtained from a separate location service such as [LJDC$^{+}$00]. If the destination is a static gateway node, however, the destination's constant position may already be known by every node. It should be noted, as well, that TAG is able to handle stale destination positions for distant nodes. These stale destination positions are updated by TAG once the packet is within the position exchange range of the destination.

The STARTPACKET function initializes the shim header for a new packet at a source node. For each new packet to be routed, the destination position and ID (*D* and *d*) are added to the shim header, along with the source node (current node) ID. Every packet starts in GREEDY mode (*M*= GREEDY). The shim header is attached to the packet and then the TAGFORWARD function, shown below, is called.

```
1: function STARTPACKET
2:     $h.M$ = GREEDY
3:     $h.s$ = self.$id$
4:     $h.D$ = destination position
5:     $h.d$ = destination id
6:     TAGFORWARD($h$)
7: end function
```

TAGFORWARD is the main routing function and is called at each hop along the path. If the packet has reached its destination, the packet is delivered. Otherwise, various methods are called depending on which mode the packet is in, with each method returning the ID of the node to which the packet should be sent next. For packets in GREEDY mode, the GREEDYORREVERTTOFACE function is called to either send the packet to the neighbor that is nearest the destination, or revert to face routing if the current node is a local minimum. If the packet is in the FACE-FWD or FACE-REV mode, the FACECOMMON method is simply called (detailed later). Lastly, for packets in the FACE-RETURN mode (in which the packet is returning to the face start node), the next node to be visited while backtracking to the face start node is determined by the RIGHTHANDFORWARD function. If, however, the packet has reached the face start node, the boundary circle is expanded and face routing is restarted by (EXPANDANDRESTARTFACE).

```
function TagForward($h$, incomingIf)
    if $h.d$ == self.$id$ then
        deliver packet
    else
        switch $h.M$
            case GREEDY
                next = GreedyOrRevertToFace($h$)
            case FACE-FWD
                next = FaceCommon($h$, incomingIf)
            case FACE-REV
                next = FaceCommon($h$, incomingIf)
            case FACE-RETURN
                if $h.f$ == self.$id$ then
                    next = ExpandAndRestartFace($h$, IncomingIf)
                else
                    next = RightHandForward($h$, false)
                end if
        drop packet on failure, otherwise forward to next and call TagForward
    end if
end function
```

```
function GreedyOrRevertToFace($h$)
    next = GreedyForward($h$)
    if next == failure then
        next = StartFace($h$)
    end if
    return next
end function
```

```
function ExpandAndRestartFace($h$, incomingIf)
    $h.r = h.r * \rho$
    return ReverseTheDirection($h$, incomingIf)
end function
```

The FACECOMMON method first determines the proposed next node to be visited according to the right-hand rule. If the packet is (1) in FACE-FWD mode, (2) at the face start node $f$, and (3) expected to visit node $e$ next, the packet has made a full loop around the face, and it is dropped because there is no route to the destination. If instead, the packet reaches a node that is nearer the destination than the face

```
function GreedyForward($h$)
    $N$ = self.$directlyConnectedNeighbors$ + self.$id$
    nearest = $g(N, h.D)$                                                    ▷ see Equation 4.1
    if nearest == self.$id$ then
        return failure
    end if
    return nearest
end function
```

```
function RightHandForward($h$, starting)
    if starting == true then
        previous = $h.D$
    else
        previous = position of node at previous hop
    end if
    current = self.$position$
    $N$ = self.$directlyConnectedNeighbors$
    incomingBearing = $\pi$ + atan2(previous.y-current.y, previous.x-current.x)    ▷ see Equation 4.2
    if $h.cw$ == clockwise then
        return $f_r(N$, incomingBearing$)$                                    ▷ see Equation 4.4
    else
        return $f_l(N$, incomingBearing$)$                                    ▷ see Equation 4.5
    end if
end function
```

```
function ReverseTheDirection($h$, incomingIf)
    if $h.cw$ == clockwise then
        $h.cw$ = counter-clockwise
    else
        $h.cw$ = clockwise
    end if
    return node connected to incomingIf
end function
```

start node, GREEDYORREVERTTOFACE is called. This method starts greedily routing unless this node is found to be another local minimum, in which case face routing is started on the face between this node and the destination. Ignoring, for now, the highlighted code, if the proposed next node lies outside the bounding circle, then we REVERSETHEDIRECITON in which the packet is traveling (changing directions, for example, from clockwise to counter-clockwise). If the packet had been in FACE-FWD mode the mode is changed to FACE-REV mode. If instead, the packet is already in FACE-REV mode the mode is changed to FACE-RETURN. In some cases, the packet is already at the face start node so the FACE-RETURN mode is skipped and the bounding circle is immediately expanded (by multiplying it by a bounding circle increase factor) and face routing is restarted with EXPANDANDRESTARTFACE.

Other pseudocode methods are provided for completeness, but are not detailed here.

```
function FaceCommon(h, incomingIf)
    next = RightHandForward(h, false)
    if h.M == FACE-FWD and self.id == h.f and next == h.e then
        return failure                                   ▷ No route to destination, drop
    end if
    if Dist(self.position, h.D) < Dist(h.F, h.D) then
        return GreedyOrRevertToFace(h)                   ▷ Fall back to greedy
    end if
    if Intersects(F, D, self.position, next.position) then   ▷ only if intersection mode is enabled
        return StartFace(h)
    end if
    if Dist(next.position, h.D) > h.r then
        next = ReverseTheDirection(h, incomingIf)
        if h.M == FACE-FWD then
            h.M = FACE-REV
            return next
        end if
        if h.f == self.id and h.e == next then
            return ExpandAndRestartFace(h)
        end if
        h.M = FACE-RETURN
        return next
    end if
    return next
end function
```

```
function StartFace($h$)
    $h.M$ = FACE-FWD
    $h.f$ = self.$id$
    $h.fd$ = Dist(h.D, self.$position$)
    $h.F$ = self.$position$                                  ▷ used only if intersection mode is enabled
    nearest = FindConnectedNodeNearestPosition($h.D$)          ▷ only if topology-aware enabled
    $h.cw$ = RightWayToReach(nearest)                          ▷ if not topology-aware, always clockwise
    $h.r$ = SmartRadius(nearest, $h.D$)                        ▷ if not topology-aware, Dist($F$,$D$) * $\rho_0$
    next = RightHandForward($h$, true)
    $h.e$ = next
    $h.t_{faceStart}$ = time.Now()
    return next
end function
```

## 4.8 Topology Awareness Overview

When a packet initiates face routing, it can proceed in a clockwise direction (right-hand), or decide instead to travel in a counter-clockwise direction. If topology information is available, it can be used to make a better choice of the face boundary exploration direction. Similarly, if some topology information is available, the initial radius of the bounding circle can be set intelligently. Topology information can also assist in the selection of the next hop for greedy routing, enabling a more optimal greedy choice. A topology-aware greedy hop, however, must always make progress towards the destination.

To enable this enhancement local topology information could be exchanged periodically via some other protocol. This would incur an overhead penalty. However, as described previously, in the case where a topology management protocol is employed, the local topology information may already be known by each node. In protocols such as LTRT [MNAK09] nodes use the positions of their neighbors to compute their own view of the local topology and make local transmission power decisions, ultimately controlling the network topology. Similarly, networks supported by directional links (where nodes share position information) can use a similar distributed topology management protocol to determine where to point their directional links. The nodes in these networks already have the local topology information necessary to benefit from this enhancement without any extra overhead.

Since this local information is essentially only used to provide “hints” to the geographic routing constructs, it doesn’t need to match the actual global topology. If a packet starts face routing in a direction that looked promising given the local topology but is found to be bad, it will eventually turn around and go the other way. If a chosen bounding circle size is too small, it will eventually be increased. This is important because in the case where local topology management protocols are used, the local topology information does not always match the global topology.

## 4.9 Topology-Aware Greedy Routing

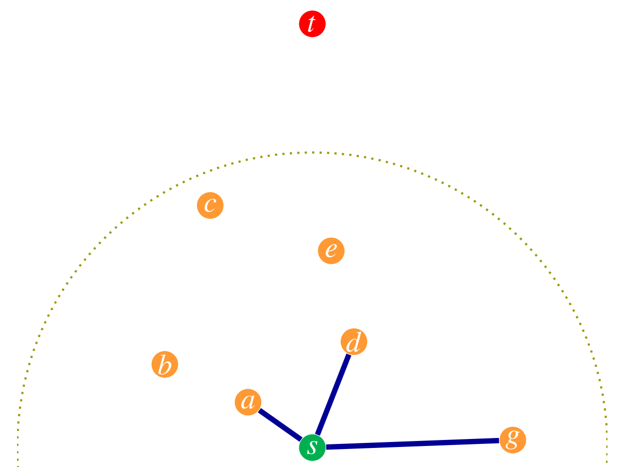
Enhancements to the base algorithm that utilize the local topology information are described in this and the following section (Section 4.10). One form of this local topology information is the by-product of the topology management protocol described in Chapter 3.

This local topology information can be used to make greedy routing more efficient. In some cases "pure" greedy routing will route a packet down a dead end. If the topology or an opinion of the topology, among nearby nodes is available, these extra hops can be avoided.

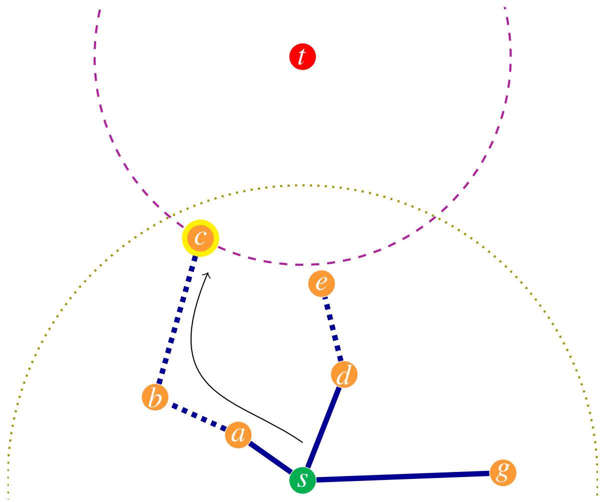
To determine the next topology-aware greedy hop for a packet, a forwarding node determines the set of nodes to which there is a path in the local topology. Next, the forwarding node determines which node in this set is nearest the destination (or is the destination), and computes the shortest path to this waypoint node. If the first hop along the computed path would advance the packet, causing it to move closer to the destination, then the packet is forwarded across that first hop. Otherwise, if the first hop doesn't advance toward the destination, the forwarding node removes the connected node nearest the destination from consideration and determines which node from the set is now nearest the destination (the second nearest overall). This continues until a next hop is found that makes progress, or it is determined that the local node is the nearest to the destination of all the nodes in its community to which it has a path in the local topology. In the latter case, the packet has reached a local minimum, and face routing takes over.

Figure 4.15 shows the decision process for the first topology-aware greedy hop taken in the same topology shown in Figure 4.3 and analyzed previously. Figure 4.15a shows the local view at the source node *s* for the example set of nodes. This local view includes only the topology connections incident at *s* and the positions of nodes in its community. When using "pure" greedy routing, only the positions of the directly connected neighbors are used to make the forwarding decision. In this example, it is easy to see that node *d* is closer to *t* than node *a* and would, therefore, be the greedy choice for the next hop. Unfortunately, as seen previously, forwarding toward *d* would lead the packet down a dead end, that would have to be retreated from, ultimately requiring four extra hops to reach the destination.

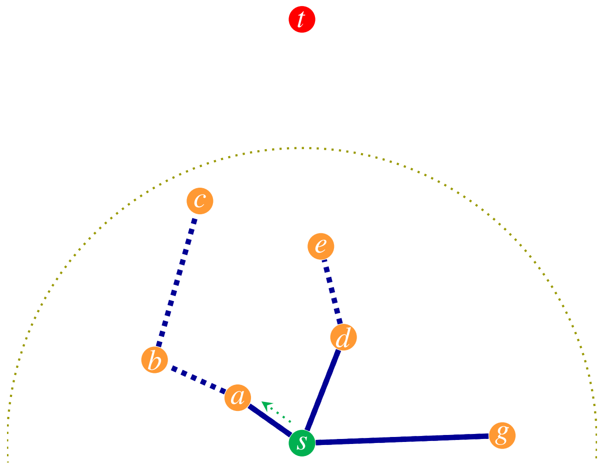
Consider, however, if topology-aware greedy routing were instead used. If the positions of other nearby



**(a) The local view at node $s$, including the positions of nodes of which it is aware (in its community), and the direct connections to $s$.**



**(b) Local topology among the community nodes. $c$ is the community node nearest to $t$, that has a path to $s$ in the local topology. The arrow shows the shortest path from $s$ to $c$. The dashed lines represent topology connections not incident to the forwarding node.**

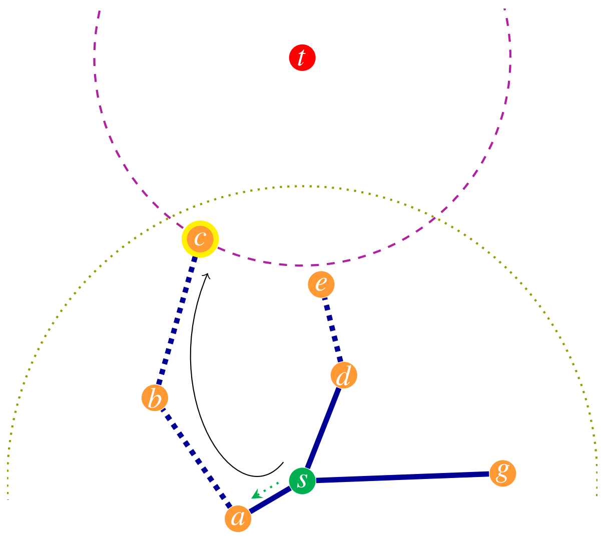


**(c) The first hop along the shortest path to $c$.**

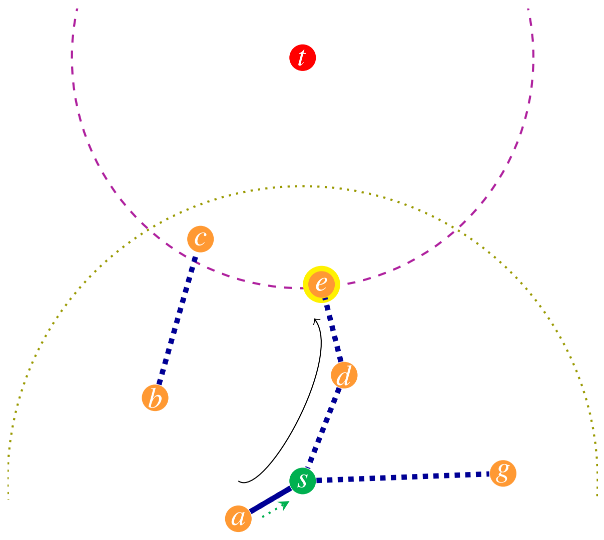
**Figure 4.15: An example motivating the advantages adding topology awareness to the normal Greedy mode.**

nodes were available, this position information could be used to construct an "opinion" of what the global topology likely looks like in the local area. This local topology is already frequently computed at each node when using the topology management framework. If a different topology management scheme were instead used, the local topology could be easily computed given the positions of nearby nodes. Figure 4.15b shows this forwarding decision process at node $s$. Notice that the topology connections not incident to the forwarding node ($s$) are dashed, indicating that they are only an opinion of the topology, and the actual topology could differ. Generally, however, the local topology matches the global topology fairly well, and it can be used to strategically make better greedy routing decisions.

The forwarding node first determines which of all the nodes to which it has a path in the local topology, is nearest to (or is) the destination. In this example, node $c$ (highlighted in the figure) is the node nearest the destination to which there is a path from node $s$. Next, the shortest path to this waypoint node is calculated and found to be $\{s, a, b, c\}$. Since $a$ is closer to $t$ than $s$, this hop satisfies the advancement requirement, and the packet is forwarded to $a$, as shown in Figure 4.15c. If the first hop towards $c$ did not advance towards the destination, then the first hop on the shortest path to the 2nd nearest node to the destination would be tested. Then the 3rd closest would be tested, and so forth until the first hop on the shortest path advances toward the destination.



**(a) The result if the next topology-aware greedy hop was not required to make progress. It would be routed toward $a$.**



**(b) If the edge between $a$ and $b$ didn't exist, then $a$ would have to forward the packet back to $s$, causing the packet to entered a routing loop.**

**Figure 4.16: An example that motivates the need for the next topology-aware greedy hop to make progress toward the destination.**

The requirement that the next greedy topology-aware hop is to a node that advances the packet toward the destination is important since the actual global topology of the network can differ from the local topology. Figure 4.16 shows a case where not requiring the next hop to advance towards the destination can result in a routing loop. In this example, node $a$ is further from the destination than node $s$. Also, for some reason, assume that the connection between $a$ and $b$ is not actually a part of the global topology, though, as shown in

Figure 4.16a, node *s* thought it should be and it was a part of *s*'s local topology. If the next topology-aware greedy hop was not required to advance the packet toward the destination, the chosen next hop would be toward *a*, since *c* is the nearest connected node to the destination *t*, and *a* is the first hop on the path to *c*.

Once the packet arrives at *a*, the local topology would not include the edge to *b*, and *a* would now attempt to use topology-aware greedy routing to route the packet given its local view of the network. According to *a*, node *e* is the node nearest the destination to which it has a path in the local topology. The first hop in the shortest path toward *e* is back to *s*. However, no state has changed at either node or in the packet header. Therefore, *s* forwards the packet right back to *a*, and the packet has entered a routing loop.

Assuming that *M* is the set of nodes in the same connected component as the forwarding node, and *t* is the destination node, the equation for the waypoint node or the connected node in the local topology nearest the destination is:

$$w(M,t) = \{\, m \in M \mid \text{dist}(m,t) \, \text{is a minimum} \,\} \tag{4.7}$$

## 4.10 Topology-Aware Face Routing

The local topology information can also be used to improve the performance of forwarding a packet in face routing mode. The node forwarding the packet can use its "opinion" of what the topology among nodes in its community should be, to determine the best direction to begin forwarding the packet around a face, and to determine the best initial radius for the circle bounding the exploration of the face boundary.

### 4.10.1 Initial Face Exploration Direction Choice

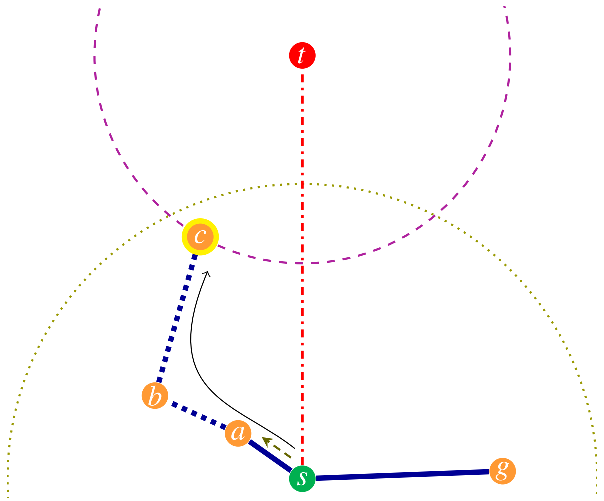
Remember, as shown in Figure 4.12, there can be cases where the choice of face boundary exploration direction can make a huge difference in terms of routing efficiency. Remember also that adaptively bounding the face exploration helps us avoid the worst case scenarios, but at a cost of often having to backtrack multiple times, increasing the number of hops in the average path. Without using extra information, upon reaching a local minimum, packets must still essentially guess the direction in which it is best to begin exploring the bounded face. Also, the initial bounding circle radius must likely be set based only on the current distance of the node from the destination. This leads to inefficiencies in the forwarding paths as sometimes the default initial circle radius is too small and must be expanded multiple times. The available local topology information can offer information to make better decisions about the initial exploration direction and the bounding circle radius.

To determine whether a packet should be forwarded in a clockwise or counter-clockwise direction around a face boundary, the forwarding node identifies the set of nodes in its community to which there is a path in the local topology. This set of nodes can also be thought of as the set of nodes in the same connected component as the forwarding node in its opinion of what the local topology among the community nodes should be. Just as in the greedy topology-aware mode, the node in this set nearest the destination is identified, and the shortest path to that node in the local topology is determined (using Dijkstra's algorithm). The forwarding node then attempts to determine which direction around the face best matches this shortest path. If one direction matches the identified shortest path better than the other, face routing proceeds, routing around the face in that direction. Otherwise, the packet defaults to routing in a clockwise direction.

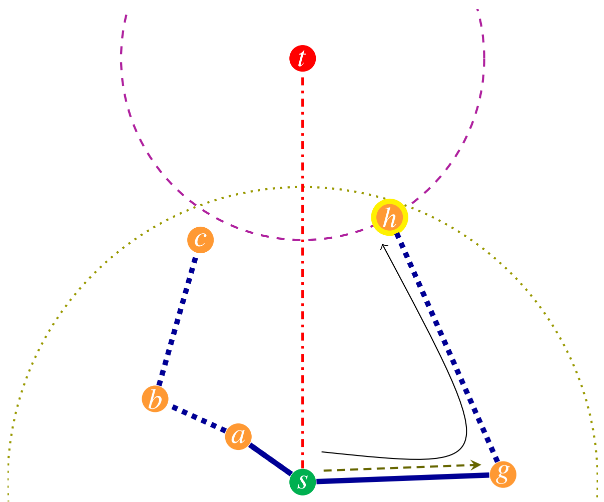
To determine which direction best matches the shortest path, the forwarding node reverses the predecessor list created by computing the shortest path to the node nearest the destination, yielding a successor list. This list allows the hops along the shortest path to be analyzed one at a time. Starting at the forwarding node, the options for the next node to be visited are determined. The nodes that would be visited going in both a clockwise direction and a counter-clockwise direction around the current face are considered. If the next hop along the shortest path computed matches the next hop around the face in a clockwise direction then the face routing direction is set to the clockwise direction. If, however, the next hop along the shortest path matches the next hop around the face in a counter-clockwise direction, the face routing direction is set to counter-clockwise.

Figure 4.17 shows examples of some of the cases possible when using this strategy. Figure 4.17a shows an example local topology similar to others studied in previous sections. Node $c$ (highlighted) is the node nearest the destination of which $s$ is aware and to which there is a path in the local topology. The shortest path to $c$ is shown by the curved arrow. The line between $s$ and $t$ is there only for reference and does not represent an actual connection. When determining the first face routing hop, one can imagine that the packet has just arrived across this imaginary connection to the destination, and use the right-hand rule as normal. In this case, using the right-hand rule (clockwise) the packet would next visit $a$. However, using the left-hand rule (counter-clockwise) the packet would next visit node $g$. The first hop on the shortest path is $a$. This matches the first hop when exploring the face in the clockwise direction, thus the exploration direction is set to clockwise and the packet is forwarded to node $a$.

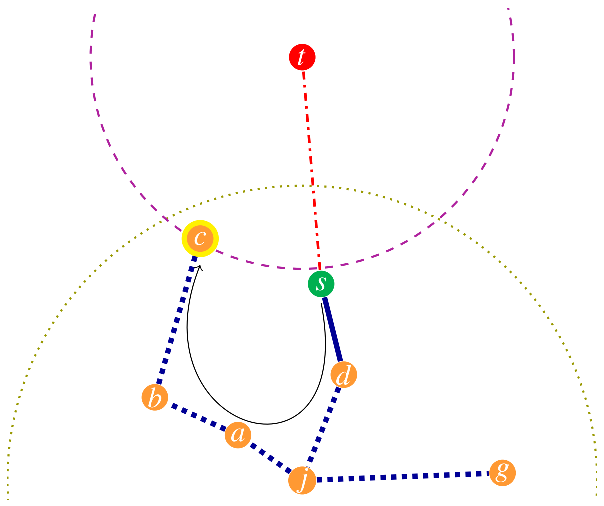
In contrast, if node $h$ and its connection to $g$ were added, as shown in Figure 4.17b, a different result



**(a) An example topology where a packet would begin routing around the face in a clockwise direction, given the topology information. The lines between $s$ and $t$ in these figures do not represent connections between the nodes, but are there only for reference.**



**(b) An example topology where a packet would begin routing around the face in a counter-clockwise direction, given the topology information.**



**(c) An example topology where the first hop in both the clockwise and counter-clockwise directions around the face leads to the same next hop node $d$.**

**Figure 4.17: Examples demonstrating selecting the face exploration direction based on the local topology.**

would be obtained. Node *h* is now the nearest node to the destination that is also reachable along a path from *s* in *s*'s local topology. The shortest path to *h* is shown by the curved arrow. Since the first hop along the shortest path matches the first hop while exploring the face boundary in the counter-clockwise direction, the direction of face boundary exploration for this phase of face routing is set to counter-clockwise.

It is possible, also, that both the counter-clockwise and clockwise directions around the face will visit the same node in the next hop. In this case, the second hops exploring the face in both directions in the forwarding node's local topology are compared with the second hop in the shortest path. If these too are the same, the third hops are compared, and so forth. This continues until either the destination is reached, or there is a divergence in the paths exploring the face in both directions. In the former case it doesn't matter which direction is taken, the result will be the same. In the latter case there are three possible outcomes: (1) the divergent hop in the shortest path matches the clockwise exploration path, (2) the divergent hop in the shortest path matches the counter-clockwise exploration path, and (3) the divergent hop in the shortest path matches neither direction's exploration path. In the first two cases, the direction is set equivalent to the direction in which the paths ultimately matched. In the last case, the shortest path does not go around the face, in which case the packet defaults to traveling in a clockwise direction.

Figure 4.17c shows an example where there is no difference between the path taken exploring the face boundary in a clockwise and counter-clockwise direction until the 3rd hop. The line segment from *s* to *t* is again just for reference and is not an actual edge in the network topology. Node *s* would first compute the shortest path to *c* (the nearest node to the destination to which *s* has a path in the local topology). Since the first face routing hop when exploring the face in either direction from *s* is *d*, the forwarding node must look farther along the path to develop an exploration direction preference. The next hop from node *d* exploring the face in either direction is *j*. The forwarding node still can deduce no preference for face exploration direction. The third hop exploring the face in both directions is determined, and finally, there is a difference. If the clockwise direction is used, the third hop around the face is to node *a*, as is the third hop in the shortest path to *c*. Therefore, the clockwise direction is likely the best direction in which to explore this face.

There is still one other case to cover, and that is if the node is the nearest node to the destination among all the nodes in its community. This situation will most likely only occur when the node density is low. In this case, one more heuristic is employed that has shown a small decrease in the mean number of hops between pairs of nodes. The forwarding node counts the number of nodes in the community to which there is no path in the local topology from the forwarding node. The number of such nodes on either side of the line from the forwarding node to the destination is determined. If there are more disconnected nodes to the left side of this line (from the point of view of the forwarding node facing the destination), then the packet is forwarded in a clockwise direction. Otherwise, the counter-clockwise direction is used.

### 4.10.2 Face Exploration Initial Bounding Circle Radius

The bounding circle radius is the other parameter controlling the face routing phase that can be intelligently set by making use of the available local topology information. Remember that the circle bounding the face exploration helps eliminate worst-case scenarios where there is a large disparity in the number of hops required to explore the face boundary in one direction versus the other. If the initial radius is set too small, the face boundary would be explored partially in both directions, and then the radius would have to be increased and the process repeated. If the radius were set too large, a larger portion of a face boundary may be inefficiently explored before hitting the boundary. Upon exploring in the opposite direction it is possible the packet may find it is able to reach the destination in only a small number of hops.

To determine the appropriate radius of the bounding circle, the forwarding node uses the same shortest path to the nearest connected node that was derived for computing the best exploration direction. The node along this shortest path that is furthest from the destination is identified, and the radius of the bounding circle is set to that node's distance from the destination plus some small safety factor. This safety factor seeks to ensure that even if the node is moving away from the packet's destination, the node will remain within the boundary until the packet has passed.

Figure 4.18 shows an example of how this works on an example topology. Having started at a local minimum at node *s*, the packet must now route around a face (that is only partially visible in the local topology). As in previous examples, node *c* is determined to be the nearest node of which *s* is aware, and to which it is connected. The shortest path to *c* is determined, and node *j* (highlighted) is determined to be the node furthest from the destination along the computed shortest path. The radius of the bounding circle for the packet starting face routing is then set to a distance slightly longer than the distance from *t* to *j*. The result is a bounding circle (shown just below *j* at the bottom of the figure) that includes the entire anticipated path of the packet around the face, as shown in the figure.

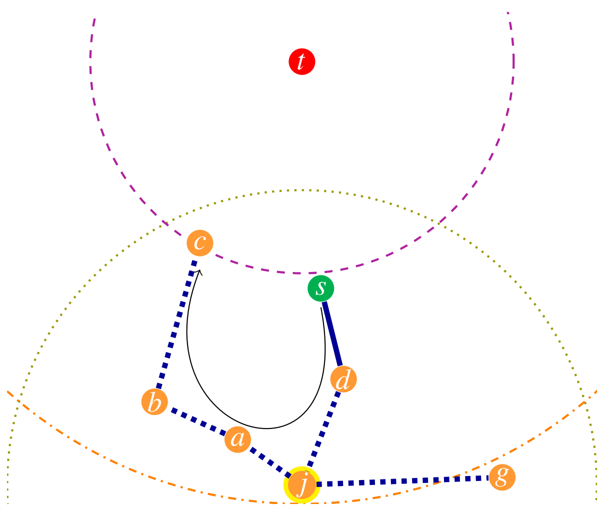


**Figure 4.18: An example showing how the radius of the bounding circle is determined.**

### 4.10.3 Implementation Summary

This section provides an overview of how the principles described above can be incorporated into the base implementation pseudocode presented in Section 4.7.

The topology-aware aspects can are inserted into the face routing function using the STARTFACE function. This function is called any time a packet is to begin routing around a face. Its purpose is to set up the header values correctly for face routing. The highlighted lines of pseudocode are the only lines added or changed to support topology awareness. As a reminder, two nodes are considered connected if the local topology graph contains a path between them, and two nodes are considered directly connected if they are connected with a path of length one hop. The FINDCONNECTEDNODENEARESTPOSITION function uses the available local topology graph to find and return the ID of the node nearest the destination to which there is a path from this node.

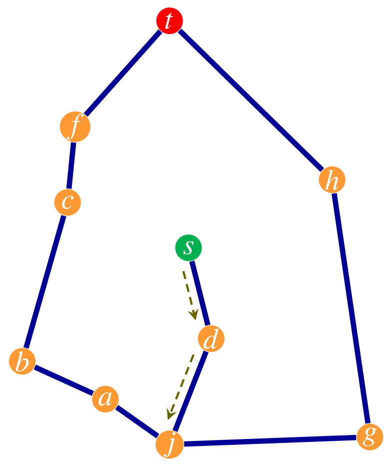
An additional function, RIGHTWAYTOREACH takes the nearest node ID and determines which direction (counter-clockwise or clockwise) face routing should proceed to efficiently reach the nearest node. This is far better than the base protocol, where faces boundaries were always initially traced in a clockwise direction. Lastly, the SMARTRADIUS function determines the appropriate radius value for the bounding circle. Previously, as in GOAFR+ this value was set to the product of an initial radius factor and the distance from the face start location to the destination location. The SMARTRADIUS function determines which node on the path to the "nearest" node is furthest from the destination and uses that distance (plus a small safety factor) as the radius value, ensuring that expected path can be traveled without increasing the radius. In cases where the “nearest” node is the current node, the radius is set such that the bounding circle encompasses the entire area within which the local topology is known (the community).

```
function SmartRadius(nearest, h.D)
    if nearest != self.id then
        return GreatestDistanceFromDestOnPathTo(nearest, h.D)
    end if
    return Dist(self.position,h.D) + communityRange
end function
```

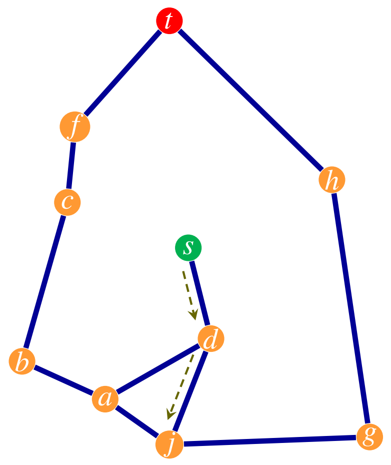
## 4.11 Dynamic Topology Support

Standard face routing can fail miserably when the movement of nodes causes changes in the topology. The example in Figure 4.19 shows how this can happen. Figure 4.19a shows a snapshot of the example topology at an initial point in time. Note that unlike some previous examples, here the entire global topology is plotted, though nodes aren’t aware of the global topology. A packet began face routing at node *s* and has explored the boundary of the face in the clockwise direction, visiting nodes *d*, and *j*. While the packet is at node *j* a new edge between nodes *a* and *d* is added to the topology as shown in Figure 4.19b. As the packet continues routing around the face it will visit node *a*, and then, blindly following the right-hand rule (taking the first right at the roundabout at *a*), it will travel across the new edge in the topology back to node *d*, as shown in Figure 4.19c. Since the packet doesn’t store a list of nodes it has visited, the packet is unaware that it has already visited this node. The packet would continue on to node *j* and then to *a* and would continue in this routing loop. Assuming there were no Time-to-Live (TTL) field associated with the packet, it would continue in this loop until the topology changed sufficiently and/or the nodes moved significantly.

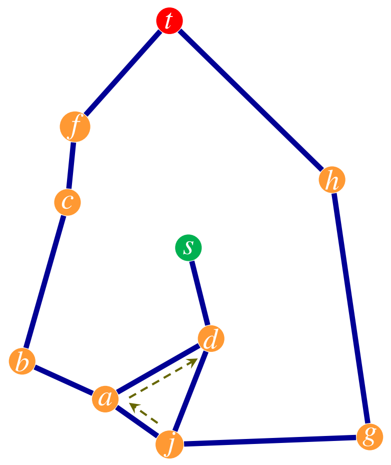
Most other geographic routing protocols assume a static or quasi-static network topology, suggesting that the routing is so fast compared to the node movement that no topology changes happen while a packet is in flight. This is obviously not realistic in a network with high link utilization, as there will always be packets flowing, some of which will be affected by a topology change. To our knowledge this work is the first to propose a greedy-face-greedy style geographic routing protocol that can operate correctly on topology graphs that are changing rapidly.



**(a) A packet is forwarded from $s$ to $d$ and then $j$ as it explores the boundary of the face.**



**(b) The connection from $a$ to $d$ is added to the topology while the packet is at node $j$.**



**(c) The packet then is forwarded toward $a$, and then over the new connection, toward $d$, following the right-hand rule. The packet is now stuck in a routing loop.**

**Figure 4.19: An example motivating the need for TAG to support fully dynamic topologies.**

### 4.11.1 Dynamic Topology Support Approach

As mentioned, TAG is able to overcome the issues with using face routing in dynamic topologies. The idea is to essentially take a snapshot of a face, in a distributed manner, each time a packet begins tracing a face boundary. The snapshot, not the potentially changed current topology, could then be used to determine how to forward the packet around the face even when the topology changes.

Consider again the example in Figure 4.19b. If node $a$ had such a snapshot, it could determine that the new connection between $a$ and $d$ did not exist when the packet began face routing. The packet could instead be forwarded to node $b$ as if the new connection didn't exist.

What if, however, instead of the link between $a$ and $d$ being added to the graph, it was instead taken away? Assume that Figure 4.19b shows the topology of the network at the time a packet starts routing around the face, and that Figure 4.19a shows the topology shortly thereafter. Assume further that the packet starts at node $s$, where it takes a virtual "snapshot" of the face that includes the connection from $a$ to $d$. While the packet is in flight toward node $d$, assume the link from $a$ to $d$ goes away. In this case, the packet is unable to explore the original boundary. What is needed is a way to get the packet to node $a$ and allow it to continue to traverse the face from there. Unfortunately, the direct path no longer exists. However, the packet could instead, take an indirect path to node $a$, traveling through node $j$. Section 4.11.3 gives the details on how this is done, but first the procedure to store the snapshot of the face boundary is described.

### 4.11.2 Storing the Snapshot

How can this virtual snapshot of a face boundary be implemented in a distributed manner? First, a time field ($t_{\text{faceStart}}$) is added to the TAG header that corresponds to the time at which the current face boundary began being traced. Second, every node maintains a table containing the time-stamped position and ID of the last node each of its links were directly connected to. The amount of storage required for this table is minimal and is limited by the maximum degree of each node. When a node needs to determine where to forward a packet as it travels around a face, it need only use the time stored in the packet to look up the IDs and positions of the nodes it was connected to at the given time. The forwarding node can then make the routing decision it would have made at the time the face routing began, and the virtual "snapshot" was taken.

### 4.11.3 Via-Points

The proposed method works until the rare situation is encountered where a node is no longer directly connected to the next node on the face snapshot. This is overcome by noting that the routing protocol allows a packet to be routed to any node whose position and ID are known. Since the position and ID of the next node along the boundary of the snapshot face are known, the ultimate destination is overridden, briefly, and the new focus of the packet becomes reaching the next intended stop along the face, called the "via-point". The shim header becomes a stack of shim headers, and a new header whose destination is the via-point is pushed on the stack. The packet is now forwarded like any new packet, toward the via-point. Once the via-point is reached, the header is popped off, and the packet is ready to continue routing around the original face, having "virtually" traced the face edge that no longer exists.

## 4.12 Topologies that are not unit disk graphs

Previously it was mentioned that TAG's implementation had a flaw when topologies are non-unit disk graphs and certain topology generation algorithms are used. In this section, the strategy for overcoming this flaw is explained. Remember that a unit disk graph contains an edge between any pair of nodes whose separation distance is lower than some maximum range. Topologies facilitated by omnidirectional data links can be theoretically modeled by a unit disk graph, however, in reality, such assumptions of connectivity are flawed because they discount real-world effects, such as physical blockages attenuating the signal and breaking the connection between a pair of nodes. Similarly, the topologies produced by the Topology Management framework described in Chapter 3 are not unit disk graphs.

The face routing algorithm described above assumes that at some point while traveling around any face one of three events will occur: (1) the packet will arrive at the destination, (2) the packet will arrive at a node that is closer to the destination than the node at which face routing began, or (3) only if there is no route to the destination, the packet will arrive back at the node at which it started, having explored the entire face boundary. The implied assumption in these cases is that either there is a node closer to the destination on the face that stands between the start node and the destination, or the destination is unreachable. This assumption is true in a unit disk graph, but what of the scenario shown in Figure 4.20 where $s$ is the source of a packet, $t$ is the destination, and the plotted circle encompasses the space that is closer to $t$ than $s$? Note that $b$ and $c$ are both outside the large circle, and both are further from $t$ than $s$. If this were a unit disk graph, the fact that $b$ is connected to $t$ would imply that $s$ should also be connected to $t$ since it is closer (by definition of unit disk graph). Note also that $s$ is a local minimum. Since this topology is not a unit disk graph, tracing the boundary of the face from $s$ to $b$ to $c$ brings the packet back to $s$ without finding any node that is closer to $t$. This violates the assumption of the proposed routing scheme since there is a route to the destination through $b$ or $c$.

To overcome this issue three lines of code are added to the FACECOMMON function (10-12 highlighted in pseudo code shown previously in Section 4.7). As the packet moves around the face, the line segment representing the next edge to be traversed is intersected with the line segment extending from the face start position to the destination position in a manner similar to basic face routing [KSU99]. If an intersection

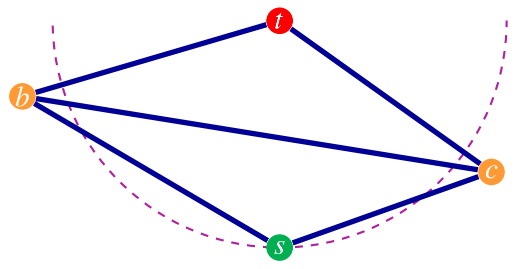


**Figure 4.20: An example of a topology that violates the Unit Disk Graph assumption.**

is detected, face routing is simply restarted at the current node. The packet will then seek to navigate the face between the current node and the destination. Depending on the geometry of the graph, the next face traversed may not be the next face encountered by the line that intersected the edge, however, progress towards the destination has been made. This very simple change to the algorithm allows it to conquer the potential issues with non-unit disk graphs. This technique ensures that even if no nodes around a face boundary are closer to the destination than the face start node, the packet will not be dropped incorrectly, but will instead appropriately transition to the next face.

This feature can be enabled or disabled in TAG, depending on whether or not the chosen topology generation algorithm generates topologies with these pathologies. For most of the simulations in this work, this feature was disabled and the DC-GG topology generation algorithm was used.

## 4.13 Simulation Results and Evaluation

To evaluate TAG, the results of a few simulations are presented here that demonstrate the advantages of TAG compared with other geographic and MANET routing protocols. More simulation results of various experiments using the topology management framework and TAG are presented in Chapter 5. In this chapter, first, the results of simulating the protocol in a theoretical square field (as in [KWZ03b]) are reported, randomly placing static nodes on a plane in a square measuring 20 units on each side. Next, a study comparing TAG and OLSR is described. Finally, results from a more realistic simulation are presented, where mobile nodes trace the paths traveled by over 600 actual aircraft moving at hundreds of miles per hour.

### 4.13.1 Simulation Environment

The ns-3 [RH10] network simulator (version 3.23 and 3.26) is used for the evaluation experiments. The default ns-3 implementation of OLSR is used. Since these protocols are not included in ns-3, a custom implementation of GFG and GOAFR+ developed for this work are used. Since the goal at this stage is to measure the performance of the routing protocol without interference from the environment or other layers of the stack, perfect wireless links and a collisionless MAC layer are assumed. Perfect links imply that no packets are dropped at the link layer and that the directional links are able to re-point and reconnect instantly. Instant re-connection could essentially be accomplished by doubling the number of links and allowing one to remain connected while its counterpart prepared to establish the next connection. A collisionless MAC layer is also not so unrealistic for directional links where interference is less common given the directional nature of the connections. Further, it is assumed (as in [KWZ03b]) that all position information required by the geometric routing protocol is available without extra communication overhead. This includes a node's knowledge of its position and the positions of nearby nodes (nodes within the position exchange range of 288 km or 1.44 units), as well as each source node's knowledge of the destination positions of its flows. Remember, for the airborne network application, the positions of nearby nodes are already regularly updated via an independent FAA-required system. Finally, all nodes are assumed to have a synchronized time reference, such as that available via GPS.

Table 4.2 lists the pertinent input parameters and simulation default values.

**TABLE 4.2: The TAG parameters for this simulation.**

| | | |
|---|---|---|
| $\rho_0$ | Initial Bounding Circle Radius (non-topology-aware version only) | 1.4 |
| $\rho$ | radius increase factor | $\sqrt{2}$ |
| $c$ | position exchange range | 1.44 or 288 km |
| $l$ | maximum link range | 1.0 or 200 km |
| $n$ | links per node (degree) | 3 |
| $u$ | topology update rate | once per second |

### 4.13.2 Varied Density with a Static Topology

First the performance of TAG, GOAFR+($\rho_0$=1.4, $\sigma=\frac{1}{100}$), and GFG are compared on a static topology using the hop stretch and path stretch metrics [LML05]. Hop stretch is the ratio of the number of hops required by a routing protocol to reach the destination $t$ to the number of hops on the shortest path (in terms of hops) between $s$ and $t$. Similarly, path stretch is the ratio of the Euclidean length of the path taken by a packet to the Euclidean length of the shortest path between $s$ and $t$. Lower path and hop stretch values imply better performance for the geographic routing protocol.

For this experiment the number of nodes is increased from a density value of 1 to a density value of 20 nodes per unit disk. As a reminder, a density value of 5 implies that on average each node will have 4 other nodes within communication range (within its unit disk). For reference, in this experiment, the density values 1 and 20 correspond to 128 and 2547 total nodes, respectively. For each run, a source node, $s$, and a destination node, $t$, were randomly selected from the simulated nodes. A single UDP packet was then sent from the selected source node towards the destination node. The number of hops taken by the simulated routing protocol was recorded, as were the number of hops in the shortest path from the source to the destination. Figure 4.21 shows a plot of the resulting hop stretch values for the various protocols.

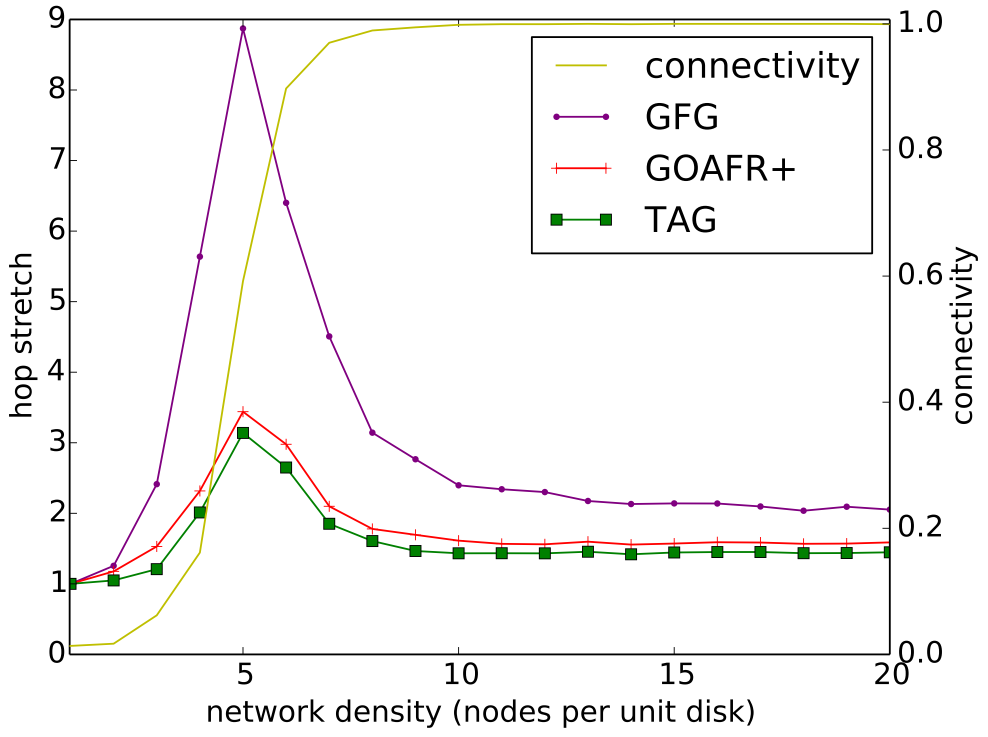


**Figure 4.21: A comparison of hop stretch values as node density increases.**

Each point on the figure represents the mean value for 2,000 trials (each with a random set of node positions and source, destination pair). Also plotted on the right y-axis is the connectivity of the network at the given densities. The results can be divided into three separate regions. The first region is that below a critical density of about five nodes per unit disk. In this region, less than half of the random node pairs have a path between them. Nodes that are able to connect to one another at these densities generally are within a few hops of one another. Hop stretch values for disconnected source/destination pairs are not included, yielding low hop stretch values for the lower densities. The second region spans from about 5 to 8 nodes per unit disk. In this region, a majority of source/destination pairs are connected, but the hop stretch values spike. This is because, although the nodes are connected, any path connecting them must wind its way through the "nearly disconnected" network. As the density increases, the hop stretch values rapidly decline as more and better paths become available. Finally, in the third region, the hop stretch values basically level off. TAG makes a slight improvement on the performance state-of-the-art GOAFR+ protocol and performs significantly better than the basic Greedy-Face-Greedy protocol (GFG). At a node density value of 20 nodes per unit disk, the hop stretch for TAG is 1.45, compared with 1.59 for GOAFR+ and 2.05 for GFG. This improvement is credited to the effective use of the local topology information at each node.

Figure 4.22 shows the path stretch results for the same experiments. The plot resembles the hop stretch plot, but GFG has an even larger spike near the critical density, so the range of the y-axis has been limited. The mean path stretch values at a density of 20 nodes per unit disk are 1.32, 1.45, and 3.14 for TAG, GOAFR+, and GFG respectively. At this density packets being routed with TAG can expect to travel 32% further than the optimal path.

### 4.13.3 Improvement Credited to Topology Awareness

How much impact does having an opinion of the local topology have on the hop stretch? To answer this question a version of TAG is simulated, but in this case, the topology aware feature has been disabled. Figure 4.23 allows the hop stretch for these two versions of TAG to be easily compared. Note the significant improvement that can be credited to the topology awareness aspects of TAG. At a density of 20, TAG-NoTopoAware has a mean hop stretch of 2.15, compared to 1.45 for TAG.

### 4.13.4 Comparison with OLSR

A simple experiment is now described comparing the TAG routing and Optimized Link State Routing (OLSR) [JMC$^{+}$01]. Remember, as described in Section 2.1.5.1, OLSR is a proactive MANET routing protocol and as such, it attempts to maintain routes between every pair of nodes in the network. To be fair to OLSR a small network is created in which flows are simulated between every pair of nodes.

Twenty-four mobile nodes tracing the actual paths taken by aircraft on July 9, 2015, are simulated. The topology management protocol described in Chapter 3 is used to form a network topology assuming that

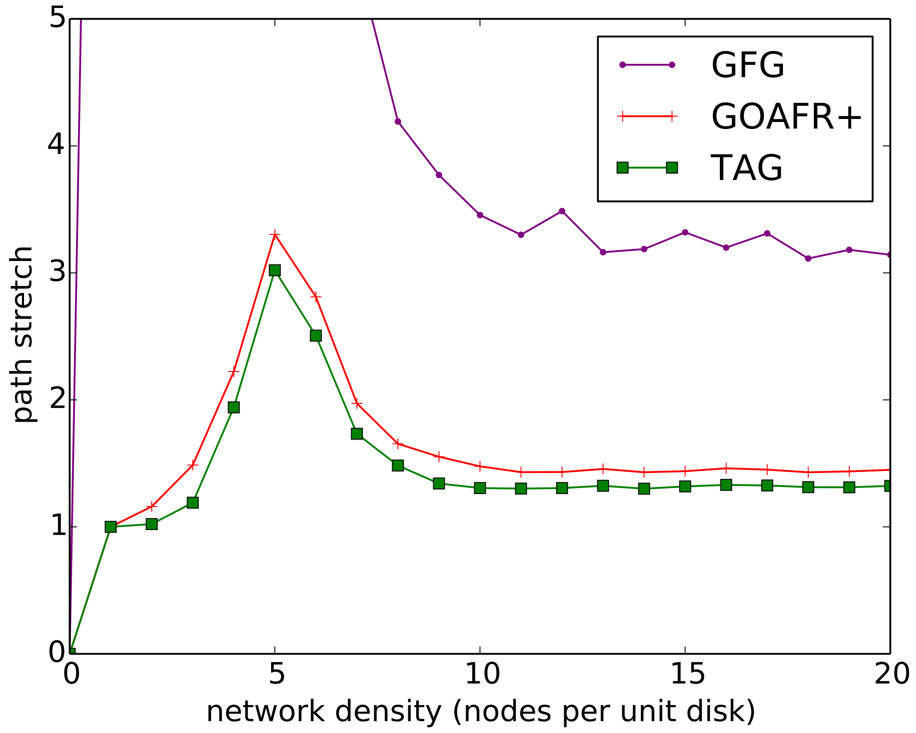


**Figure 4.22: A comparison of path stretch values as node density increases.**

three directional links are mounted on each aircraft each having a maximum air-to-air link range of 200 km. The node movement and topology management are such that there always exists a topological path between every pair of nodes in the network. A low-rate (10 Kbps) UDP flow between every pair of mobile nodes in the network is simulated for 1 hour. The default Hello interval of 2.0 seconds is used for OLSR.

The results are summarized in Table 4.3. TAG successfully delivers every packet achieving a PDR of 1.0, while OLSR is only able to successfully deliver 94.2% of the packets (a PDR of 0.942). In addition, OLSR sends over 281,000 overhead packets (a total of 53 Megabytes). Tag has no overhead outside of the small additional header attached to each packet. Similar OLSR experiments with about 600 nodes have yielded PDRs as low as 0.44.

### 4.13.5 Realistic Airborne Network with Mobility

Results are now presented for another experiment which simulates data flowing through a network of hundreds of simulated aircraft, whose positions and movements match those recorded from actual commercial aircraft flying above the United States on July 9, 2015. The performance of several geographic routing

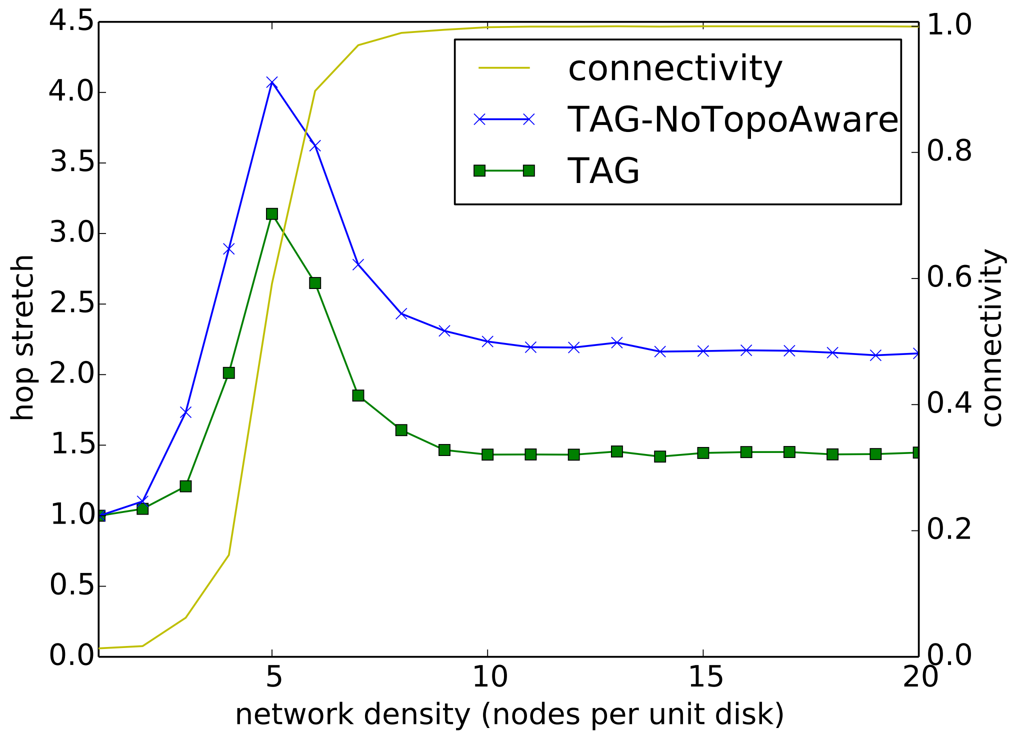


**Figure 4.23: A comparison of the hop stretch values with and without topology awareness.**

**TABLE 4.3: A routing comparison of OLSR and TAG.**

| protocol | packet delivery ratio (PDR) | routing management packets (overhead packets) |
|---|---|---|
| OLSR | 0.942 | 281,000 |
| TAG | 1.0 | 0<br>(assuming nearby nodes exchange position info) |

protocols are compared in this environment.

Table 4.4 shows the simulation parameters for this experiment. Since the objective is to simulate a network of commercial aircraft in the United States that join the network after take-off, we first remove from the data set all general aviation aircraft, all aircraft position samples below 10,000 feet, and all aircraft position samples further than about 200 km from the border of the continental United States. Next, since the goal is to test the routing protocol, not necessarily the connectivity of the topology protocol, a set of nodes are selected that are guaranteed to remain connected (with a path between every pair of nodes) during the duration of each simulation, given that the topology management framework is used with a maximum link range of 200 km and 3 links per aircraft. Nodes are allowed to join the network over the course of the simulation, but are not allowed to leave. If nodes were allowed to leave, packets queued at a departing node could be lost. The largest subset of aircraft meeting these conditions are selected from the 1 hour period between 1:00 p.m. and 2:00 p.m. Central time on July 9, 2015. The subset contains 200 nodes at 1:00 p.m. and an additional 473 nodes are added during the 1 hour simulation, yielding 673 total nodes when the simulation ends. Thus, the density of the network increases as the simulation progresses from 3.11 to 10.47 nodes per unit disk (note, however, that the density metric calculation generally assumes aircraft are spread across the country, which is not the case here, such that the actual density is greater). Figure 4.24 shows the network at the beginning of the one-hour simulation, and Figure 4.25 shows that expanded network which covers over half of the country at the end of the 1-hour time period.

**TABLE 4.4: The simulation parameters for the dynamic airborne network experiment.**

| | | |
|---|---|---|
| routing params | Routing Protocols | TAG, GOAFR+, GFG, and TAG without topology awareness |
| | TAG radius increase factor | $\sqrt{2}$ |
| network device params | Network Device Type | wireless point-to-point |
| | maximum link range | 200 km |
| | links per node (degree) | 3 |
| | air-to-air link rate | 10 Gbps |
| | propagation speed | speed of light |
| topology management params | position exchange range (community range) | 288 km |
| | topology update rate | once per second |
| | topology generation algorithm | DC-TRT |
| Internet Protocol (IP) prams | time-to-live (TTL) | disabled |
| flow params | flow style | 1 flow each per 1,000 simulations |
| node mobility params | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 9, 2015, 01:00 p.m. CT |
| | duration | 1 hour |
| | number of nodes | 200 growing to 673 nodes |
| | density | 3.11 to 10.47 nodes per unit disk |
| application params | application type | UDP Constant Bit Rate (CBR) |
| | flow rate | 10 kbps |

In striving for perfect PDR, the number of hops each packet may take is not limited (the IP TTL countdown is deactivated). Figure 4.26 shows the cumulative distribution function (CDF) of the mean numbers of hops for each flow. One thousand flows are simulated, each in a separate experiment, to rule out the possibility of one flow interfering with another during the simulations. The curve for the optimal route (given a

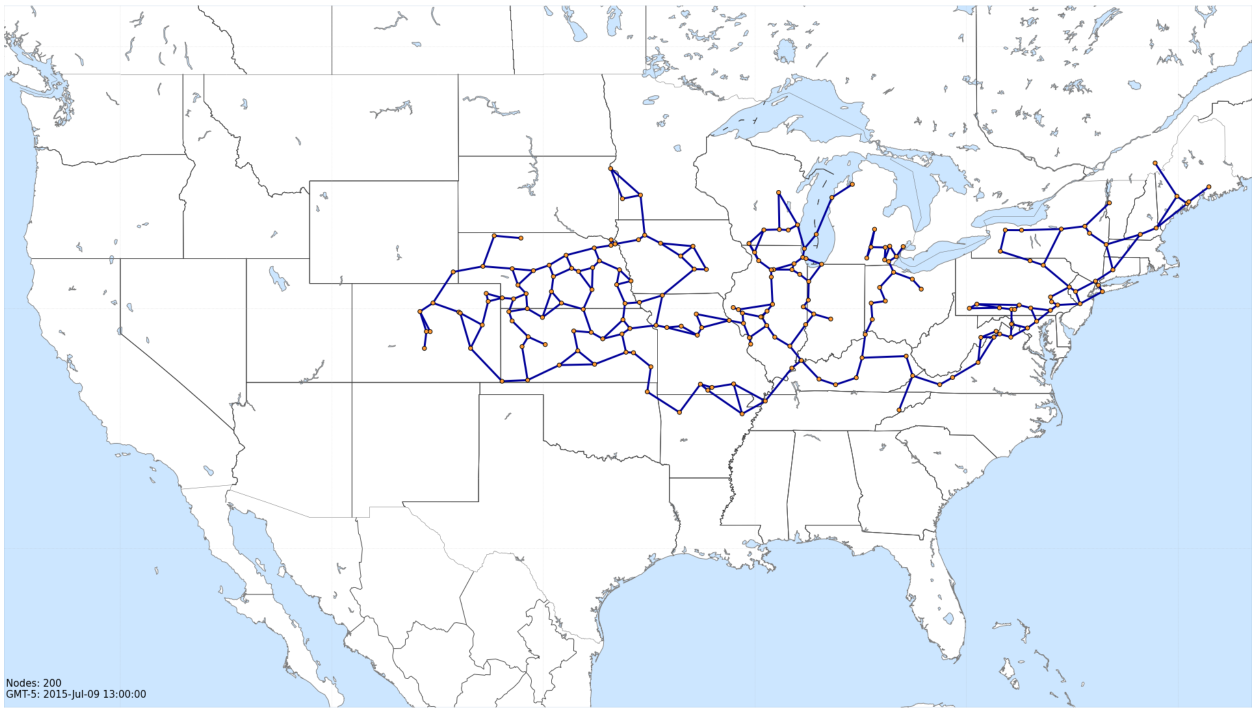


**Figure 4.24: The network of aircraft used for the dynamic node simulation. A subset of 200 aircraft over the United States at 1:00 p.m. July 9, 2015.**

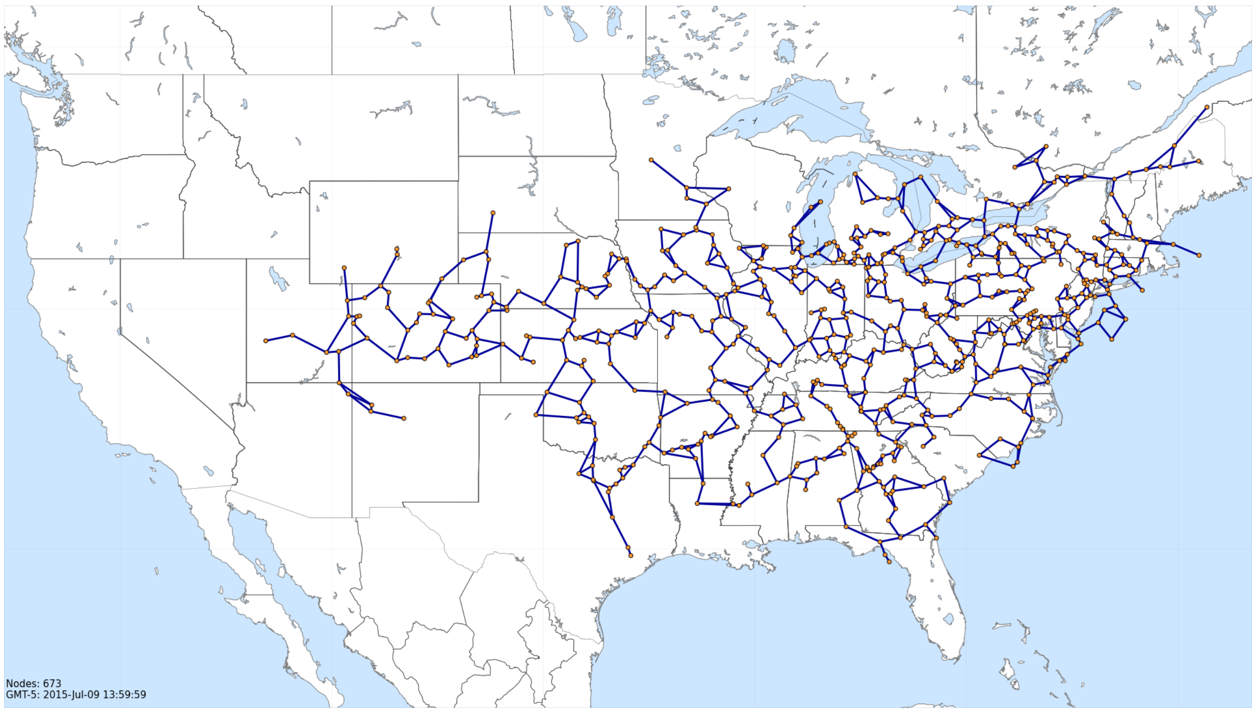


**Figure 4.25: The network of aircraft used for the dynamic node simulation. A subset of 673 aircraft over the United States at 1:59:59 p.m. July 9, 2015.**

global view of the network) is also plotted for comparison. TAG performs better than GOAFR+, especially in the tail of the distribution, with the worst performing flow requiring only 111 hops on average for TAG, but over 2,000 hops for GOAFR+ and nearly 3,000 hops for GFG. The high numbers of hops are a result of either situations where a packet gets stuck in a loop until the topology changes allowing progress, or cases where the basic geographic protocols make bad choices or must backtrack several times while expanding the bounding circle. Also notable is the fact that many flows using GOAFR+ were unable to guarantee delivery, with some achieving PDRs lower than 0.96. The large numbers of hops, in the worst case, highlight how helpful a small amount of local topology information can be, and the importance of fully supporting dynamic network topologies.

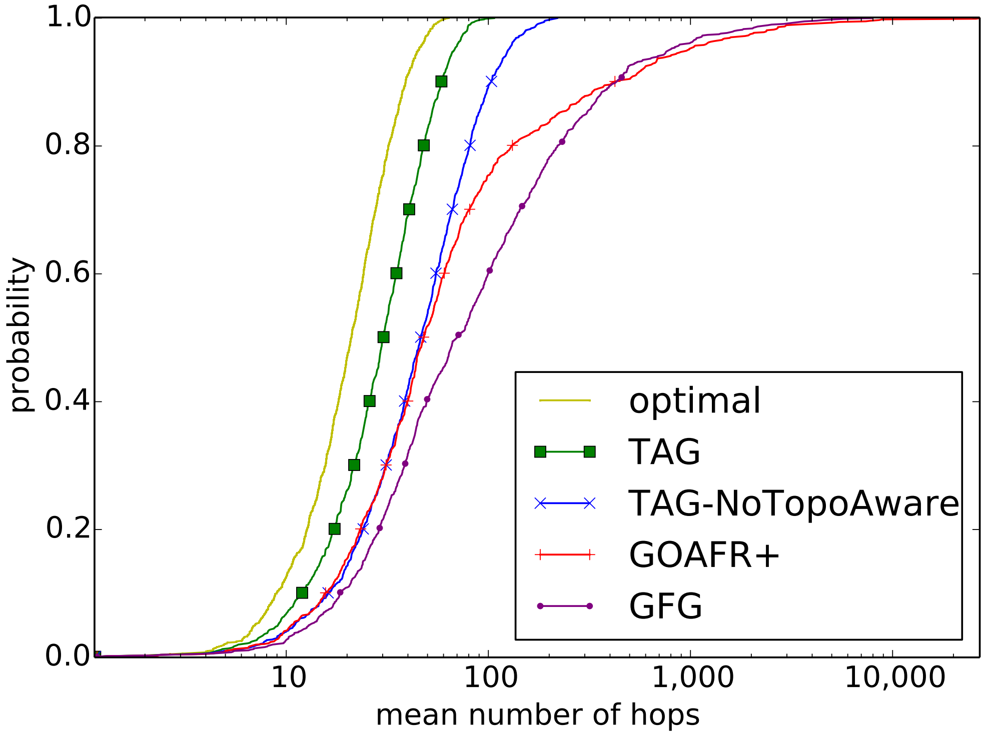


**Figure 4.26: A plot of the CDF of the mean numbers of hops in an airborne network including hundreds of aircraft. (1 low-rate UDP flow per experiment, 1,000 experiments, 3 links per node, 1 hour simulated per experiment)**

## 4.14 Summary

TAG is a new geographic routing protocol that advances the state of the art by (1) supporting topologies that are not unit disk graphs, (2) using local (potentially unreliable) topology information to make better geographic forwarding decisions, and (3) fully supporting dynamic topologies without the risk of forming routing loops. In this chapter, the routing protocol was detailed, and then simulated comparisons of TAG in a theoretical network and a real-world large-scale mobile airborne network were presented. TAG outperforms GOAFR+, GFG, and OLSR in the simulations.

# CHAPTER 5: AIRBORNE NETWORK SIMULATIONS AND RESULTS

Having now examined the topology management framework and topology aware geographic routing in the previous chapters, this chapter presents the results of a series of simulations that demonstrate the capabilities of these protocols working together to manage and route traffic through the airborne network. In addition, this chapter gives an overview of the implementation of the simulation models developed in the ns-3 simulator to simulate these airborne networks and the supporting protocols.

## 5.1 Simulation Environment

The simulations described in Chapter 4, and those to be discussed in this chapter, use the ns-3 network simulator. Ns-3 is an open-source discrete event network simulation tool used for education and research. Several new ns-3 models had to be designed and implemented to support the simulation of airborne networks. In this section, the various components added to ns-3 to support our airborne network simulations are described.

### 5.1.1 Aircraft Mobility

Most other mobile networking simulations use some sort of mobility model to determine the movements of nodes in a simulation. Rather than use such a model to simulate aircraft mobility, the simulations in this work use previously collected flight path data to model the movements of aircraft in the network. This makes the simulations all the more realistic but also requires that on solve all the issues accompanying the use of real-world data. To accommodate incorporating this data into our simulations a new ns-3 model was created.

As mentioned in Section 2.1.2.2, the Federal Aviation Administration (FAA) makes available to some industry and research partners, aircraft position information for most of the aircraft in the United States National Airspace System (NAS). This data, called Aircraft Situation Display to Industry (ASDI) [Vol00], includes not only the position and velocity of each aircraft but also other information such as the flight call sign and flight plan information. The simulations in this work make use of the ASDI data recorded during the month of July 2015. The ASDI data set includes position updates about once per minute for each aircraft, though more precision is available in real-time. Since the time resolution required for networking is much finer (milliseconds or less), linear interpolation is used to obtain the expected positions of aircraft between position samples in the data set.

The Tracktable open-source library [RW15] is used in a new aircraft mobility model to read and parse the ASDI data files. These files contain a list of trajectories each of which is made up of a list of potentially many trajectory points. Each trajectory point contains at least an aircraft's call sign, a time stamp (including the date), and the geodetic position of the aircraft (latitude, longitude, altitude) at the given time. This geodetic position is also translated into a Cartesian coordinate system called Earth-centered Earth-fixed (ECEF).

The ECEF coordinate system has its origin (0,0,0) at the center of the earth's mass and is fixed to the earth (rotating with the earth as it spins). Calculating and storing the ECEF positions allows for quickly determining the distance between two aircraft, using normal Cartesian methods. When determining the distance between nodes in the simulation, the ECEF positions are used, but when computing the topology, the latitude and longitude values are used.

The mobility model uses ns-3's existing constant velocity mobility model to schedule the mobility of the nodes. It first sets appropriate initial positions for each aircraft. Then, as a pre-processing step before a simulation begins, the mobility model schedules velocity changes at the appropriate times. Ns-3's constant velocity mobility model handles efficiently determining the interpolated node positions as the simulation runs. To accurately determine what velocity to set when the aircraft arrives at waypoint $n$, waypoint $n$+1 is used. The velocity (including both speed and direction) required to travel from waypoint $n$ and reach waypoint $n$+1 at exactly the time associated with waypoint $n$+1 is determined. An event is then scheduled at the time associated with the arrival at waypoint $n$ that updates the node's velocity to the calculated value. The velocities are computed and scheduled in this way for each waypoint of each flight. The aircraft mobiltiy model also handles the extra processing required to accommodate nodes that enter and/or leave the simulation.

### 5.1.2 Wireless Point-to-Point Channel

Ns-3 has built-in support for wired point-to-point links and for various wireless link types, but not necessarily for directional wireless data links. These links have some attributes of wireless links. For example, they may communicate with any other node within range (assuming they are pointed correctly).

However, they also share attributes with point-to-point connections in that network devices (links) are paired and communication is generally limited such than only paired network devices may exchange data. What is needed is a wireless point-to-point link type combining some of the characteristics of both point-to-point links and wireless links.

For this work, a new model was added to ns-3 to simulate the unique characteristics of these directional wireless connections. Unlike omnidirectional channels that send packets to all the Network Devices attached to the channel, this new channel model need only send a packet to the Network Device whose antenna is connected with, and pointing at, this device. To accomplish this, the channel stores a connection map that indicates which Network Devices are connected at the current time.

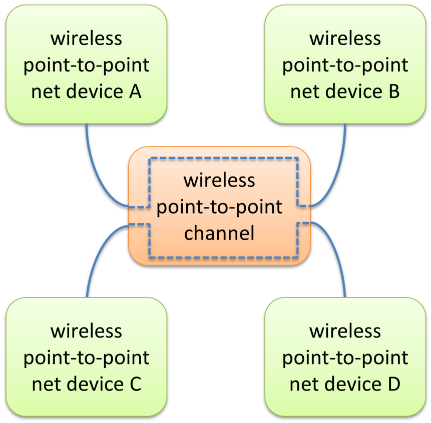

wireless point-to-point net device A
wireless point-to-point net device B
wireless point-to-point channel
wireless point-to-point net device C
wireless point-to-point net device D

**Figure 5.1: Diagram showing the relationship of the components that enable simulating wireless point to point links.**

Each time a packet enters the channel it determines which network device, if any, is connected with (pointing at) the source Network Device. The receipt of the packet is then scheduled on the destination network device at some future time, where the future time accounts for the transmission time and the propagation delay. Figure 5.1 shows a diagram of how these components work together. Many wireless point-to-point network devices can be connected to a single wireless point-to-point channel. When two network devices are connected (as net devices A and B or C and D are in the figure), packets sent from one device arrive at the other, and vice versa. This mapping can change over time as commanded by the topology management application (described shortly), with network devices disconnecting and re-connecting in the channel as the network evolves. Packets sent by network devices that are not connected to another network device are dropped. For these simulations, the topology management application explicitly controls the connection map in the wireless point-to-point channel. A more realistic model could be employed in the future where the current antenna pointing direction could be taken into account, and packets could be dropped unless antennas are correctly aligned.

### 5.1.3 Wireless Point-to-point Network Device

The wireless point-to-point network device model is much the same as ns-3's traditional point-to-point network device except that the wireless point-to-point protocol header contains two extra fields. These fields are the sending and receiving media access control (MAC) addresses. When sending a packet the network device enqueues the packet in the transmit queue. Then, if there are no other packets in the queue, the network device begins simulating the transmission of the packet, sending it to the channel for distribution to the connected network device, if any. When receiving a packet the network device determines whether the packet was corrupted, given the error model (bit error rate). If the packet was corrupted it is discarded, and otherwise, it is forwarded to the next level in the stack.

### 5.1.4 Topology Management Application

The topology management application is run on each node to control the topology of the network in a distributed manner. Each node periodically, but in sync with other nodes, generates an opinion topology and points its links accordingly, as described in Chapter 3.

### 5.1.5 Steerable Directional Antenna Model

Steerable directional antennas often employ a tracking system to automatically adjust the pointing of the antennas once connected. These simulations assume the existence of such a tracking system. Upon creation of a connection between two nodes, where the connection did not exist in the previous iteration, each node explicitly informs the channel of its intention to connect to the remote node. Once both nodes have connected, the connection table in the channel is updated to reflect the connection, and allow packets to flow.

## 5.2 Simulations

In the remainder of this chapter details a variety of experiments that use the topology management framework described in Chapter 3 and the geographic routing protocol (TAG) described in Chapter 4. These experiments test the ability of the protocols to deal with high-density scenarios, high rate flows, and long durations, just to name a few. Since the amount of computation and memory required in the simulation for a single long duration, very high-rate, very large-scale, high-fidelity experiment is prohibitive, one goal of these simulations is to test these various aspects independently such that confidence can be built in the effectiveness of the full-functioning system without being able to simulate it explicitly.

### 5.2.1 Default Simulation Parameters

Table 5.1 lists the default simulation parameters. The following parameter values should be assumed in the simulations described below unless otherwise stated.

-

**TABLE 5.1: A table listing the default simulation parameters.**

| | | |
|---|---|---|
| routing params | routing protocol/algorithm | TAG |
| | TAG radius increase factor | 1.5 |
| network device params | Network Device Type | wireless point-to-point |
| | maximum link range | 200 km |
| | links per node (degree) | 3 |
| | transmit queue type | drop tail queue |
| | transmit queue max size | 100 packets |
| | air-to-air link rate | 10 Gbps |
| | maximum transmission unit (MTU) | 1500 bytes |
| | propagation speed | speed of light |
| topology management params | position exchange range (community range) | 288 km |
| | topology update rate | once per second |
| | topology generation algorithm | DC-GG |
| Internet Protocol (IP) params | time-to-live (TTL) | enabled, 255 |

Unless otherwise stated, the aircraft trajectories used in these simulations have been filtered to remove trajectories for aircraft with general aviation call signs and to avoid portions of trajectories that are below 10,000 feet elevation, or outside of the established continental United States boundary (about 200 km off the coast).

### 5.2.2 Global Routing Comparison

This experiment compares the performance of the Topology Aware Geographic Routing protocol (TAG) to that of an ideal routing protocol, in a static network topology formed among real aircraft positions at some point in time. The ideal routing protocol used for the experiment is ns-3's Global Routing protocol.

It computes shortest path unit cost routes between every pair of nodes in the network before the simulation begins and adds the appropriate static route entries to the routing table at each node. The experiment consists of two simulations. In the first simulation, the topology management framework is initially used to establish the static network topology and TAG is then used to route traffic among the set of aircraft. For the second simulation, however, the wireless-point-to-point links in the topology are replaced by normal point-to-point links and ns-3's Global Routing model is used to pre-compute static routes between every pair of nodes in the static network topology. At run-time, the packets follow these shortest path routes that are stored in the routing table associated with each node. The intent of the experiment is to be able to gauge TAG's performance in terms of delay and numbers of hops, compared with the ideal routing protocol's performance.

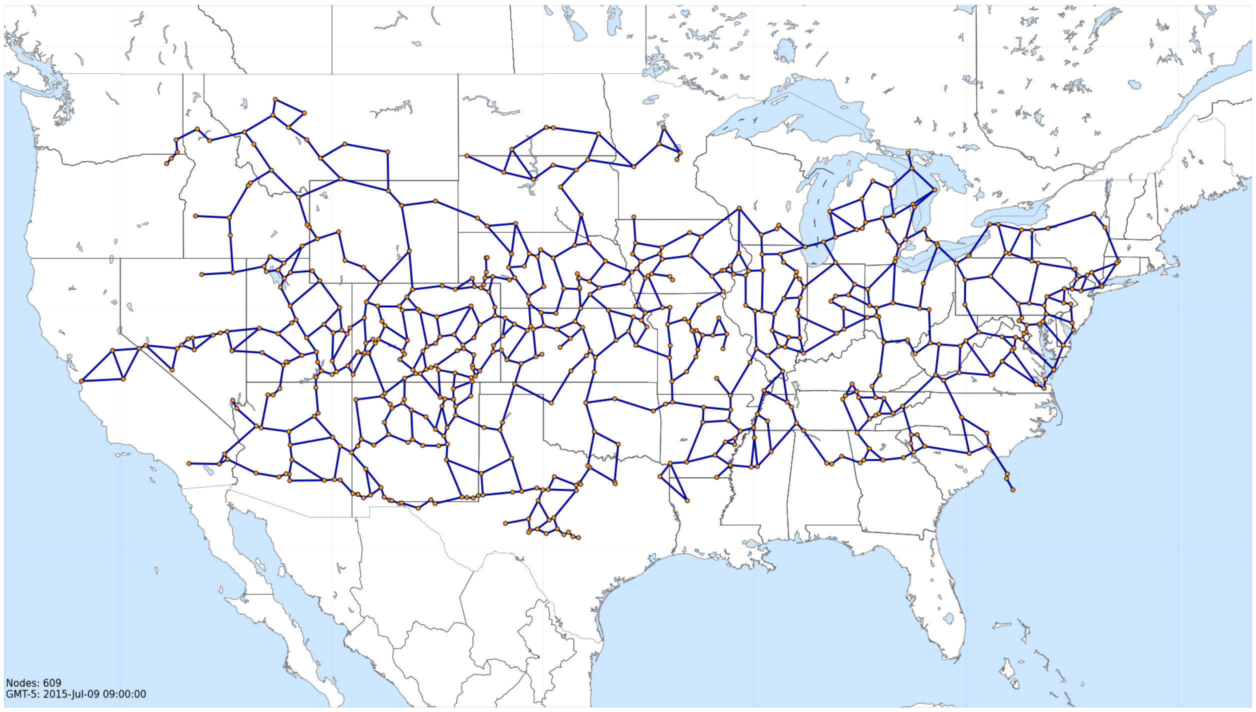


**Figure 5.2: The network of aircraft used for the global routing comparison. A subset of 609 aircraft spread across the United States at 9:00 a.m. July 9, 2015. Density: 9.47 nodes per unit disk.**

The positions of 609 aircraft spread across the United States at 9:00 a.m. on Thursday, July 9, 2015, and a DC-GG topology among them are shown in Figure 5.2. The density of this network is 9.47 nodes per unit disk. Table 5.2 lists the simulation parameters specific to this experiment. One thousand flows between random pairs of nodes are simulated, with each flow sending application data at a constant rate of 100 kbps. The experiments is run for 60 simulated seconds, during which node positions remain fixed.

In Table 5.3 the overall results from the experiment are compared. Notice that both TAG and global

**TABLE 5.2: A table listing the Global Routing Comparison parameters.**

| | | |
|---|---|---|
| flow params | flow style | flows between random node pairs |
| | number of flows | 1,000 |
| node mobility params | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 9, 2015, 09:00 a.m. CT |
| | duration | 1 minute |
| | number of nodes | 609 nodes |
| | density | 9.47 nodes per unit disk |
| application params | application type | TCP Constant Bit Rate (CBR) |
| | flow rate | 100 kbps |
| | TCP congestion avoidance algorithm | NewReno |
| | TCP send and receive buffer size | 400,000 |
| routing params | routing algorithm/protocol | TAG and GlobalRouting |

routing have a packet delivery ratio of 1.0, no packets are dropped by either protocol. Both protocols transmit over 1.5 million packets and over 800 megabytes of data. However, because TAG has a shim header, it ultimately transmits more bytes in total and has a slightly higher throughput. On average, TAG packets must travel almost 9 hops more (37% more) than the globally routed packets, with TAG packets traveling almost 32 hops on average compared to just over 23 hops for global routing. There is a direct correlation between these higher numbers of hops for TAG, and the one-way packet delay experienced by TAG's packets. The globally routed packets, on average, reached their destinations 1.28 ms (15.6%) faster that those routed with TAG (8.2 ms compared with 6.9 ms).

**TABLE 5.3: A table listing the results of the Global Routing Comparison.**

| | TAG | Global Routing |
|---|---|---|
| tx packets | 1,589,253 | 1,589,253 |
| rx packets | 1,589,253 | 1,589,253 |
| rx bytes | 934,877,270 | 818,861,801 |
| aggregate throughput | 124.65 Mbps | 109.182 Mbps |
| PDR (packet delivery ratio) | 1.0 | 1.0 |
| mean one-way delay | 8.20298 ms | 6.92499 ms |
| mean hops per packet | 31.9284 | 23.2758 |

Figure 5.3 shows the cumulative distribution function (CDF) for the mean one-way delay, averaged over all the packets routed from the source to the destination (excluding the ACK flows) in each of the 1,000 flows. Note that for both TAG and global routing, for nearly 30% of the flows the delay is less than about 5 milliseconds. Global routing has a median one-way delay of 6.45 ms and a maximum one-way delay of 17.91 ms. Comparing this to TAG's median one-way delay of 7.06 ms (9.5% more) and maximum one-way

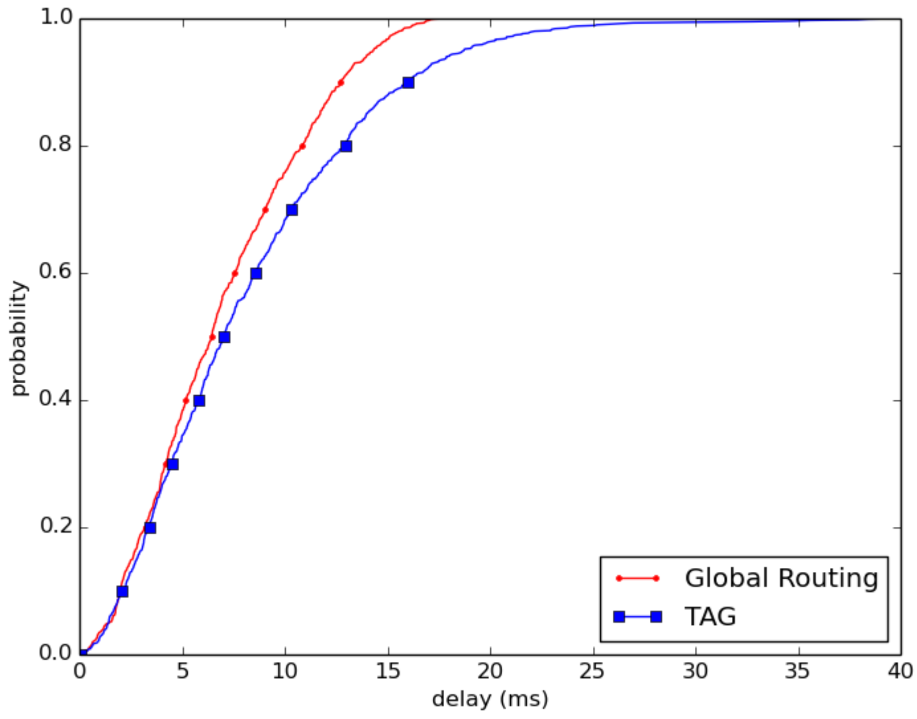


**Figure 5.3: The CDF of the mean delay over all the packets in each flow.**

delay of 39.98 ms, it is easy to see that a small percentage of packets are delayed twice as long when TAG is used. Note that the propagation time alone for a packet traveling at the speed of light directly from Seattle to Miami would be 15 ms. Since the speed of light is slower through fiber-optic cables (approximated at $2x10^8$), however, the propagation time for communicating through a single very long fiber-optic cable stretched between the same cities would be closer to 22 ms. In comparison, a maximum one-way delay (including transmission time and non-optimal paths) of under 40 ms is quite good. As will be seen next, at these low rates this delay can largely, if not completely, be attributed to the higher numbers of hops traveled by these packets. These flows with long mean delays are likely ones where packets must route around a big face to advance towards the destination. With global topology information, the global routing protocol is able to avoid these situations. TAG, however, must explore using only local information attempting to make progress towards the destination. This causes an increase in the delay but allows TAG to be very agile, responding immediately to changes in the dynamic network. Because TAG requires no overhead packets and uses only local information, it is inherently scalable. These benefits, however, must be weighed against the increase in delay and a potential increase in network congestion.

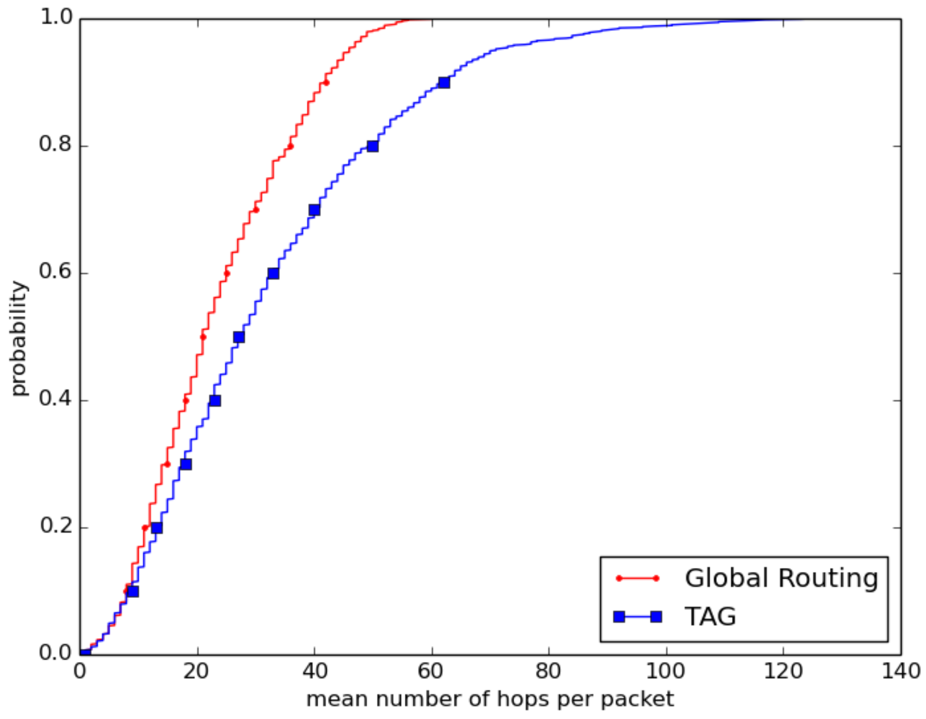


**Figure 5.4: The CDF of the mean number of hops per packet in each flow.**

In Figure 5.4 the cumulative distribution function (CDF) for the mean number of hops traveled by the packets in each flow (excluding ACK flows) is presented. The plot is similar in form to that of the delay CDF. As expected, global routing performs better, but TAG doesn't lag too far behind with a median of 27 hops compared with global routing's 21 hops. A small percentage of TAG's flows must travel over many more hops on average with a maximum of 125 hops, more than double global routing's maximum of 60 hops. As mentioned above, this is one of the costs of using geographic routing.

These simulations show that, as expected, in a static network TAG performs worse than global routing in terms of throughput and one-way delay. But, for a significant percentage of the flows, TAG is able to remain fairly competitive with global routing.

### 5.2.3 Large-Scale and High-Density

In this experiment, the scalability of the topology management and geographic routing are showcased. One thousand simultaneous flows between random pairs of nodes are simulated in a network of nearly 8,000 nodes. It is estimated that there are often around 7,000 aircraft in the skies over the United States; including general aviation aircraft, military aircraft, passenger aircraft, and cargo aircraft. This simulation seeks to form a network and route data among even more aircraft than that. The simulation is not intended to validate the capacity of the network, but rather ensure that packets are able to be routed effectively despite the large scale. For this reason, low-rate flows are simulated. Because the ASDI data set (aircraft mobility data) does not contain enough data to reach this high node density by itself, a composite data set was generated for these simulations. The aircraft trajectory data from July 16, 2015, starting at noon central time is augmented with the aircraft trajectory data from July 17, 2015, starting at 1:00 p.m. central time. The time stamp for each position update from the July 17th data has 25 hours subtracted from it such that the times for these position updates overlap with those of the July 16th data. The call signs for the flights are also augmented to avoid issues where multiple flights with the same call sign exist at the same time. The 25-hour offset is chosen to avoid the potential situation where regular daily flights could essentially overlap with one another, making the simulation less realistic.

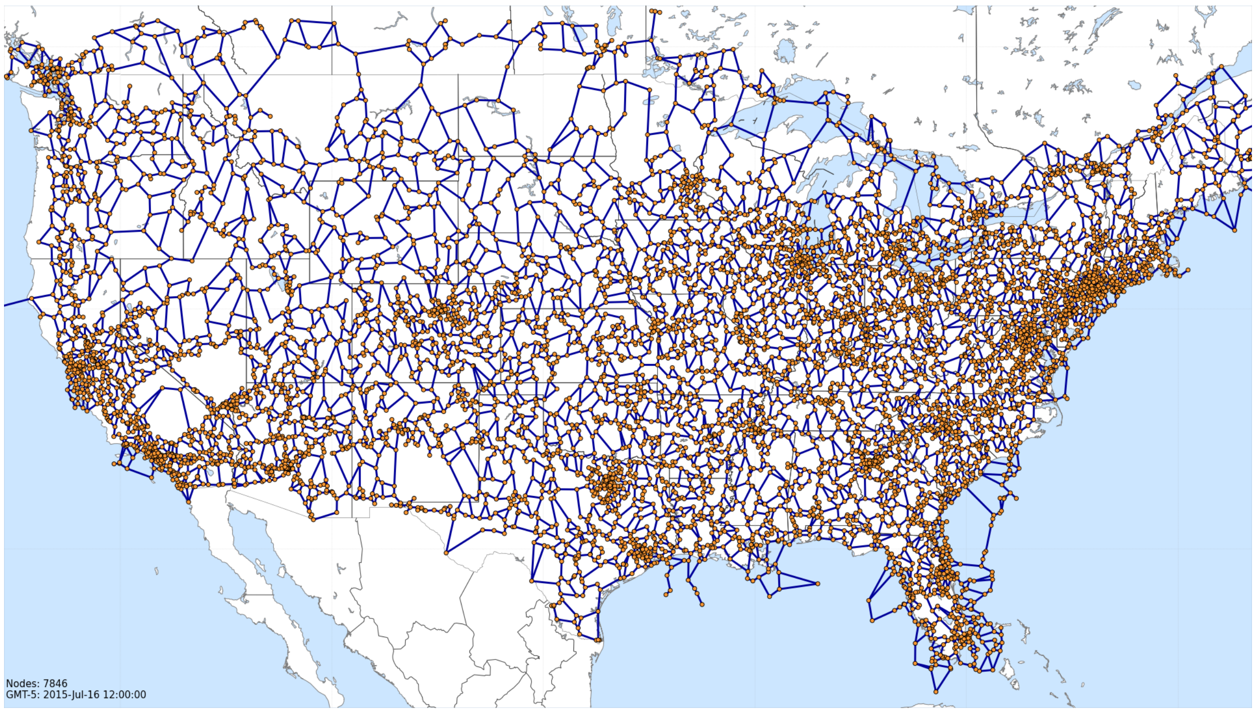


**Figure 5.5: A snapshot of the network of aircraft used for the Large-Scale High-Density Experiment. A composite of the air traffic on July 16, 2015, at 12:00 noon and July 17, 2015, at 1:00 p.m. central time.**

Figure 5.5 shows a snapshot of the nodes and topology at the start of the simulation (July 16, noon central time). Notice that the density is much higher than the other simulations in this chapter, and there are many nodes clustered at busy airports. The number of nodes initially is 7846, and over the first minute the number of nodes varies between 7837 and 7851 nodes, which corresponds to densities of 121.88 and 122.10 nodes per unit disk.

Table 5.4 shows the simulation parameters for this experiment. Three different simulations are run, one using TCP as the transport protocol, another using UDP, and a third using UDP and a smaller community range of 100 km (about one-third the default range of 288 km). One thousand flows each sending TCP messages at a constant rate of 10.984 kbps are simulated in the first simulation. In the other simulations, 1,000 UDP flows are simulated, each sending data at a constant rate of 11.192 kbps. These rates equate to the sending of one packet per flow per second. This essentially allows the simulation to test the path between each of the 1,000 node pairs once a second. The aircraft in this simulation are dynamic, with the nodes following real aircraft trajectories from the dates indicated.

**TABLE 5.4: A table listing the Large-Scale High-Density Experiment parameters.**

| | | |
|---|---|---|
| flow params | flow style | flows between random node pairs |
| | number of flows | 1,000 |
| node mobility params | mobility model | composite actual aircraft trajectories (ASDI) |
| | start date/time | July 16, 2015, 12:00 p.m. CT combined with July 17, 2015, 1:00 p.m. CT |
| | duration | 1 minute |
| | number of nodes | 7837 to 7851 |
| | density | 121.88 to 122.10 |
| application params | application type | TCP constant bit rate (CBR) and UDP constant bit rate (CBR) |
| | flow rate | 10.984 kbps (TCP) and 11.192 kbps (UDP) |
| topology management params | position exchange (community) range | 288 km and 100 km |

An overview of the simulation results is contained in Table 5.5. The simulations only route a relatively small number of packets and send a total of about 78 to 91 megabytes (MB) of data during the 1-minute simulation yielding an aggregate throughput of about 10 to 12 Mbps. Notice that even with the high density and the large number of hops on average, only 18 TCP and 8 UDP packets are dropped in the first two simulations, maintaining packet delivery ratios (PDRs) in the three 9's range (0.999847 and 0.999856). There are a high number of hops on average, but relatively low one-way delays. This is because the propagation time dominates the delay, and although the packets must travel over many hops, each hop is over a short distance. In both simulations using the default community range, all the dropped packets were discarded because the number of hops exceeded the time-to-live (TTL) value of 255. The impact of using a smaller community range can be seen in the higher mean one-way delay and the larger number of mean hops per packet in the third simulation. In addition, in the third simulation where a smaller community range is used, there were 3,706 packets dropped resulting in a lower PDR of 0.933415. The benefit of the smaller community range, however, is nearly an order of magnitude decrease in the computational load, indicated by the simulation completing almost 7 times faster than the same UDP simulation using the default community range. Adjustment of the position exchange range allows for a trade-off between delivery accuracy and delay, and computational load.

**TABLE 5.5: A table listing the results of the Large-Scale and High-Density experiment.**

| | TCP and TCP ACKs | UDP | UDP 100 km community range |
|---|---|---|---|
| tx packets | 117,341 | 55,658 | 55,658 |
| dropped packets | 18 | 8 | 3,706 |
| rx bytes | 91,104,166 | 83,475,000 | 77,928,000 |
| aggregate throughput | 12.15 Mbps | 11.13 Mbps | 10.39 Mbps |
| PDR (packet delivery ratio) | 0.999847 | 0.999856 | 0.933415 |
| mean one-way delay | 9.21735 ms | 9.26088 ms | 9.68679 ms |
| mean hops per packet | 83.2585 | 83.2315 | 110.155 |

In Figure 5.6 the cumulative distribution function (CDF) for the mean one-way delay per packet for each of the 1,000 flows is shown. For TCP, the reverse stream of ACKs is separated from the TCP data stream and analyzed and plotted separately, since the forward and reverse streams often travel different paths. Excluding the simulation using a smaller position exchange range, all the CDFs are very similar, and each varies only slightly from the other. Each value used to generate the CDF is the mean one-way delay among all of the packets transmitted in the given flow. The maximum of these values is 25.84 ms, 26.79 ms, and 25.64 ms for TCP, UDP, and TCP ACKs respectively. Although it is often the case that ACK packets flowing from the receiver to the data source may travel a different path than the data packets, on average the path lengths and delays are essentially the same. Notice also, as would be expected, the TCP and UDP packets experience essentially the same one-way delay. The median one-way delay is 8.27 ms for the TCP flows, 8.18 for UDP flows, and 8.32 ms for the TCP ACK flows. In many respects, the topology management framework and geographic routing actually function better at these higher densities. This is likely because with a higher density of nodes there are fewer large faces, on average, and therefore fewer local minima. This allows greedy routing to succeed more often, avoiding the need to resort to potentially costly face routing. When

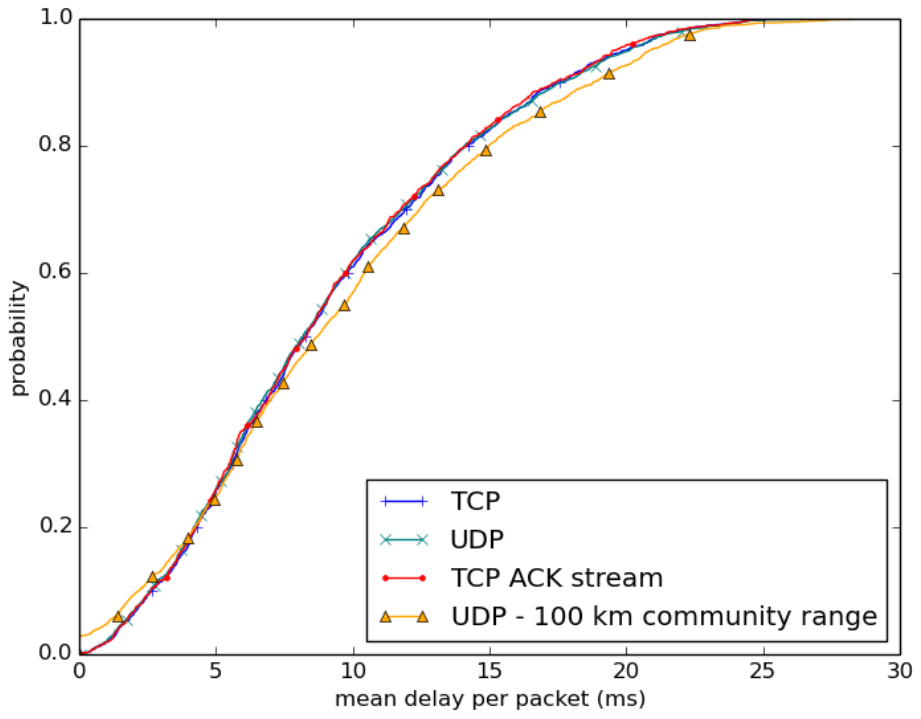


**Figure 5.6: The CDF of the mean one-way delay per packet for each flow in the Large-Scale High-Density simulations.**

a larger community range is used there is a noticeable impact on the delay. However, the median one-way delay is still just 8.73 ms and the maximum one-way delay is 29.92 ms for the delivered packets.

Figure 5.7 is a plot of the cumulative distribution function (CDF) for the mean number of hops per packet for the 1,000 flows in each experiment. As with delay, there is very little difference in the distributions for the mean number of hops experienced by the TCP, UDP, and TCP ACK flows using the default position exchange range. The median number of hops is 76.64 for TCP, 76.11 for UDP, and 76.39 for TCP ACKs. The maximum number of hops experienced by the average packet in a flow is 219.38 for TCP, 221.24 for UDP, and 223.10 for TCP ACKs. In both of these experiments, the handful of packets that were dropped were all dropped due to IP time-to-live (TTL) expirations. These packets are dropped once they have traveled over 255 hops (with the TTL set to its maximum allowable value). In these rare circumstances, the path taken by the packet is so long that it is dropped by the Internet Protocol. When the smaller community range is used there is an increase in the average number of hops in a packet's path. The median number of hops increases to 90, with a maximum of 247 hops. With the decreased position exchange range each node has a

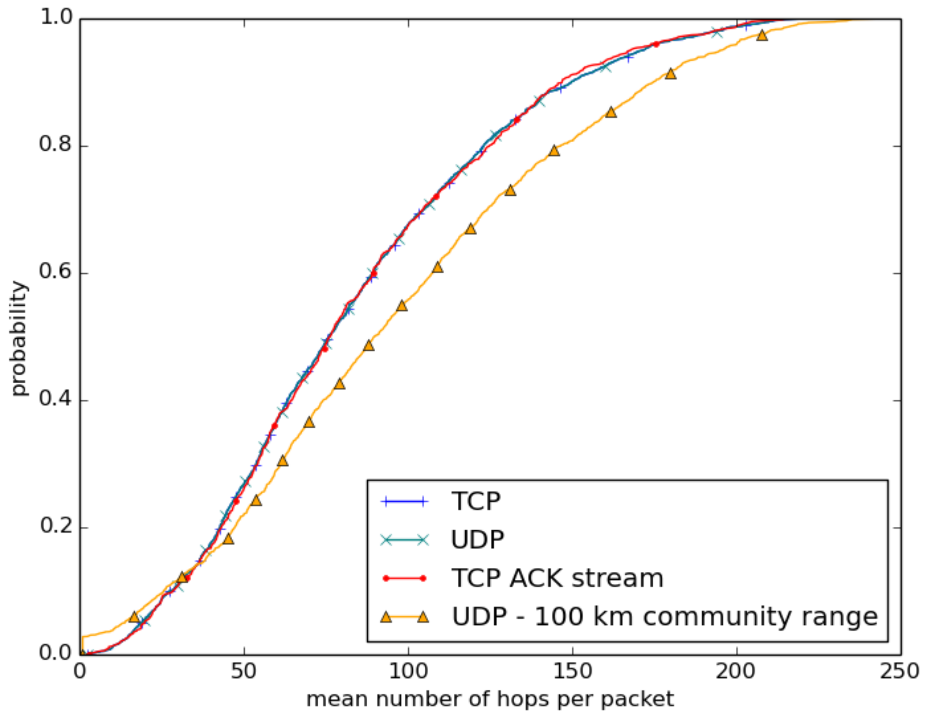


**Figure 5.7: The CDF of the mean number of hops per packet for each flow in the Large-Scale High-Density simulations.**

smaller local view, and therefore makes less optimal forwarding decisions, thus increasing the average hop count.

This last figure, Figure 5.8, presents a scatter plot showing the correlation between the mean number of hops and the mean one-way delay for the TCP experiment. Each sample is presented as a light disk. Darker areas indicate areas where many disks overlap. The fact that most of the samples lie in a diagonal line indicates that, as expected, the delay is tightly coupled to the numbers of hops. For flows with shorter delays and shorter numbers of hops, there is less variance, and the samples are packed tightly. For larger numbers of hops and longer delays, there is more variance and the samples are spread out. This spreading is a result of queuing on the links and/or the diversity of lengths of hops in the paths.

This experiment demonstrates that the topology management framework and TAG are capable of operating in high-density, large-scale simulated networks of well over 7,000 dynamic nodes, while successfully routing nearly all of the packets, and without incurring large amounts of delay in an uncongested condition. Additionally, this experiment gives a glimpse at how decreasing the position exchange range can impact

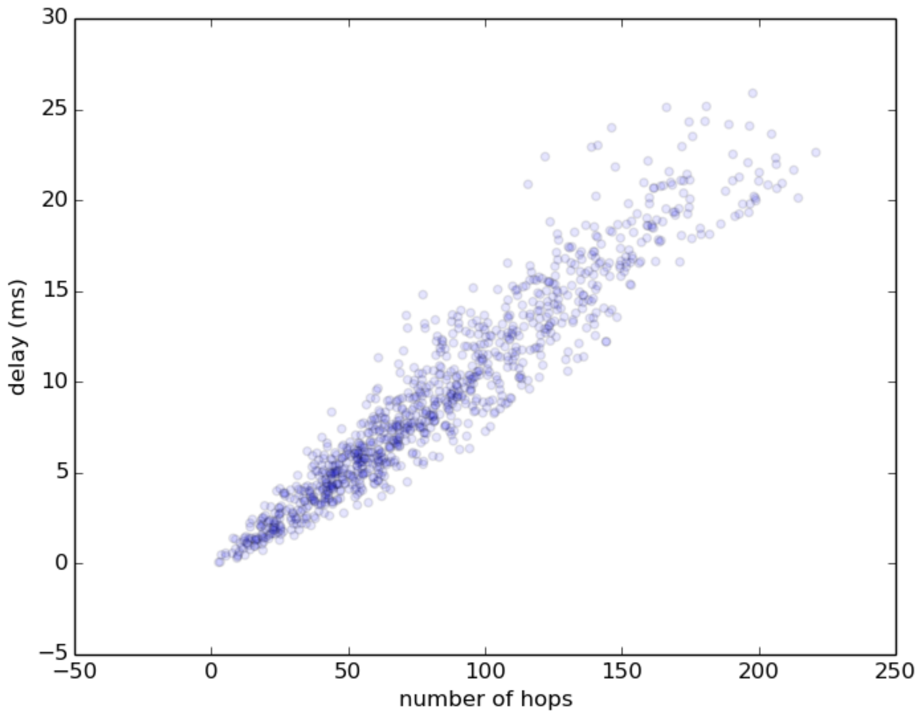


**Figure 5.8: A scatter plot showing the correlation between the mean number of hops and the mean one-way delay.**

flows in the network.

### 5.2.4 Single High-Rate Flows

The simulations presented thus far have only simulated low-rate flows. What about high-rate flows? In this experiment single high-rate flows are simulated in an airborne network of hundreds of simulated aircraft flying real aircraft trajectories. Table 5.6 shows the simulation parameters, in addition to the default parameters, used for this experiment.

Figure 5.9 shows a snapshot of the network of simulated nodes as they exist at the start of these experiments. One pair of nodes is selected for the high-rate experiments. The pair of nodes was selected such that the mean path length, in hops, between them approximates the mean overall path length experienced by one thousand flows simulated in the same network topology.

For the first experiment, a 9.5 Gbps UDP constant bit rate (CBR) flow is simulated for 1 minute. Figure 5.10 shows the resulting throughput and numbers of hops for the experiment. Notice that the throughput stays steady at just under 10 Gbps (including header/overhead bytes). The number of hops also remains

**TABLE 5.6: A table listing the Single High-Rate TCP Flow Experiment parameters.**

| flow params | flow style | single flow |
|---|---|---|
| node mobility | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 9, 2015, 13:00 CT |
| | duration | 22 seconds, 60 seconds, and 7 minutes |
| | number of nodes | 264 to 304 (growing) |
| | density | 4.11 to 4.73 nodes per unit disk |
| application params | application type | TCP Bulk Send and UDP constant bit rate |
| | TCP congestion avoidance algorithm | NewReno |
| | TCP send and receive buffer size | 60 MB |
| network device params | Maximum Transmission Unit (MTU) | 1500 and 9000 |

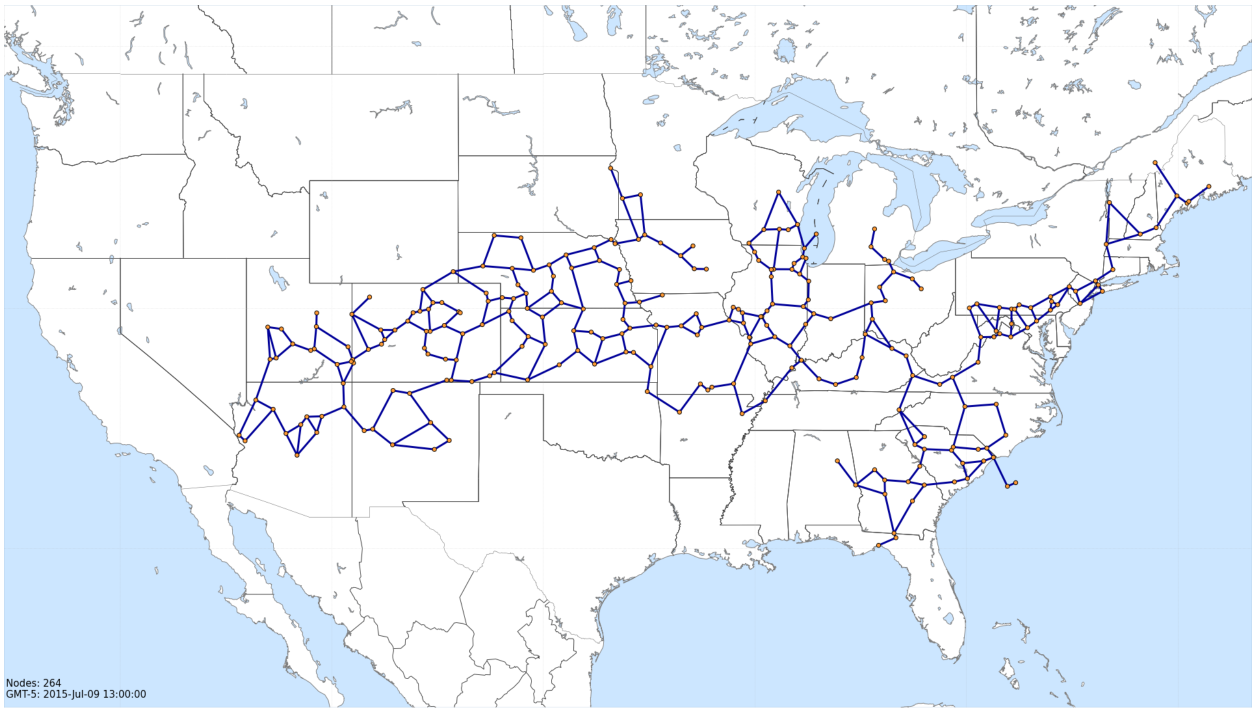


**Figure 5.9: A snapshot of the initial network of aircraft used for the single high-rate flow experiments and the all pairs experiments. The 264 nodes shown here span a large portion of the United States at 1:00 p.m. on Thursday, July 9, 2015.**

fairly steady averaging just under 40 hops per packet. The path taken by the packets, however, varies substantially over the 1-minute simulation.

Unfortunately, flows between pairs of nodes are not always this perfect. Figure 5.11 shows the results of a 1-minute simulation of a 9 Gbps UDP constant bit rate (CBR) flow. The figure also shows the throughput and number of hops as the simulation of this flow progresses. Note that except for a 7 second time period (just before the 40-second mark), the network was able to maintain a consistent UDP throughput. At the 39 second mark, there is a change in the path taken by the packets and about a 60% drop in throughput, to about 3.5 Gbps. This is likely the result of the flow encountering a condition where packets must travel over a link multiple times in attempting to overcome a local minimum. Notice that the issue causing the drop in throughput is not the number of hops, since there are more hops along the path for nearly half of the simulation without any throughput issues. The throughput is negatively impacted until the packets begin taking a better path due to a change in the network topology or a change in the relative positions of the nodes.

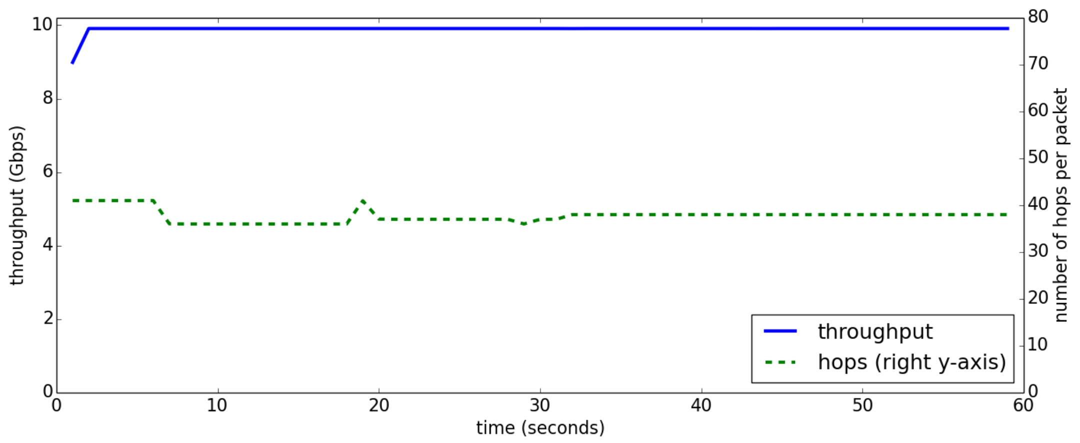


**Figure 5.10: The throughput and number of hops for a one minute single high-rate UDP flow experiment.**

Next, a similar experiment is performed, but this time using TCP as the transport protocol instead of UDP and using New Reno as the congestion control algorithm. A single Bulk Send application sends data as fast as possible from the source node to the destination node. A maximum transmission unit (MTU) of 9000 bytes, the maximum jumbo frames size, is used for this experiment instead of the default 1500 byte value. Changing the MTU enables the congestion window to ramp up more quickly for this experiment.

Figure 5.12 shows the resulting throughput and numbers of hops over the 21-second simulation. Note that the time resolution of the plot is finer than in the previous experiment. The various stages of the TCP congestion control algorithm can be seen including slow start, and the ramping up of the congestion window

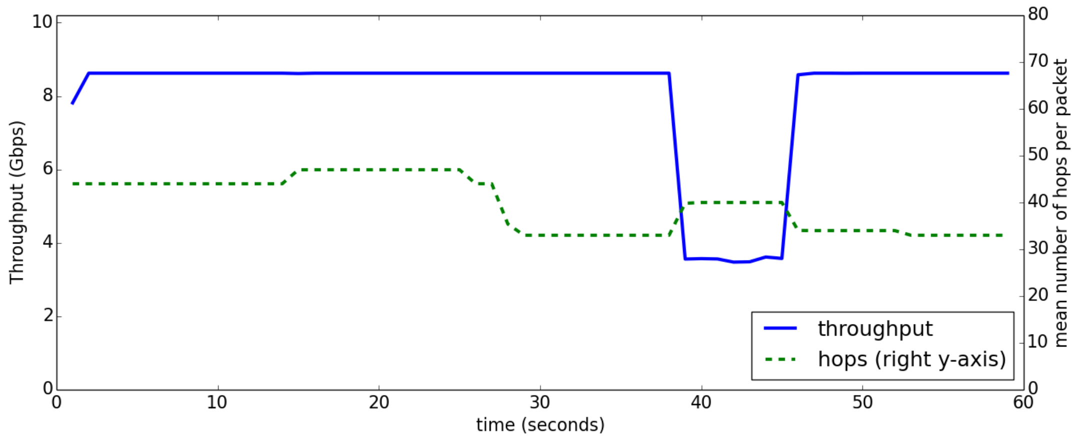


**Figure 5.11: The throughput and number of hops for a one minute single high-rate UDP flow experiment with a drop in throughput.**

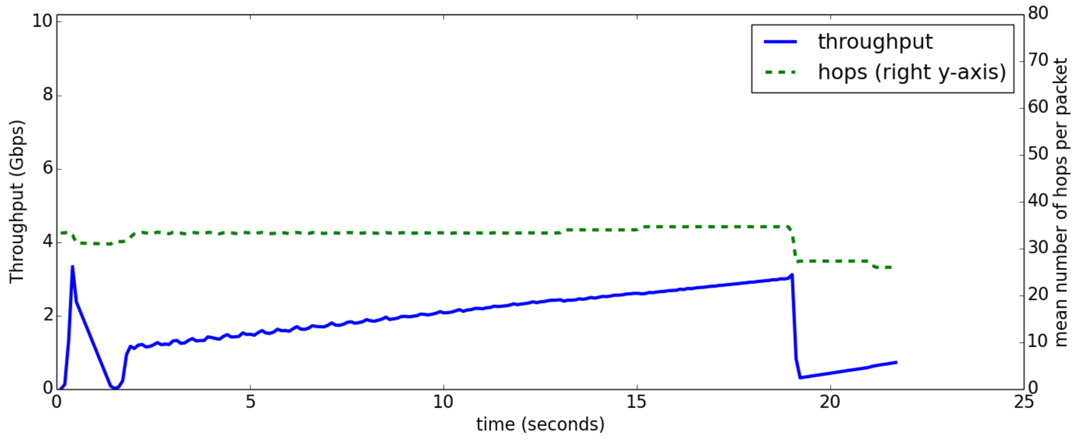


**Figure 5.12: A simulation showing how TAGs ability to begin sending packets over shorter paths can cause TCP to react negatively, decreasing the throughput potential for a high-rate flow.**

size during congestion avoidance. Notice that the instantaneous throughput is much less than that of UDP, with TCP never reaching above about 3.4 Gbps, though the link capacity is 10 Gbps. About 19 seconds into the simulation there is a sharp decrease in the number of hops traversed by each packet. The topology of the network or the positions of the nodes changed such that TAG now found a path that was 6 hops shorter. The unintended consequence here, however, is that the packets traveling the shorter path arrive at the destination before the packets that just missed taking advantage of the new shorter path. When TCP starts receiving the packets taking the shorter path, it sends acknowledgments of the last received packet before the gap where the remaining in-flight packets should be. On the other end of the connection, TCP unwittingly assumes that the packets, that are still traveling on the original (slower) path, must have been dropped, and upon receiving a triple duplicate ACK enters the fast retransmit phase. TCP reduces the congestion window causing a substantial drop in throughput, requiring the congestion window and throughput to be slowly scaled back up. TCP is unaware of TAG's ability to find and use better paths instantaneously. Even though no actual packet loss occurred after the first few seconds of this simulation, TCP limited the throughput, assuming there was congestion along the paths. TAG's agility leads to low throughput, in general, for high-rate flows even though the network has the capacity to support more throughput.

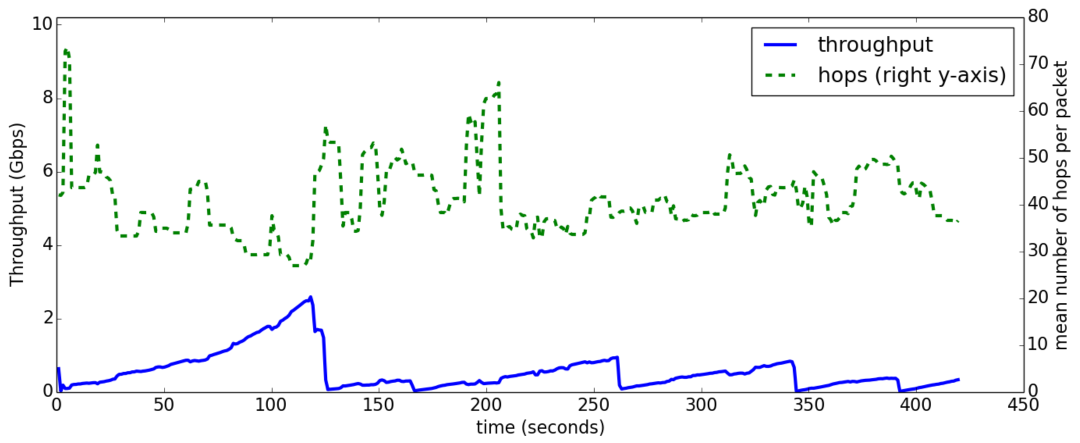


**Figure 5.13: The throughput and numbers of hops encountered after changing the duplicate ACK threshold to 10.**

Tuning the TCP parameters can yield somewhat improved performance. Figure 5.13 shows the result of simulating a single TCP bulk send flow through the same airborne network. In this simulation, however, the duplicate ACK threshold is set to 10, meaning that 10 duplicate ACKs must be received before the fast retransmit state is entered. The MTU is also reset to the default of 1500 bytes. The familiar saw-tooth pattern is evident in the graph as TCP attempts to estimate the capacity of the links. The throughput, however, is far below the desired throughput of 8 to 9 Gbps. A different congestion control algorithm or transport protocol may be needed to efficiently support high-rate TCP-like flows in the proposed airborne network architecture.

This simulation demonstrates the following:

- The network topology management and TAG are capable of handling high-rate UDP flows, but there can be cases where the throughput is drastically cut due to changes in the paths taken by packets.

- TCP can punish TAG for finding shorter paths in the network, causing packets to arrive out of order.

- It may by necessary to use a different congestion control algorithm or a different transport protocol if high-rate TCP-like flows are to be supported by the airborne network.

### 5.2.5 Long Duration

This experiment tests the network with a large number of low-rate UDP flows over a long simulation time in a dynamic network. During the simulation, thousands of flows are simulated sending data between random pairs of nodes whose in-flight times overlap during the simulation. Each of the flows is a low-rate UDP flow sending at a rate of 10 kbps. Simulating at these low rates certainly does not shed any light on the effect of congestion in the network, but that is not the purpose of this experiment. The intent, rather, is to demonstrate that data is able to be consistently routed through this medium-density network over long periods of time. Table 5.7 gives the pertinent simulation parameters for this experiment.

**TABLE 5.7: A table listing the Long Duration Experiment parameters.**

| | | |
|---|---|---|
| flow params | flow style | flows between random node pairs |
| | number of flows | ˜200-300 at a time, 4,000 (in a full 8-hour experiment) |
| node mobility params | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 16, 2015 9:00 a.m. |
| | duration | ˜5.25 hours |
| | number of nodes | 2032 to 2262 |
| | density | 31.60 to 35.17 |
| application params | application type | UDP constant bit rate (CBR) |
| | flow rate | 10 kbps |

Figure 5.14 shows a snapshot of the network 1 hour into the 5-hour simulation, and Figure 5.15 displays a similar snapshot, but 3 hours into the simulation, when the network is more dense. The density of the nodes increases from about 31 to over 35 nodes per unit disk (2032 to 2262 nodes total) over the course of just over five hours. In this experiment, there has been no attempt made to ensure that the aircraft are always able to form a global topology that is not partitioned. This means that this is a more realistic experiment than the Single High-Rate experiments, where destinations were guaranteed to be reachable.

Figure 5.16 shows a plot of the throughput over the course the long duration experiment. Also plotted (lighter) against the right y-axis is the number of flows active during the experiment. Notice that the number of flows active at any one point in time in the simulation varies over the course of the experiment because the flows are initialized such that packets are only transmitted during the time that both the source and destination aircraft for the flow are in-flight. The throughput changes in response to changes in the number of flows, and on no occasion does the throughput drop significantly without a similar drop in the number

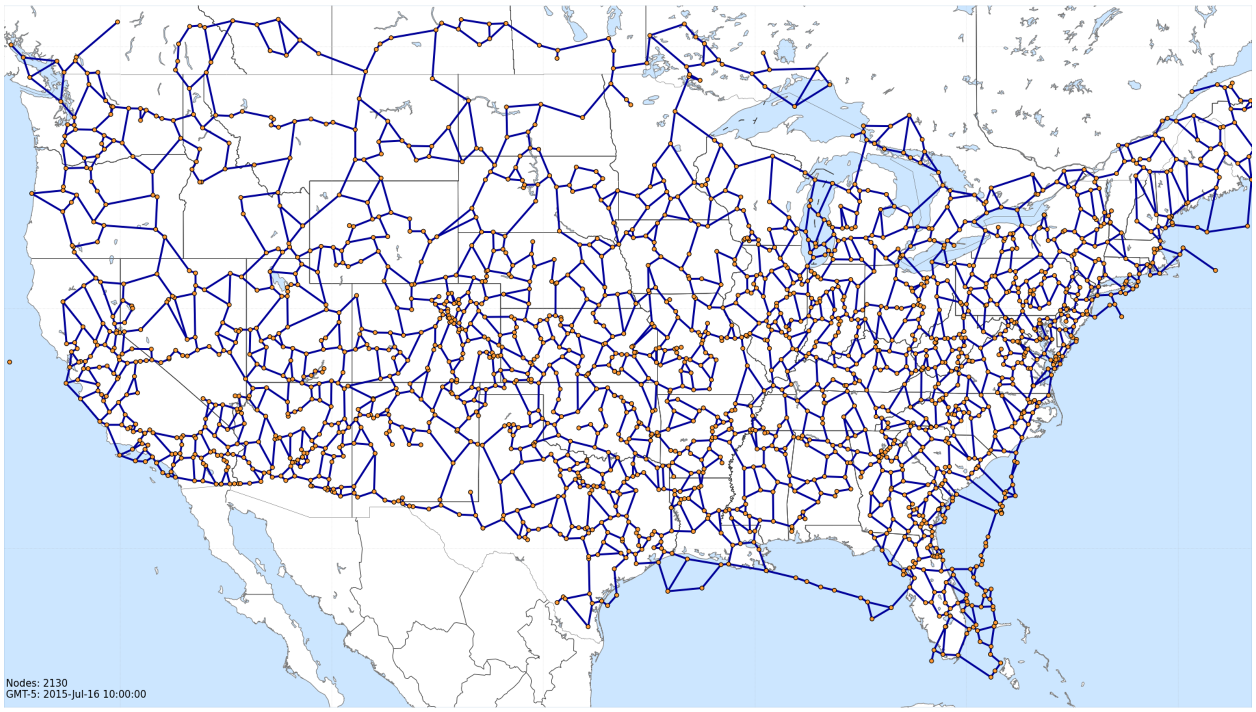


**Figure 5.14: A snapshot of the network of aircraft at the start of the long duration experiment. July 16, 2015 at 10:00:00 central time.**

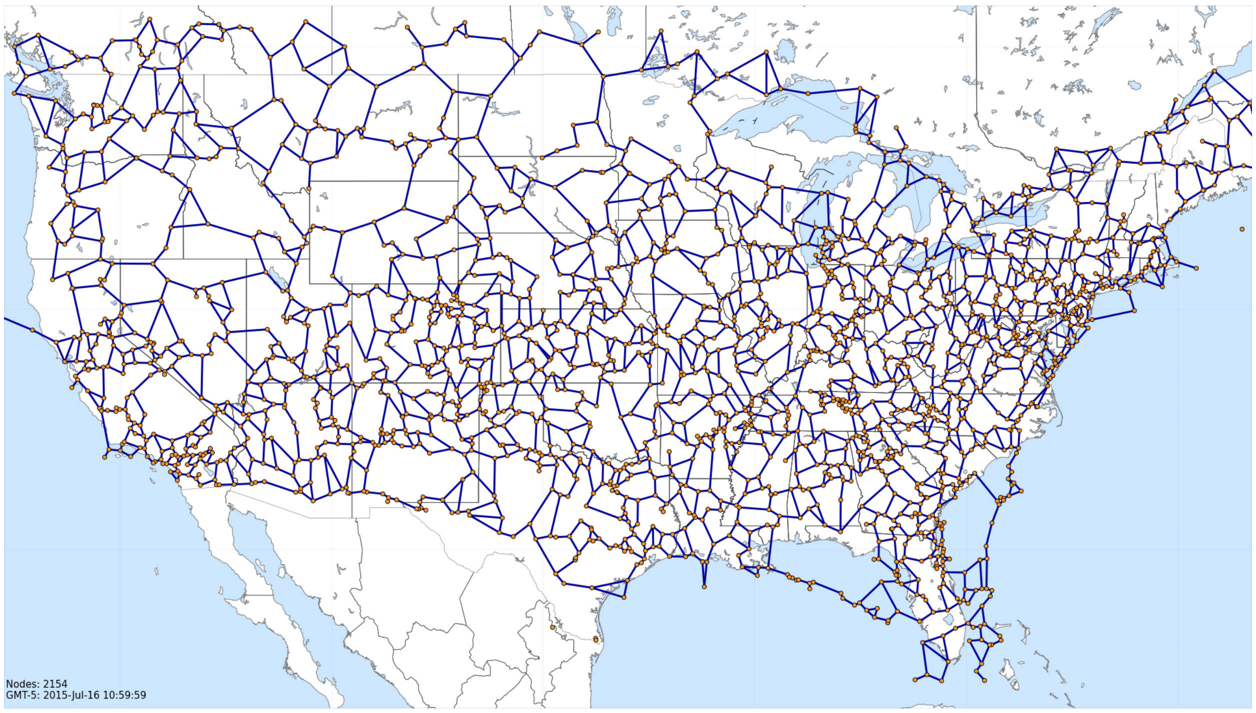


**Figure 5.15: A snapshot of the network of aircraft after 2 hours of the long duration experiment, July 16, 2015 at 12:00:00 central time.**

of flows. The minimum throughput experienced during the experiment is 1.68 Mbps and the maximum throughput is 4.02 Mbps, with a mean of 2.38 Mbps.

This experiment gives one confidence that the topology of a medium-density network is able to be managed successfully, for hours on end, and that TAG is able to route data consistently over long periods of time.

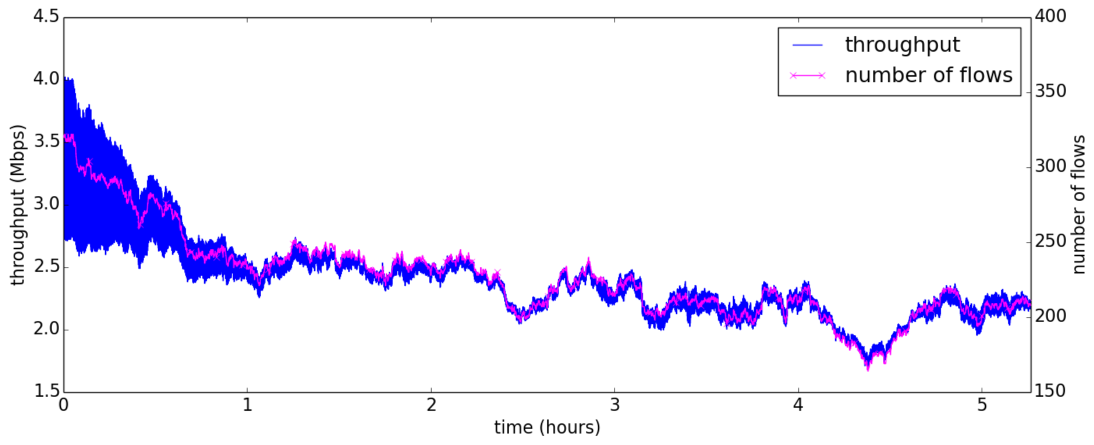


**Figure 5.16: A plot of the throughput over the course of the long duration experiment.**

### 5.2.6 Effect of Loss

The experiments described to this point have assumed that the data links have no loss, but this is an unrealistic assumption. This experiment intends to shed some light on how the network will operate when different bit error rates are applied to the connections. The bit error rate is simulated by having the network device at each hop essentially roll a die to determine whether or not to actually receive the packet. Packets that aren't correctly received are dropped. The determination of whether or not to drop the packet is made by first translating a given bit error rate (BER) into a packet error rate (PER), taking into account the size of the packet and assuming uniformly distributed bit errors. A random number is then generated and compared with the packet error rate to determine whether or not to drop the packet.

This simulation is run on a medium-density set of nodes as seen on July 16, 2015, at 7:00 a.m. CT. Figure 5.17 shows a snapshot of what the network looked like at this time. There are 1479 to 1483 nodes active at any one point during the simulation, producing a node density of about 23 nodes per unit disk. The aircraft trajectories for this simulation have only been filtered to remove aircraft position updates outside of the continental United States and to remove aircraft position update below 10,00 feet altitude. No attempt has

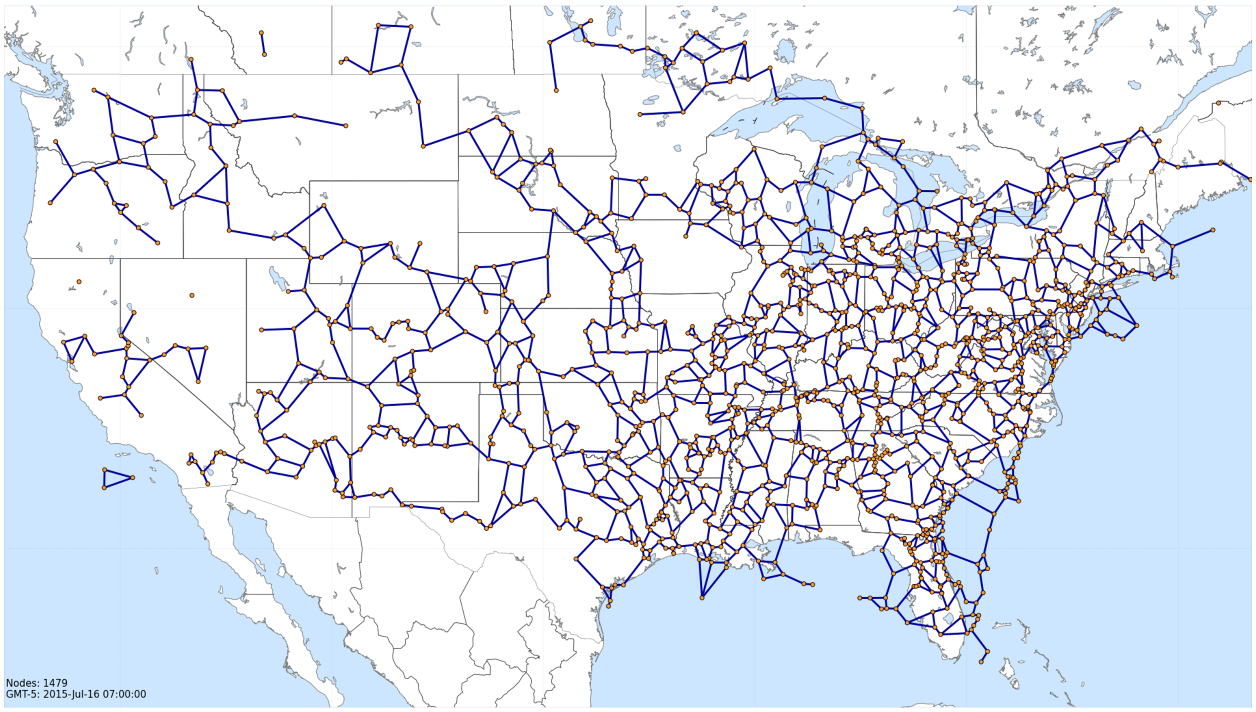


**Figure 5.17: A snapshot of the network of aircraft used to analyze the effect of loss.**

been made to ensure that paths exist between the source and destination node pairs for the flows. Therefore, some of the delivery errors in these simulations may come as a result of there being no path between the nodes.

The parameters of the experiment are displayed in Table 5.8. One thousand flows between random pairs of nodes are simulated for 1 minute. Each flow sends TCP messages such that 1 megabit of application data is sent every second. Five bit error rates are compared: $1x10^{-6}$, $1x10^{-7}$, $1x10^{-8}$, $1x10^{-9}$, and $1x10^{-10}$.

Bagley et al. report on a United States Department of Defense (DoD) demonstration where hybrid RF/FSO links are used to exchange data at just over 10 Gbps between an aircraft and a surface terminal on a mountain top [BHJ+12]. They report power at the detector, also called power in the fiber (PIF) values, for the hour-long experiment. The experiment indicates that except for a few short periods where the connection appears to have been lost, the PIF values approach -20 decibels-milliWatts (dBmW) always staying above -30 dBmW when the link range is about 100 km. As the link range approaches 200 km the PIF values decrease to just above -40 dBmW. According to the authors, their system must maintain a PIF above -40 dBmW to maintain a reliable link. The same work presents a graph showing the tested bit error rates (BERs) given various system configurations assuming a 10 Gbps link rate. With optical automatic gain control (OAGC) and forward error correction (FEC) enabled, PIF values down to -39 dBmW yield a BER

**TABLE 5.8: A table listing the Effect of Loss Experiment parameters.**

| | | |
|---|---|---|
| flow params | flow style | flows between random node pairs |
| | number of flows | 1,000 |
| node mobility params | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 16, 2015, 7:00 a.m. CT |
| | duration | 1 minute |
| | number of nodes | 1479 to 1483 |
| | density | 23.00 to 23.06 |
| application params | application type | TCP constant bit rate (CBR) |
| | flow rate | 1 Mbps |
| network device params | bit error rate (BER) | 1x$10^{-6}$, 1x$10^{-7}$, 1x$10^{-8}$, 1x$10^{-9}$, and 1x$10^{-10}$ |

of 1x10⁻12. With no OAGC and no FEC at -20 dBmW, a link can still achieve a BER of just better than 1x10⁻8. These results were presented nearly 5 years before this writing, and today's systems are expected to perform even more reliably.

Figure 5.18 shows the CDF for mean throughput for each of the 1,000 flows (excluding ACK flows), and for various bit error rates. Notice that a small percentage of flows are unable to send any data (the curves are

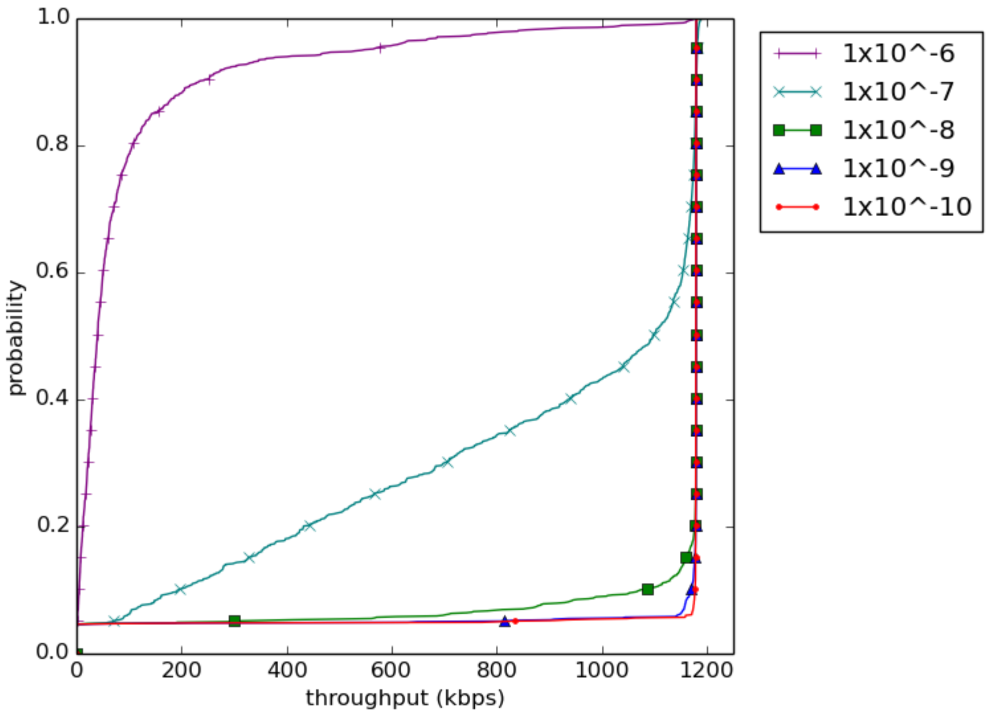


**Figure 5.18: The CDF of the throughput at various bit error rates (BER).**

offset from the x-axis). This is because the network at this early hour (5 a.m. PT) is partitioned and several nodes, especially those in the California area, are disconnected from the rest of the network. This would not be an issue if ground stations were used to link the entire network together even if the air network were partitioned.

The graph indicates that flows in this medium density network can pass 1 Mbps data with little or no impact when the bit error rates are below 1x$10^{-9}$. Bit error rates above 1x$10^{-8}$ begin to significantly impact the throughput of the network. With a bit error rate of 1x$10^{-6}$ over ˜90% of the flows are able to send at rates less than 200 kbps. This is not an issue that needs to be addressed, however, since the demonstration described above indicates that bit error rates well below 1x$10^{-8}$ are achievable over long ranges especially if error correction and optical gain control are employed.

Similarly, Figure 5.19 confirms the above findings, looking specifically at packet delivery ratio for the same flows. Notice, again, that for a small percentage of the node pairs the PDR is zero because the nodes are in separate parts of the partitioned network. As with the throughput results, these results indicate that a bit error rate of 1x$10^{-9}$ in this network won't have a severe impact on the network, but a bit error rate above 1x$10^{-8}$ would.

This experiment demonstrates a medium-sized airborne network's ability to cope with bit error rates as high as 1x$10^{-9}$. Further, the experiment shows that once the bit error rate reaches $1x10^{-6}$, the network is essentially no longer able to function.

### 5.2.7 All Node Pairs - TCP Flows

The next experiment is one where a constant bit-rate (CBR) TCP flow is generated between every pair of nodes in a dynamic network of 270 aircraft. It demonstrates the ability of the network to handle large numbers of flows and gives a glimpse into how congestion affects the network when aggregate throughput is high. Two different simulations are run; one with a bit rate of 100 kbps and another with a bit rate of 1 Mbps. The flows are one-way flows, in that they send application data only in one direction, though there is also a stream of TCP acknowledgment (ACK) packets flowing in the opposite direction for each one-way flow.

The average number of seats on a domestic aircraft in the United states was projected to be 128.2 in 2015, and is steadily increasing [FAA15]. It is assumed that before long there will be 135 seats per aircraft, on average. This experiment assumes that every aircraft has that exact number of seats and that all those

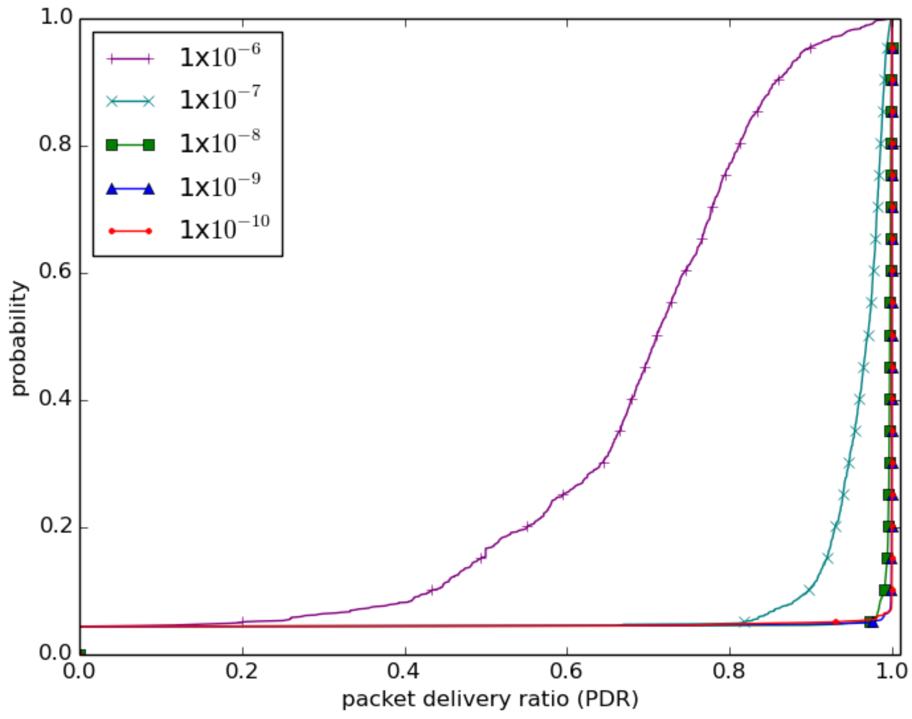


**Figure 5.19: The CDF of the packet delivery rate (PDR) at various bit error rates (BER).**

seats are occupied. It then simulates each passenger sending and/or receiving two TCP data streams to/from two passengers on different aircraft, at the given rate. Each passenger is the endpoint for two flows. Some passengers are the source for both flows, others are the destination of both flows, and some are the source for one flow and the destination of another. The purpose of this experiment is to show that the routing protocol and topology management framework are sufficient to handle large numbers of medium-rate flows. Table 5.9 lists the parameters for this experiment in addition to, or that differ from, the default parameters listed previously.

Three different simulations are run at various rates and durations. One-minute simulations at a rate of 100 kbps and 1 Mbps are run, as well as a 10-minute simulation at the 100 kbps rate. The number of nodes active in the network when the simulation starts is 264, a density of 4.11 nodes per unit disk. At the 1 minute mark there are 269 nodes active, and at the 10-minute mark 304 nodes are a part of the network, yielding density values of 4.18 and 4.73 nodes per unit disk respectively. Over 42,000 flows are generated by the 1-minute experiment, and the 10-minute experiment generates over 54,600 flows.

Figure 5.9 (shared with a previous simulation) shows the topology of the network at the start of the

**TABLE 5.9: A table listing the All Pairs Experiment parameters.**

| | | |
|---|---|---|
| flow params | flow style | all pairs (a flow between every pair of nodes) |
| | number of flows | 41,905 and 54,615 (10 minutes) |
| node mobility params | mobility model | actual aircraft trajectories (ASDI) |
| | start date/time | July 9, 2015 13:00 CT |
| | duration | 1 minute and 10 minutes |
| | number of nodes | 264 to 269 (at 1 minute) to 304 (at 10 minutes) |
| | density | 4.11 to 4.18 (at 1 minute) to 4.73 nodes per unit disk (at 10 minutes) |
| application params | application type | TCP constant bit rate (CBR) |
| | flow rate | 100 kbps and 1 Mbps |

simulation. The network spans a good portion of the middle of the United States and the East Coast. The nodes in the simulated network follow the paths of actual aircraft flying on Thursday, July 9, 2015 at 1:00 p.m. The entire set of aircraft in flight at the given time was filtered to achieve the desired connectivity and density for this experiment. The nodes simulated represent only a small fraction of the total number of aircraft in the skies at the given time. The aircraft for this simulation were filtered to ensure that aircraft join but do not leave the network and that the network is able to maintain all of the aircraft in a single connected component using the DC-GG topology generation algorithm with the topology management framework. The purpose of this filtering is to (1) obtain a medium-density network where the all-pairs experiment could run in a reasonable amount of time, and (2) to ensure that any drop in PDR could be attributed to the routing protocol or congestion and not connectivity issues or node departures.

**TABLE 5.10: A table listing the results of the All Node Pairs Experiment.**

| | 1 minute, 100 kbps | 10 minutes, 100 kbps | 1 minute 1 Mbps |
|---|---|---|---|
| transmitted packets | 28,798,648 | 330,958,751 | 287,920,595 |
| dropped packets | 113 | 27 | 2,274,666 |
| received bytes | 29,757,841,609 (29.8 GB) | 344,042,181,888 (344.0 GB) | 293,952,719,180 (294.0 GB) |
| aggregate throughput | 3.96771 Gbps | 4.58723 Gbps | 39.1937 Gbps |
| PDR (packet delivery ratio) | 0.999996 | 0.999999918 | 0.992097 |
| mean one-way delay | 7.80229 ms | 7.07133 ms | 8.09107 ms |
| mean hops per packet | 28.1276 | 25.753 | 27.7813 |

Table 5.10 gives a summary of the results of these simulations. In the 1 minute, 100 kbps simulation nearly 28 million packets were transmitted and only about 113 were dropped, yielding a packet delivery ratio of 0.999996. The PDR remains above 0.99 for the other two simulations as well, even though the 1 Mbps

simulation encounters some congestion. Aggregate throughput is the total number of bytes successfully received divided by the simulation time. The 1 Mbps simulation had an aggregate throughput of over 39 Gbps and simulated the transmission of nearly 294 gigabytes (GB) of data. The mean one-way delay for the simulations ranged from about 7.07 to 8.09 ms. Note that this is the mean delay for all of the flows, some of whose packets traveled from Arizona to Maine, over many hops, and others whose packets traveled over only the single hop to a directly connected neighbor.

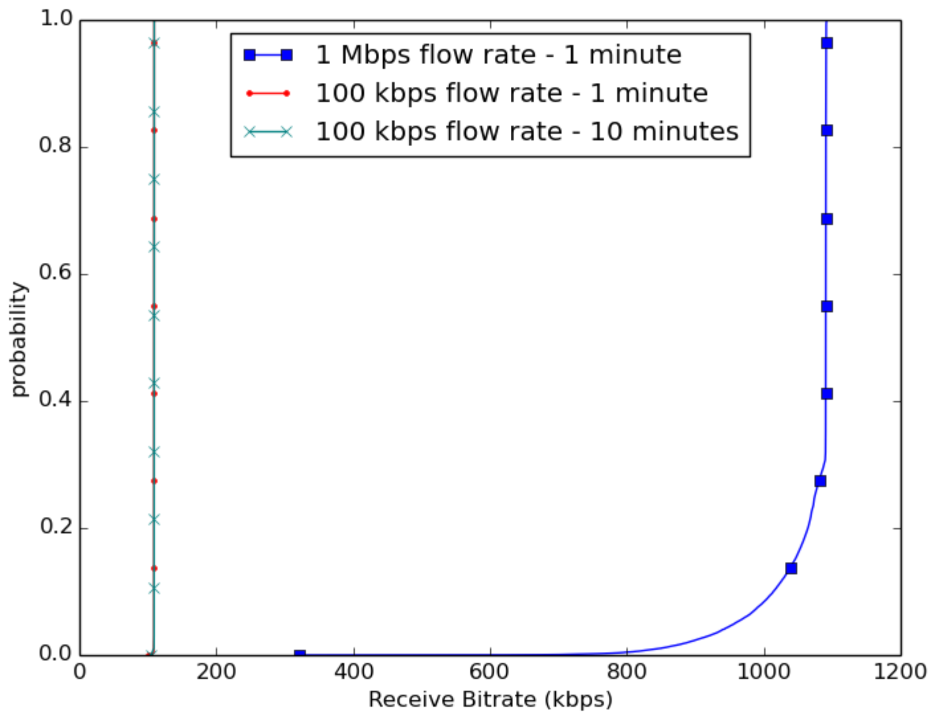


**Figure 5.20: The CDF of the throughput for the All Node Pairs simulations.**

Figure 5.20 shows the cumulative distribution function (CDF) for the throughput of the flows for the three simulations. The throughput analyzed here is also the receive bitrate. It is the number of bits successfully received in a flow divided by the number of seconds that elapsed from the time the first packet was received until the last packet was received. It should be noted that only the flows carrying application data are considered here. The reverse flows carrying only the stream of acknowledgment (ACK) packets are ignored for this and the next 3 plots. Both simulations sending application data at 100 kbps result in a throughput of 109 kbps. The throughput is over 100 kbps because it includes the transmission of the overhead bytes in the TCP and IP headers, and the additional TAG header.

The throughput of the 1 Mbps flows is much more interesting. Notice that 30.9% of the flows have a throughput lower than 1090 kbps. The other 69.1% of the flows consistently have a throughput of about 1178 kbps. This is evidence of the fact that there is congestion in the network when the 1 Mbps flow rate is used for the tens of thousands of flows. Notice that these flows are still able to send and receive data, but at slightly lower rates. Because of the higher probability of visiting bottleneck links over long paths, the flows affected by the congestion are more likely to be those sending data over long distances.

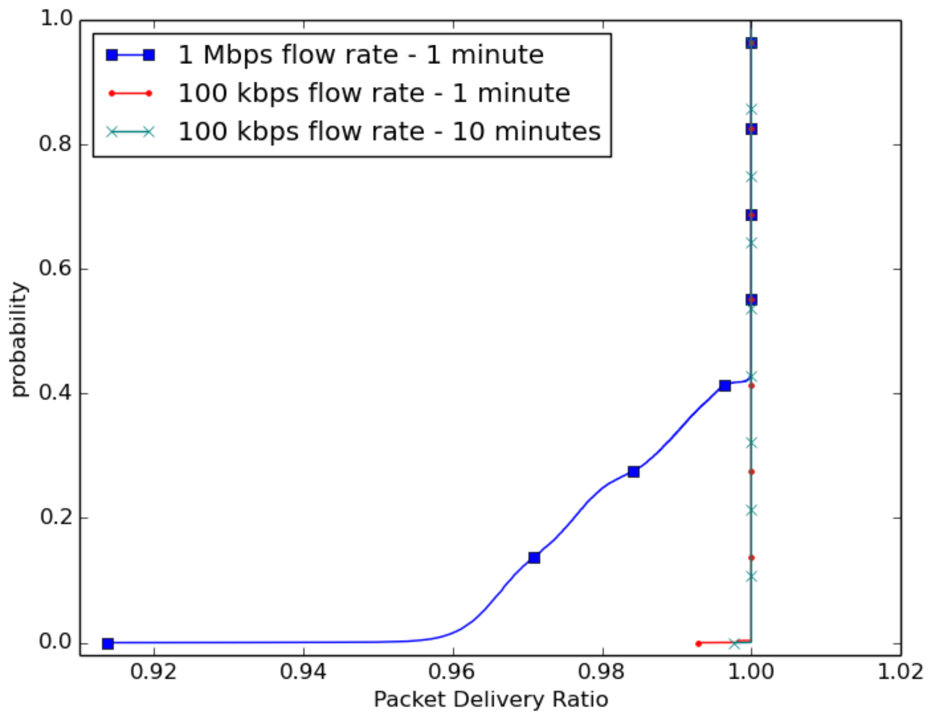


**Figure 5.21: The CDF of the packet delivery ratio for the All Pairs Experiment.**

In Figure 5.21 evidence of congestion can also be seen when looking at the packet delivery ratio (PDR). Note that the x-axis here has been shortened to show only PDRs of 0.91 and higher. For the 100 kbps simulations, the PDR is consistently 1.0 except for a small percentage of flows, most of which also have a PDR above 0.99. This is evidence that the topology management framework and geographic routing protocol are dependably routing the hundreds of millions of packets. However, when the rate is increased to 1 Mbps some of the links become congested and packets are dropped. For 41.9% of the flows, the PDR is less than 0.999. One factor contributing to this congestion is the fact that the network is sparse and spread out with a few critical links that connect the left half of the network the right half. A more dense network

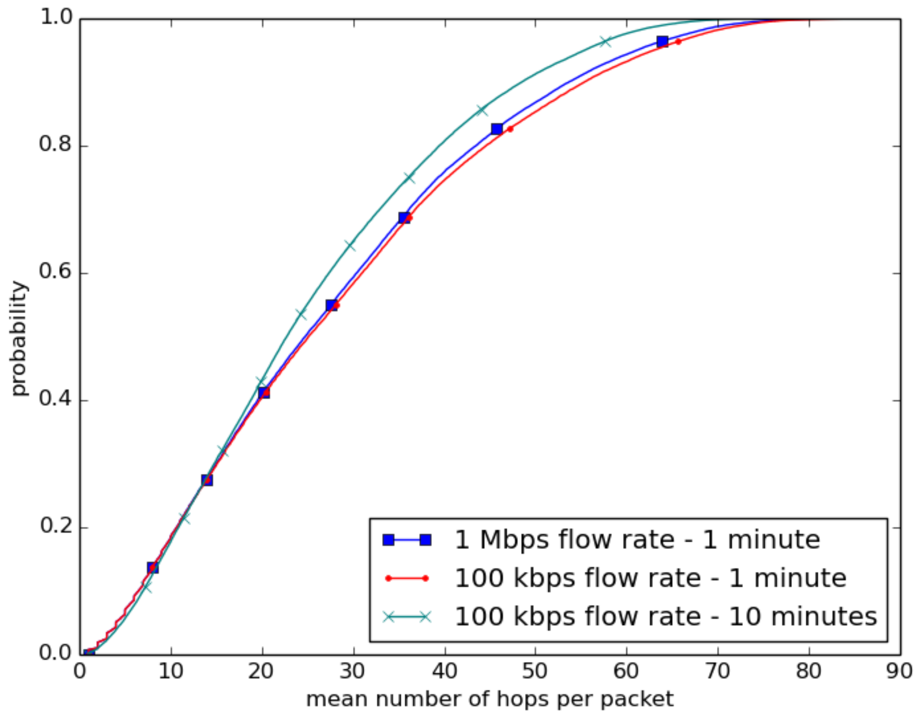


**Figure 5.22: The CDF of the mean number of hops per packet for each flow in the All Pairs Experiment.**

with more interconnections and fewer bottlenecks would likely yield better results.

A comparison of the cumulative distribution functions (CDFs) of the mean number of hops for each of the flows can be seen in Figure 5.22. Notice that there is little difference in the distribution for any of the simulations under about 15 hops. Notice also that the 10 minute 100 kbps rate simulation has a higher probability of having fewer hops after that point. This is likely due to the fact that more aircraft are added to the network during the extra 9 minutes in this simulation leading to more connections and better paths on average. The number of hops reported here includes only the hops for successfully received packets. Since in the 1 Mbps simulation, many of the packets are dropped, and there is a higher probability that those traveling over many hops will be dropped, there is an apparent improvement in the mean number of hops for the 1 Mbps simulation over the 100 kbps simulation.

Finally, figure 5.23 plots the cumulative distribution function of the mean one-way delay per packet for each of the simulations. Here, once again, the 10-minute simulation performs best and has a higher probability of having fewer average hops in its flows. This is again likely because of the introduction of new

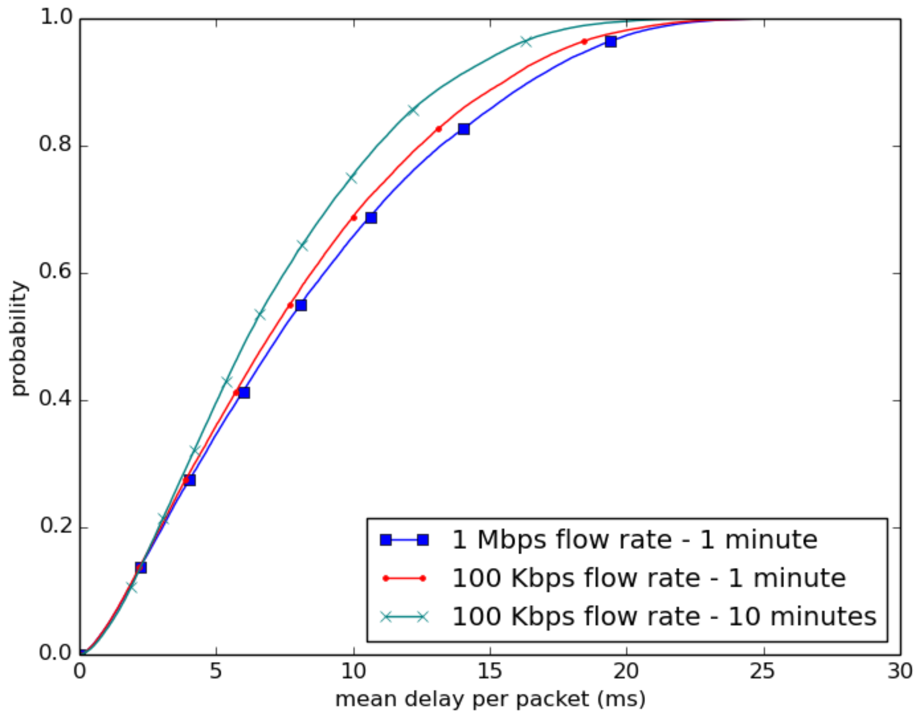


**Figure 5.23: The CDF of the mean one-way delay per packet for each flow in the All Pairs Experiment.**

nodes joining the network, increasing the network density, and creating shorter paths through the network. Because of the large numbers of packets queued in the congested simulation where tens of thousands of flows are trying to send at a rate of 1 Mbps, the mean one-way delay for the 1 Mbps simulation is higher than the delay for the 100 kbps 1-minute simulation.

This experiment demonstrates the following:

- The topology management framework and TAG are capable of handling over 40,000 flows and hundreds of millions of packets.

- Despite frequent changes in the network topology TAG is able to route the packets with near perfect packet delivery ratio.

- In the simulated network, congestion is experienced when a 1 Mbps flow is generated between every pair of nodes. However, even in this congested state, data is able to flow, though at decreased rates.

## 5.3 Summary

This chapter described the simulation models added to ns-3 enabling these large-scale networks to be simulated, and then detailed a sequence of experiments investigating various aspects of the airborne network's performance. The topology management framework and TAG protocol are able to perform well in large-scale high-density conditions, over long durations, and are able to support tens of thousands of medium-rate flows. In addition, the network is able to cope with bit error rates as high as $1\text{x}10^{-9}$.

# CHAPTER 6: CONCLUSIONS AND FUTURE WORK

The field of airborne networking is just in its infancy, and this work has only touched on the many captivating questions that must be answered, and the many problems that must be understood and solved to make these networks functional and efficient. In the first section of this chapter, the contributions and conclusions of this work are listed. The remainder of the chapter introduces several areas of future work and presents ideas for improving the airborne networking architecture.

## 6.1 Overview of Main Contributions

This work advances the state-of-the-art in the areas of efficient distributed topology management and geographic routing. An overview of each of the two main contributions of this work is given in the following sections.

### 6.1.1 Efficient Topology Management

When a small number of steerable directional communication links, that focus their transmissions in a single direction, are used to connect nodes in a dense network, it becomes necessary to explicitly manage the physical topology of the network. A scalable framework for managing such a physical topology has been presented. The only inter-node communication necessary to enable this topology management is the periodic exchange of position information between nearby nodes. The effectiveness of the topology management framework relies on a specialized topology generation algorithm meeting a certain set of requirements. Most existing topology generation algorithms are unable to generate the degree-constrained planar topologies desired. Therefore, several novel topology generation algorithms were designed and compared. To manage the topology of the network each node uses the positions of nearby nodes and computes an opinion of what the topology should be among these nodes by using the topology generation algorithm. It then forms the connections of which it is a part in the opinion topology. There is no explicit communication between the nodes negotiating the forming of connections. The cost of this distributed management framework is that some of the nodes will point at a node that has a differing opinion of what the topology should be, causing it to not point back. This can happen when the sets of nearby nodes differ for the two nodes attempting to connect. However, with an appropriate topology generation algorithm and enough node density, these one-way connections don't have a significant impact on the network.

### 6.1.2 Geographic Routing Protocol

The topology of the airborne network is highly dynamic, changing as nodes join the network, leave the network, and move about at high velocities. This constant topological change makes it extremely challenging to route packets through the large-scale network. Proactively updating routing tables to deal with the topology changes requires that numerous overhead packets be sent, and can result in many packets being dropped while waiting for a routing table to be updated. On the other hand, finding full-length paths through the network to a destination, on demand, causes the first packets sent to a node to be delayed, and also requires that many overhead packets be sent to establish the path and to repair it whenever there is a change in the topology along the path. Any time the path is being repaired packets must be buffered (delayed) or dropped.

Instead of either of these approaches, this work introduces a new geographic routing protocol that is able to route data without the need to exchange any routing overhead packets, while ensuring the packets virtually always arrive at their destinations when a path to the destination exists. This is accomplished by using the position information of the nodes to make forwarding decisions at each hop along a path. Each packet has a small shim header added to it. This header includes information such as the approximate latitude and longitude of the destination node, the packet's mode, etc. Nodes then use only the information in the packet header and information locally available at the node to determine where to forward the packet next. Existing state-of-the-art geographic routing protocols greedily route the packet and use face routing to recover from local minima.

The second main contribution of this work is a geographic routing protocol called Topology Aware Geographic routing (TAG). TAG advances the state-of-the-art of geographic routing in 1) its ability to cope with topology changes while a packet is in flight, and 2) its use of local topology information to make better geographic routing decisions. TAG is agile and able to instantaneously respond to changes in the network and not drop or delay packets while a new path is formed. In addition, TAG does not require numerous overhead packets to be generated and exchanged. The cost of this type of routing is the small extra header attached to each packet, and a small number of extra hops, on average, in the path to the destination.

## 6.2 Conclusions

This section lists the conclusions that are drawn from this work.

### 6.2.1 Topology Management

- It is possible to manage the topology of a large-scale network of nodes (connected by directional data links), in a distributed manner with no communication between the nodes except for the exchange of position information between nearby nodes.
- Aircraft that already use a separate system (ADS-B) to exchange position information with nearby nodes can manage the network with absolutely no new overhead and no topology management overhead messages on the data links, leaving the entire link capacity available for use by other applications and protocols.
- The distributed management of the network topology can be realized by having each node track the current position of nearby nodes, and allowing each node to use this information to periodically and independently compute, using a topology generation algorithm, an opinion of what the topology should be among its nearby nodes. Then, given this local topology, forming or attempting to form the connections of which it is a part in the computed topology.
- A topology generation algorithm that works well, in practice, at connecting nodes into an acceptable network topology is degree-constrained Gabriel graph (DC-GG). This graph is generated by first finding the Unit Gabriel Graph among a set of nodes, then finding a degree-constrained spanning tree that is a sub-graph of the Unit Gabriel Graph, and finally adding edges from the Unit Gabriel Graph, longest first, until no more edges can be added without violating degree constraints.
- Using DC-GG to form a global topology in a distributed manner yields topologies where less than 2% of the data links that were commanded to point result in one-way connections, for networks with densities greater than about 17 nodes per unit disk (1093 nodes in the continental United States).
- In a topology managed by the distributed topology management framework and using DC-GG as the topology generation algorithm 2.08% of the edges will be added or removed each iteration, assuming an update interval of 1 second. This means that on average the time between re-pointing events will be about 96 seconds.

- When DC-GG is used with the topology management framework with node densities greater than 20 nodes per unit disk (1286 nodes in the continental United States), the shortest path in the global topology will be at least three times as long as the shortest path in a topology where each node has an unlimited degree and where no planarity constraint exists.

### 6.2.2 Geographic Routing

- Packets can be routed in fewer hops, on average, by a geographic routing protocol, if that protocol makes use of the opinion topology generated as a by-product of the topology management.
- Unlike the other state-of-the-art geographic routing protocols, TAG is able to route packets that are in-flight when the topology changes without risk of them being trapped in a routing loop.
- TAG routes packets in fewer hops than GOAFR+ or GFG in a simulated medium-scale airborne network.

### 6.2.3 Airborne Network

- In a static medium-density network TAG packets must travel through 37% more hops, on average, than globally routed packets.
- The topology management framework and TAG are able to successfully route low-rate TCP and UDP traffic through a network of over 7,800 nodes (122 nodes per unit disk).
- TAG's ability to quickly adapt to changes in the network topology can cause packets to arrive out of order and cause TCP to unwittingly assume there is congestion in the network when there is not. Because of this, high-rate TCP flows generally suffer from low throughput with TAG.
- The topology framework and TAG are able to cope with bit error rates as high as $1\text{x}10^{-9}$ with minimal impact on the throughput.
- The topology management framework and TAG are able to handle over 40,000 flows sending data between all pairs of nodes in a network of 270 aircraft at per-flow rates of 100 kbps and 1 Mbps, although some congestion is experienced at the 1Mbps rate.

## 6.3 Future Work

### 6.3.1 Trajectory Prediction

Commercial aircraft are unique when compared to many other types of mobile nodes, in that both their future short-term and long-term positions can be predicted fairly accurately. This section describes how this fact could be used to improve the algorithms and protocols supporting an airborne network.

#### 6.3.1.1 Short-Term Prediction

Unlike people or cars, in-flight aircraft cannot in an instant change their direction of travel. Also, excluding maneuvers immediately after takeoff or while preparing to land, most commercial aircraft essentially fly straight-line paths between departure and destination airports with occasional gentle turns or infrequent deviations from a planned route. This means that predicting where a commercial aircraft will be 10 seconds, 30 seconds, or even a minute into the future can be done with relatively high accuracy. This fact can help nodes anticipate topology changes or better deal with the realities of imperfect systems, such as missed position updates.

Suppose, for example, that a local node was only able to successfully receive half of the position updates sent by a distant node. The position of the remote node could be extrapolated given the sparsely received information (including velocity info), and lessen or eliminate the impact of these losses.

Using short-term prediction, a node could compute not only the current local topology but also predicted future topologies among the nodes in its community. During each iteration of the topology management framework, a node could extrapolate the positions of its community nodes 5 seconds, and 10 seconds into the future, and compute topologies for these anticipated node positions. This would essentially provide a prediction of the topology changes over the next 10 seconds. The resolution and range of these predictions could obviously be changed, assuming computational resources were available. In this way, a node would essentially be able to predict changes in the topology, and link resources could be managed more efficiently. For example, if a critical change in the topology was expected to occur two seconds in the future, and it was anticipated that it would take two seconds to re-point and form that connection, the node could begin the process now, breaking a redundant connection early. One could imagine many similar optimizations facilitated by knowing where nodes will be and what the topology will look like some number of seconds in the future.

#### 6.3.1.2 Long-Term Prediction

Commercial aircraft also are highly predictable in the long-term. All civilian aircraft must file a flight plan before take-off. For commercial air traffic, these flight plans generally include an ordered list of waypoints. These waypoints are selected from a standard pre-defined set of geographical points spread across the globe. In addition, a custom waypoint can be defined as an offset from a pre-defined point, and used in a flight plan. This basic flight plan data is exchanged between nodes by the ADS-B system and is maintained in an FAA database. By making assumptions about the departure times and travel velocities, one can determine where all of the aircraft will be for hours into the future. These predictions would be fairly accurate except when flight plans change, aircraft deviate significantly from the flight plan, or flights are canceled.

This very accurate forecasting would allow for detailed analysis of the expected network coverage and anticipated networks topology hours into the future. Some have suggested that each node could even pre-compute its time-dependent topology before it takes off[TGS$^+$08]. Once in flight, at pre-defined times, the aircraft would change where its directional links were pointing. This scheme would not allow for changes after the pre-computation phase and would rely on assumptions about aircraft velocity. My distributed framework requires no communication overhead on the directional links but is able to respond to velocity or flight plan deviations and changes.

### 6.3.2 Simulation Speed and Accuracy

#### 6.3.2.1 Caching Routes

The geographic routing protocol does not maintain any type of a routing table at the nodes. Packets make independent decisions at each hop of their path. This has the benefit of enabling packets to immediately respond to changes in the network topology. However, the cost of this agility is that similar calculations must be made at a node by each packet traveling the same path. This is wasteful, and it seems this information could be cached in some type of routing table. However, just storing off a route and associating it with some future time (a constant offset from the current time) after which it will no longer be valid would likely destroy the ability of the protocol to quickly respond to changes in the network. Consider that the mobile nodes in the network likely have some maximum velocity. For example, commercial aircraft don't travel faster than about 700 miles per hour. This physical limit can assist with intelligently caching the forwarding decisions, allowing significant savings in computation time, without sacrificing the ability to quickly react to changes in the network topology.

The following example describes how this intelligent caching could work. For simplicity, consider just the basic greedy routing mode for now. In greedy mode, packets are simply forwarded toward the directly connected node nearest the destination, unless the current node is nearest, in which case a local minimum has been reached. Suppose that a node has two directly connected neighbors: node *a* and node *b*. Suppose also that *a* is currently closer to the destination than *b*, but that *a* and *b* are flying towards one another such that before long *b* will be closer to the destination. Assume, further, that the network topology is constant. Notice that every packet destined for that same destination node will be forwarded to node *a* until the instant at which node *b* becomes closer to the destination, after which every packet destined for that destination will be forwarded to node *b*. If *a* and *b* are, for example, 14 miles apart, then in the worst case if both aircraft are flying directly at one another at the maximum velocity of 700 miles per hour (1400 mph closing rate), there is at least 36 seconds before *b* will be nearer to the destination than *a*. During this time packets could be blindly sent to *a* without actually calculating which node is closer. Likely an update could be made after 36 seconds and the time could be extended. More sophisticated estimations of the future positions of directly connected neighbors could also be used.

This technique could be implemented as a table on each node containing an entry for each destination to which flows currently sending packets through this node are destined. Each entry would include the destination node ID, the outgoing interface number, and a time at which this entry should be considered stale, and removed (not unlike similar fields in many routing algorithms).

Packets being routed at the node would look up their destination in the table, and, assuming the entry is not stale, the packet could be immediately forwarded out the interface in the table entry. If the entry was stale or non-existent, the packet would be greedily routed as normal, but just before forwarding the packet onward, a new entry for the given destination could be added to the table, including a new stale time, representing the earliest possible time that a change to the cached entry may be needed. In addition, any time a topology change affected the set of directly connected nodes, the table would need to be cleared so that new decisions could consider the impact of the changed topology on the forwarding decisions.

This caching scheme would yield substantial savings in computation time without sacrificing the ability to quickly react to changes in the network topology. For TAG, where the local topology information is also used to make these decisions, the determination of the expiration time for an entry would be a bit more complicated, but still possible. Packets in face routing mode could also benefit from this type of enhancement, though each row of the table would then need to also include a mode entry.

#### 6.3.2.2 Higher Fidelity

Early in this work a more realistic physical layer model was developed for a free-space optical communication link. Unfortunately, the high computational complexity of high fidelity physical layer simulations made this model too time consuming to use. Increasing the fidelity of the simulated optical links is another area of interesting future work.

To increase the accuracy of the simulations, a sophisticated link budget for the free-space optical communication links could be modeled; including the impact of altitude, atmospheric turbulence, distance between nodes, laser beam width, laser wavelength, visibility, diameter of the receiver, and so forth. In addition, the position and orientation of each link could be modeled. The channel, that includes all of the network devices, could then determine which links were able to communicate and at what signal-to-noise ratio. Interference could also be modeled in such a simulation, and bit errors could be introduced based upon this signal-to-noise ratio. The result would be a much more realistic simulation of the network, but also one requiring significantly more computation and even longer run times.

### 6.3.3 Avoiding Cloud Occlusion

The simulations in this dissertation assume that there are no clouds occluding the transmissions between nodes. Commercial passenger and cargo aircraft generally remain at a very high cruising altitudes (above 30,000 feet) for the bulk of their journeys. At these altitudes, the direct connections between aircraft, over the expected ranges, will generally be free from laser-attenuating clouds. The connections to aircraft climbing to their cruising altitude and connections to ground stations, however, could be problematic. Suppose that every aircraft had a decently accurate 3D model of the clouds currently in their vicinity. This information could be used by the topology management algorithm to avoid connections that would intersect clouds. To implement this, ray-tracing techniques from the computer graphics world could be employed. Each node would, when computing its local potential connections graph, send a ray between each pair of nodes in its community. If the ray intersected any of the objects representing clouds the connection would be removed from consideration. In this way, the topology of the network could completely avoid connecting through clouds. This would be a simple addition to the topology management framework. A potentially more difficult task may be maintaining an accurate 3D view of the clouds at each node.

### 6.3.4 Avoiding Congestion

The airborne network can become congested with certain routes through certain nodes being used more than others. One area of future work is developing techniques to counteract or avoid this congestion. One way to potentially reduce congestion would be to more fully support multipath routing. Two packets that leave a source node at about the same time could be allowed to take completely different paths through the network. There are cases in the TAG implementation where it is unclear which of two options is better, and therefore a default action is taken. If instead, at such a point, a random choice was made, it would lead to more diversity in the network paths, but also potentially more issues with TCP detecting network congestion when none exists.

Another idea for increasing path diversity and reducing congestion would be to use a method similar to GLSR[Med11]. As described in Section 2.2.5.5, in GLSR a node greedily seeks to simultaneously maximize the packet's advance toward the destination and minimize the packet's queuing delay when determining where to forward the packet. Congested would suddenly become less desirable from a greedy standpoint, and packets are motivated to use a different path, when possible. This would have the outcome of lowering or alleviating the congestion and utilizing more of the links. This could be implemented in TAG by first making TAG aware of the outgoing queue length on each data link attached to the node. In greedy mode, in addition to considering the local topology, the lengths of the queues holding packets ready to be transmitted across each link would also be considered. The less congested links would be favored for situations where multiple greedy next hops exist.

In face mode, the queue size could be considered when determining which direction around a face to route a packet. If either direction appears to require about the same number of hops, the lengths of the queues at the outgoing interfaces could be used to make an informed decision about the direction in which the face boundary should be explored.

### 6.3.5 Parallelization

The ns-3 network simulator provides support for parallelizing execution across multiple cores and multiple distributed compute resources. Unfortunately, this functionality is only supported for point-to-point network devices and communication channels. The simulations in this work use a hybrid between point-to-point network devices and wireless devices. Each standard point-to-point channel connects exactly two network devices, but the standard wireless channel includes all of the network devices that are a part of the wireless subnet. These airborne network simulations require that a network device be able to form a point-to-point-like connection with one node, and then later re-point and connect with a different node. Therefore, a channel is desired that includes all of the network devices, in the same way a wireless channel includes all of the wireless devices. However, this channel should only allow specific pairs of net devices to communicate with one another, like a set of point-to-point connections. As described in Section 5.1.2 the custom wireless point-to-point net device and channel in ns-3 fills this need. Unfortunately, this custom network device type and channel are not able to support parallelized execution as easily as the point-to-point versions.

One aspect of interesting future work is enabling the parallelization of these new hybrid network device and channel models. To understand how this might be done for the airborne network requires some background on discrete event simulation. The idea of parallel discrete event simulation is to divide the nodes among several compute resources, called ranks. The simulation for all the packets flowing between nodes in the same rank can be carried out independently by that rank, and it can proceed at its own pace. However, any packets flowing to a node in a different rank must use a message passing infrastructure to pass the packet to the other rank. In a discrete event simulation, events that are to execute at a given time are placed in a priority queue. After each event is processed, the event at the front of the priority queue is removed, and simulation time advances to the time associated with this event. Because of this, the simulation time can advance in a non-uniform way. For this reason, and the fact that each rank may have other factors influencing its execution speed, a simulation may progress at different rates on each rank. The fundamental problem with this is that it may be possible for a rank to receive a packet that was intended to be delivered to one of its nodes at a simulation time that is now in the past. To overcome this issue, some technique must be employed to avoid this situation, or to roll-back the simulation when it happens.

One parallel discrete event simulation technique supported by point-to-point links in ns-3 is the granted-time window method. In this method, ranks are granted a window of time into the future in which they are free to proceed without fear of receiving a message that should have arrived at a time in the past. Upon reaching the end of the window, each rank must wait for all the other ranks to catch up. Inter-rank messages are then exchanged by the ranks, before proceeding with the simulation in the next time window. Note that each point-to-point connection that spans multiple ranks has a delay associated with it. To set the granted-time window, the simulator finds the shortest delay on all the connections that span a given pair of ranks. In this way, even a message that was sent by one rank to another rank immediately at the start of a granted-time window, is guaranteed to have a receive event that is scheduled for a time later than the start of the next granted-time window (since the delay in crossing the link is at least as long as the granted-time window). The receiving rank is able to synchronize and receive messages from other ranks before it continues on to the next window (in which the receive event for the message would be processed).

Just the initial problem of assigning nodes to ranks appears challenging for a large-scale airborne network. Suppose, you decide to start out small and allocate only 4 ranks to a simulation of an airborne network spanning the continental United States. One could assign nodes to ranks based on their starting geographic position. Aircraft in the northwest region at the start the simulation, for example, could be assigned to one rank, and aircraft starting in the southeast another, and so on. Since most aircraft only communicate directly with other nodes nearby, initially only the nodes near the rank boundaries would need to pass messages between ranks. However, as the simulation progressed, nodes would leave their initial geographic region. For example, a node heading from Albuquerque, New Mexico to Atlanta, Georgia would eventually cross the dividing line between the southwest and southeast regions. After it crosses this boundary it may start directly connecting with more and more nodes that are assigned to a different rank. As the simulation progresses it is likely that fewer and fewer nodes would be in the region where they started. Depending on the duration of the simulations and the number and size of regions, it may not take long to lose all of the benefits of running in parallel, and instead, be overwhelmed by the overhead necessary to synchronize the ranks.

It is possible one could devise some scheme in which nodes could essentially move between ranks during the simulation as they moved in and out of geographic regions. This, however, may require some careful handling of and transfer of state information from one rank to another, and would likely require significant changes to the existing simulation code. Another potential solution might be to assign nodes to ranks based on some other quality that is not as time-variant. Suppose, for example, that all the nodes leaving California and heading for the east coast of the United States were assigned to one rank, and all the nodes just leaving Florida and headed toward New England were assigned to another rank. This type of a partitioning would likely avoid the severe degradation over time, but may still require a lot of message passing between ranks since the nodes would directly connect with other nodes whose destinations areas were different.

Another issue that must be addressed to support this type of parallelization is determining the appropriate granted-time value. If this value is set too small the result would be an increase in synchronization overhead, allowing ranks to get a smaller amount of work done during each iteration. However, if the value were set too large, an event could arrive at a rank after its execution time had already passed. Unlike traditional point-to-point connections, the connections between airborne nodes in our simulations do not have a specified fixed delay. Instead, the delay is essentially a combination of the transmission time and the propagation time, which for free-space optical communication links is actually just the speed of light through the atmosphere. If a lower bound can be placed on the distance between any two aircraft in our simulations, an appropriate granted time could be determined. If the simulation could ensure, for example, that two aircraft will never fly closer to one another than 5 km, then the granted time could be set to about 15 microseconds. A small value, but long enough to transmit about 21 kilobytes of information at a rate of 10 Gbps.

There may be other parallelization schemes that could also be considered, such as optimistic schemes, in which, no effort is expended to ensure against events arriving at rank at a time later than their expected execution time. An optimistic parallel discrete event simulator, instead, rolls-back the execution to the point where the event was scheduled upon receiving an old event. The simulation then begins re-executing from there. There would likely be significant effort to implement such a scheme in ns-3, but it may also be an option to consider for our dynamic network.

Regardless of how it is accomplished, being able to simulate these large-scale networks in a distributed manner is essential moving forward.

## 6.4 Conclusion

This dissertation has demonstrated that a topology management framework and a routing algorithm can be developed that are capable of connecting thousands of commercial aircraft, flying normal routes, into a large-scale airborne network sufficient to support tens of thousands of 1 Mbps data flows.

Communication networks of the future will be vastly different from those we are familiar with today. Large-scale airborne networks are quickly becoming a reality as companies like Google and Facebook actively seek to deploy and control networks that will be able to reach more of the world's population. The protocols and methods described in this dissertation address some of the issues with managing connections and routing data within these high-capacity continental-scale airborne networks. It is my hope that these protocols will help to enable these networks to really "take off".

## REFERENCES

[ACK04] N. Ansari, Gang Cheng, and R.N. Krishnan. Efficient and reliable link state information dissemination. *Communications Letters, IEEE*, 8(5):317–319, May 2004.

[APC$^+$12] L. C. Andrews, R. L. Phillips, R. Crabbs, T. Leclerc, and P. Sauer. Final report: Channel characterization for free-space optical communications. Technical report, Florida Space Institute, University of Central Florida, 2012.

[APH01] Larry C. Andrews, Ronald L. Phillips, and Cynthia Y. Hopen. *Laser Beam Scintillation with Applications (SPIE Press Monograph Vol. PM99)*. SPIE Publications, March 2001.

[BHJ$^+$12] Zachary C. Bagley, David H. Hughes, Juan C. Juarez, Paul Kolodzy, Todd Martin, Malcolm Northcott, H. Alan Pike, Ned D. Plasson, Brian Stadler, Larry B. Stotts, and David W. Young. Hybrid optical radio frequency airborne communications. *Optical Engineering*, 51(5):055006–1–055006–25, 2012.

[BMSU99] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, DIALM '99, pages 48–55, New York, NY, USA, 1999. ACM.

[BRMD07] Q. Balzano, John Rzasa, Stuart Milner, and Christopher Davis. High capacity tactical networks with reconfigerable, steerable, narrow-beam agile point-to-point rf links. In *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1–7, Oct 2007.

[BRSS12] K.-D. Buchter, A. Reinhold, G. Stenz, and A. Sizmann. Drivers and elements of future airborne communication networks. In *German Air and Space Congress 2012, Berlin*, - Lilienthal-Oberth eV, Bonn, Sep 2012. German Society for Aeronautics and Astronautics.

[BSM$^+$09] D.M. Boroson, J.J. Scozzafava, D.V. Murphy, B.S. Robinson, and H. Shaw. The lunar laser communications demonstration (llcd). In *Space Mission Challenges for Information Technology, 2009. SMC-IT 2009. Third IEEE International Conference on*, pages 23–28, July 2009.

[CAC$^+$09] E. Ciaramella, Y. Arimoto, G. Contestabile, M. Presi, A. D'Errico, V. Guarino, and M. Matsumoto. 1.28-Tb/s (32 $\times$ 40 Gb/s) Free-Space Optical WDM Transmission System. *Photonics Technology Letters, IEEE*, 21(16):1121–1123, Aug 2009.

[CaRCR15] Sérgio Conceição, Filipe Ribeiro, Rui Campos, and Manuel Ricardo. Novel ns-3 model enabling simulation of electromagnetic wireless underground networks. In *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, pages 9–16, New York, NY, USA, 2015. ACM.

[CCSM13] F. Cadger, K. Curran, J. Santos, and S. Moffett. A survey of geographical routing in wireless ad-hoc networks. *IEEE Communications Surveys Tutorials*, 15(2):621–653, Second 2013.

[CFG$^+$09] J. Cunningham, D. Foulke, T. Goode, D. Baber, B. Gaughan, M. Fletcher, D. W. Young, J. C. Juarez, J. E. Sluz, and J. L. Riggins. Long range field testing of free space optical communications terminals on mobile platforms. In *MILCOM 2009 - 2009 IEEE Military Communications Conference*, pages 1–7, Oct 2009.

[CRJ$^+$12] Egemen K. Cetinkaya, Justin P. Rohrer, Abdul Jabbar, Mohammed J.F. Alenazi, Dongsheng Zhang, Dan S. Broyles, Kamakshi Sirisha Pathapati, Hemanth Narra, Kevin Peters, Santosh Ajith Gogi, and James P.G. Sterbenz. Protocols for highly-dynamic airborne networks.

In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 411–414, New York, NY, USA, 2012. ACM.

[DOD05a] Implementation of network-centric warfare. Technical report, United States Department of Defense, Office of Force Transformation, 2005.

[DOD05b] Joint Concept of Operations for Global Information Grid NetOps. Technical report, United States Department of Defense, United States Strategic Command, 2005.

[DOD13] Unmanned systems integrated roadmap fy2013-2038. Technical report, United States of America Department of Defense, December 2013.

[DSSM15] Emrecan Demirors, Bharatwaj G. Shankar, G. Enrico Santagati, and Tommaso Melodia. Seanet: A software-defined acoustic networking framework for reconfigurable underwater networking. In *Proceedings of the 10th International Conference on Underwater Networks & Systems*, WUWNET '15, pages 11:1–11:8, New York, NY, USA, 2015. ACM.

[DTBW13] R.W. Devaul, E. Teller, C.L. Biffle, and J. Weaver. Balloon network with free-space optical communication between super-node balloons and rf communication between super-node and sub-node balloons, July 18 2013. WO Patent App. PCT/US2013/020,705.

[DTS05] Jason Derenick, Christopher Thorne, and John Spletzer. Hybrid free-space optics/radio frequency (fso/rf) networks for mobile robot teams. In LynneE. Parker, FrankE. Schneider, and AlanC. Schultz, editors, *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 263–268. Springer Netherlands, 2005.

[EM04] B. Epstein and V. Mehta. Free space optical communications routing performance in highly dynamic airspace environments. In *Proceedings of Aerospace Conference, IEEE.*, volume 2, pages 1398–1406 Vol.2, 2004.

[FAA05] Air traffic bulletin. Technical Report Special Issue # 2005-3, Federal Aviation Administration, August 2005.

[FAA10] Automatic dependent surveillance-broadcast (ADS-B) out performance requirements to support air traffic control (ATC) service; final rule, May 2010.

[FAA15] FAA aerospace forecast - fiscal years 2015-2035. Technical report, Federal Aviation Administration, 2015.

[FAA16] Air traffic by the numbers. Technical report, Federal Aviation Administration, March 2016.

[Fac16] State of connectivity 2015: A report on global internet access. Technical report, internet.org by facebook, February 2016.

[FCC16] 2016 broadband progress report. Technical Report GN Docket No. 15-191, Federal Communications Commission, January 2016.

[Fin87] G. G. Finn. Routing and Addressing Problems in Large Metropolitan-Scale Internetworks. University of Southern California ISI Research Report ISI/RR-87-180, March 1987.

[FS10] H. Frey and I. Stojmenovic. On delivery guarantees and worst-case forwarding bounds of elementary face routing components in ad hoc and sensor networks. *IEEE Transactions on Computers*, 59(9):1224–1238, Sept 2010.

[Gar14] Juliette Garside. Facebook buys UK maker of solar-powered drones to expand internet. *The Guardian*, March 2014.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GK00] P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, Mar 2000.

[Guy16] Jessica Guynn. Facebook's Aquila drone completes first test flight. *USA Today*, July 2016.

[GW02] Baksheesh Ghuman and Heinz Willebrand. Making free-space optics work. In *Free Space Optics: Enabling Optical Connectivity in Today's Networks*. 2002.

[HSSJ02] Zhuochuan Huang, Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo. Topology control for ad hoc networks with directional antennas. In *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*, pages 16–21, 2002.

[JM96] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[JMC$^+$01] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, pages 62–68, 2001.

[JRO$^+$09] A Jabbar, J.P. Rohrer, A Oberthaler, E.K. Cetinkaya, V. Frost, and J.P.G. Sterbenz. Performance comparison of weather disruption-tolerant cross-layer routing algorithms. In *INFOCOM 2009, IEEE*, pages 1143–1151, April 2009.

[JS09] Abdul Jabbar and James PG Sterbenz. Aerorp: A geolocation assisted aeronautical routing protocol for highly dynamic telemetry environments. In *International Telemetering Conference*, 2009. Las Vegas, NV.

[Kel14] Heather Kelly. Facebook looks to drones, lasers and satellites for internet access. *CNN*, March 2014.

[Kel15] John Keller. Darpa moving forward with effort to develop 100-gigabit-per-second airborne wireless data links. *Military & Aerospace Electronics*, March 2015.

[KGKS05a] 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*, NSDI'05, pages 217–230, Berkeley, CA, USA, 2005. USENIX Association.

[KGKS05b] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. On the pitfalls of geographic face routing. In *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '05, pages 34–43, New York, NY, USA, 2005. ACM.

[KGT$^+$09] N. Krishnamurthi, A. Ganguli, A. Tiwari, Bao-Hong Shen, J. Yadegar, and G. Hadynski. Topology control for future airborne networks. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1–7, 2009.

[KK00] Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[KST$^{+}$10] A. Kothari, Bao-Hong Shen, A. Tiwari, A. Ganguli, Shuli Xu, and D. Krzysiak. Performance characterization of ad hoc routing protocols with mobility awareness. In *Military Communications Conference, 2010 - MILCOM 2010*, pages 80–85, 2010.

[KSU99] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *in proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[KWZ02] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically Optimal Geometric Mobile Ad-Hoc Routing. In *6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), Atlanta, Georgia*, September 2002.

[KWZ03a] 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*, DIALM-POMC '03, pages 69–78, New York, NY, USA, 2003. ACM.

[KWZ03b] Fabian Kuhn, Rogert Wattenhofer, and Aaron Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '03, pages 267–278, New York, NY, USA, 2003. ACM.

[KWZZ03] 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*, PODC '03, pages 63–72, New York, NY, USA, 2003. ACM.

[L3:11] Mini-T2 transceiver. Technical Report 16-DSD/SS-089, L-3 Communications, Jul 2011. Rev 102.

[LHS03] N. Li, J.C. Hou, and Lui Sha. Design and analysis of an mst-based topology control algorithm. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1702–1712 vol.3, March 2003.

[LJDC$^{+}$00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 120–130, New York, NY, USA, 2000. ACM.

[LML05] B. Leong, S. Mitra, and B. Liskov. Path vector face routing: geographic routing with local face information. In *13TH IEEE International Conference on Network Protocols (ICNP'05)*, page 12, Nov 2005.

[Mag] Yael Maguire. Building communications networks in the stratosphere. Blog, Facebook Code, July.

[MC98] J.N. Martin and M.J. Colella. Broadband wireless communications via stratospheric halo aircraft. In *Military Communications Conference, 1998. MILCOM 98. Proceedings., IEEE*, volume 1, pages 45–49 vol.1, 1998.

[MC00] J.N. Martin and N.J. Colella. High-speed internet access via stratospheric HALO aircraft. In *Emerging Technologies Symposium: Broadband, Wireless Internet Access, 2000 IEEE*, pages 5 pp.–, 2000.

[Med11] D. Medina. *Geographic Load Share Routing in the Airborne Internet*. Utz, 2011.

[MHRR10] D. Medina, F. Hoffmann, F. Rossetto, and C.-H. Rokitansky. A crosslayer geographic routing algorithm for the airborne internet. In *2010 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2010.

[MHRR12] D. Medina, F. Hoffmann, F. Rossetto, and C.-H. Rokitansky. A geographic routing strategy for north atlantic in-flight internet access via airborne mesh networking. *Networking, IEEE/ACM Transactions on*, 20(4):1231–1244, August 2012.

[MM96] Rudolf Mathar and Jürgen Mattfeldt. Optimal transmission ranges for mobile communication in linear multihop packet radio networks. *Wireless Networking*, 2(4):329–342, December 1996.

[MNAK09] K. Miyao, H. Nakayama, N. Ansari, and N. Kato. Ltrt: An efficient and reliable topology control algorithm for ad-hoc networks. *IEEE Transactions on Wireless Communications*, 8(12):6050–6058, December 2009.

[Mur15] Madhumita Murgia. Google's project loon balloons may face ban in india. *The Telegraph*, December 2015.

[NAJ15] Ben Newton, Jay Aikat, and Kevin Jeffay. Simulating large-scale airborne networks with ns-3. In *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, pages 32–39, New York, NY, USA, 2015. ACM.

[NAJ16] Ben Newton, Jay Aikat, and Kevin Jeffay. Geographic routing in extreme-scale highly-dynamic mobile ad-hoc networks. In *2016 IEEE 24th International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems*, Sept 2016.

[NBPD$^+$04] Thomas E. Noll, John M. Bron, Marla E. Perez-Davis, Stephen D. Ishmael, Geary C. Tiffany, and Matthew Gaier. Investigation of the helios prototype aircraft mishap. Technical report, NASA, January 2004.

[New16] Casey Newton. Facebook takes flight. *The Verge*, Jul 2016.

[NGUS12] Truc Anh N Nguyen, Siddharth Gangadhar, Greeshma Umapathi, and James PG Sterbenz. Performance evaluation of the aerotp protocol in comparison to tcp newreno, tcp westwood, and scps-tp. In *Proceedings of the International Telemetering Conference (ITC)*, 2012. San Diego, CA.

[O'B89] Donald Q. O'Brien. Maze demystified. *New York Times*, July 26, 1989.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 234–244, New York, NY, USA, 1994. ACM.

[PR99] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, Feb 1999.

[RED13] John Rzasa, Mehmet Can Ertem, and Christopher C. Davis. Pointing, acquisition, and tracking considerations for mobile directional wireless communications systems. *Proc. SPIE*, 8874:88740C–88740C–10, 2013.

[RH10] George F. Riley and Thomas R. Henderson. The ns-3 network simulator. In Klaus Wehrle, Mesut Gnes, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 15–34. Springer, 2010.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, RFC Editor, January 2001.

[RW15] Mark D. Rintoul and Andrew T. Wilson. Trajectory analysis via a geometric feature space approach. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(5-6):287–301, 2015.

[San05] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, Jun 2005.

[Sim16] Tom Simonite. 10 breakthrough technologies: Project loon. *MIT Technology Review - Best in Tech: 2015*, pages 20–25, 2016.

[SLM15] M. Strohmeier, V. Lenders, and I. Martinovic. On the security of the automatic dependent surveillance-broadcast protocol. *IEEE Communications Surveys Tutorials*, 17(2):1066–1087, Secondquarter 2015.

[SPM$^+$11] L.B. Stotts, N. Plasson, T.W. Martin, D.W. Young, and J. Juarez. Progress towards reliable free-space optical networks. In *Military Communications Conference, 2011 - MILCOM 2011*, pages 1720–1726, Nov 2011.

[TGK$^+$09] A. Tiwari, A. Ganguli, A. Kothari, S. Avadhanam, J. Yadegar, M. Compton, and K. Hopkinson. Feasibility of communication planning in airborne networks using mission information. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1–7, 2009.

[TGS$^+$08] A. Tiwari, A. Ganguli, Ashwin Sampath, D.S. Anderson, Bao-Hong Shen, N. Krishnamurthi, J. Yadegar, M. Gerla, and D. Krzysiak. Mobility aware routing for the airborne network backbone. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, 2008.

[TNAK10] Ngo Duc Thuan, H. Nishiyama, N. Ansari, and N. Kato. On performance evaluation of reliable topology control algorithms in mobile ad hoc networks. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5, 2010.

[TP13] E. Teller and W.G. Patrick. Balloon clumping to provide bandwidth requested in advance, November 21 2013. WO Patent App. PCT/US2013/035,959.

[USA07] Airborne network architecture, system communications description & technical architecture profile. Technical report, USAF Airborne Network Special Interest Group, October 2007.

[Vol00] Aircraft situation display to industry: Functional description and interface control document. Technical Report Version 4.0, Volpe Center, August 2000. Report no. ASDI-FD-001.

[YHSJ15] David W. Young, Hugh H. Hurt, Joseph E. Sluz, and Juan C. Juarez. Development and demonstration of laser communications systems. *Johns Hopkins APL Technical Digest*, 33(2):122–138, September 2015.