

## Air Force Institute of Technology AFIT Scholar

---

Theses and Dissertations

Student Graduate Works

---

6-13-2013

# Dynamic Network Topologies

Heather A. Lingg

Follow this and additional works at: <https://scholar.afit.edu/etd>

Part of the [Digital Communications and Networking Commons](#)

---

### Recommended Citation

Lingg, Heather A., "Dynamic Network Topologies" (2013). *Theses and Dissertations*. 883.  
<https://scholar.afit.edu/etd/883>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact [richard.mansfield@afit.edu](mailto:richard.mansfield@afit.edu).



## DYNAMIC NETWORK TOPOLOGIES

THESIS

Heather A. Lingg, GG-12, DAF

AFIT-ENG-13-J-04

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

***AIR FORCE INSTITUTE OF TECHNOLOGY***

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED  
DISTRIBUTION STATEMENT A

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States

AFIT-ENG-13-J-04

## DYNAMIC NETWORK TOPOLOGIES

THESIS

Presented to the Faculty  
Department of Electrical and Computer Engineering  
Graduate School of Engineering and Management  
Air Force Institute of Technology  
Air University  
Air Education and Training Command  
in Partial Fulfillment of the Requirements for the  
Degree of Master of Science in CyberSpace Operations

Heather A. Lingg, B.S.C.S.

GG-12, DAF

June 2013

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DISTRIBUTION STATEMENT A

AFIT-ENG-13-J-04

## DYNAMIC NETWORK TOPOLOGIES

Heather A. Lingg, B.S.C.S.  
GG-12, DAF

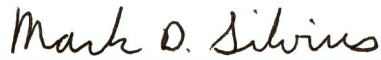
Approved:



Kenneth Hopkinson, PhD (Chairman)

31 May 2013

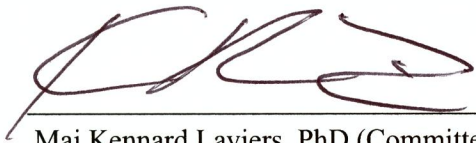
Date



Maj Mark Silvius, PhD (Committee Member)

31 May 2013

Date



Maj Kennard Lavers, PhD (Committee Member)

31 May 2013

Date

## **Abstract**

Demand for effective network defense capabilities continues to increase as cyber attacks occur more and more frequently and gain more and more prominence in the media. Current security practices stop after data encryption and network address filtering. Security at the lowest level of network infrastructure allows for greater control of how the network traffic flows around the network. This research details two methods for extending security practices to the physical layer of a network by modifying the network infrastructure. The first method adapts the Advanced Encryption Standard while the second method uses a Steiner tree. After the network connections are updated, the traffic is re-routed using an approximation algorithm to solve the resulting multicommodity flow problem. The results show that modifying the network connections provides additional security to the information. Additionally, this research extends on previous research by addressing enterprise-size networks; networks between 5 and 1000 nodes with 1 through 5 interfaces are tested. While the final configuration depends greatly on the starting network infrastructure, the speed of the execution time enables administrators to make infrastructure adjustments in response to active cyber attacks.

*To my husband for .... well, everything.*

## Acknowledgments

I would first like to thank my advisor, Dr. Kenneth Hopkinson, for his advice, guidance, and patience. This document is a long time coming; I hope my performance was worth the wait.

I would also like to thank Nicholas Kerner for his programming and troubleshooting assistance.

I also thank my supervisors, branch chiefs, and chain of command for their flexibility and understanding in my undertaking of this endeavor.

Finally, I thank my family and friends for their never-ending support and encouragement. In particular, my husband's love, advice, support, proofreading, and troubleshooting were invaluable. I could never have done this without the support of everyone. Thank you!

Heather A. Lingg



## Table of Contents

	Page
Abstract . . . . .	iv
Dedication . . . . .	v
Acknowledgments . . . . .	vi
List of Figures . . . . .	ix
List of Tables . . . . .	x
List of Symbols . . . . .	xii
List of Abbreviations . . . . .	xiii
1 Introduction . . . . .	1
1.1 Premise . . . . .	1
1.2 Goals . . . . .	2
2 Background and Previous Research . . . . .	5
2.1 Introduction . . . . .	5
2.2 Cryptology and the Advanced Encryption Standard . . . . .	5
2.3 Graph Theory . . . . .	7
2.4 Multicommodity Flow . . . . .	11
2.5 Previous Research . . . . .	16
2.6 Summary . . . . .	20
3 Methodology . . . . .	21
3.1 Problem . . . . .	21
3.2 Algorithm . . . . .	23
3.2.1 AES . . . . .	23
3.2.2 Steiner . . . . .	26
3.2.3 Network Traffic Routing . . . . .	29
3.3 Testing . . . . .	31
3.3.1 Experiments . . . . .	31
3.3.2 Metrics . . . . .	32
3.4 Summary . . . . .	35

4	Results and Analysis . . . . .	36
4.1	AES Results . . . . .	36
4.1.1	Execution Time . . . . .	36
4.1.2	Network Costs . . . . .	39
4.1.3	Network Differences . . . . .	44
4.2	Steiner Results . . . . .	48
4.2.1	Execution Time . . . . .	48
4.2.2	Network Costs . . . . .	52
4.2.3	Network Differences . . . . .	55
4.3	Comparisons . . . . .	58
4.4	Summary . . . . .	61
5	Conclusions and Future Work . . . . .	62
	Bibliography . . . . .	65

## List of Figures

Figure	Page
2.1 AES encryption process . . . . .	7
3.1 Original Network . . . . .	27
3.2 Shifted Network . . . . .	28
4.1 AES total times shows speed improvement over previous methods . . . . .	37
4.2 AES generational times do not have large deviations from the mean . . . . .	38
4.3 AES edge re-configuration occurs with no growth across interfaces . . . . .	40
4.4 AES routeing time dwarfs edge re-configuration time exponentially . . . . .	41
4.5 AES costs vary with traffic demands . . . . .	43
4.6 AES differences vary unpredictably over different configurations . . . . .	45
4.7 Steiner times show improvement over AES times . . . . .	48
4.8 Steiner interfaces contribute to overall generation time . . . . .	50
4.9 Steiner edge re-configuration times shows great improvement over AES method	51
4.10 Steiner traffic routing contributes most to overall generational time . . . . .	52
4.11 Steiner costs are similar to AES costs . . . . .	54
4.12 Steiner differences exponentially decrease as interfaces linearly increase . . . . .	56
4.13 Complete possible networks only assist differences if generated configuration is not sparse . . . . .	59

## List of Tables

Table	Page
3.1 AES Process in a Network . . . . .	27
(a) Original Network . . . . .	27
(b) SubBytes . . . . .	27
(c) ShiftRows . . . . .	27
(d) ShiftColumns . . . . .	27
(e) AddRoundKey . . . . .	27
4.1 Standard Deviation for Total Time for 10 Generations, AES, in units of network flow . . . . .	38
4.2 Standard Deviation for Average Time per Generation, AES, in seconds . . . . .	39
4.3 Standard Deviation for Edge Shift Time per Generation, AES, in seconds . . . . .	42
4.4 Standard Deviation for Traffic Route Time per Generation, AES, in seconds . . . . .	42
4.5 Standard Deviation for Average Total Cost, AES, by formula 3.2 . . . . .	43
4.6 Ratio of Configuration Cost vs. Total Possible Cost, AES, via formula 3.2 . . . . .	44
4.7 Standard Deviation for Average Total Topological Differences, AES, in units of network flow . . . . .	46
4.8 Active Edges in All Generations, AES . . . . .	46
4.9 Node Connectiveness, AES . . . . .	47
4.10 Standard Deviation for Total Time for 10 Generations, Steiner, in seconds . . . . .	49
4.11 Standard Deviation for Average Time per Generation, Steiner, in seconds . . . . .	50
4.12 Standard Deviation for Average Edge Shift Time per Generation, Steiner, in seconds . . . . .	53
4.13 Standard Deviation for Average Traffic Route Time per Generation, Steiner, in seconds . . . . .	53
4.14 Standard Deviation for Average Total Cost, Steiner, via formula 3.2 . . . . .	54

4.15	Ratio of Configuration Cost vs. Total Possible Cost, Steiner, via formula 3.2 . . .	55
4.16	Standard Deviation for Topological Differences, Steiner, in units of network flow	57
4.17	Number of Active Edges in All Generations, Steiner . . . . .	57
4.18	Node Connectiveness, Steiner . . . . .	58
4.19	Standard Deviation, Differences in units of network flow . . . . .	60
4.20	$\mu - PCI$ of Complete Graphs . . . . .	60
4.21	Connectiveness of Complete Graphs . . . . .	60

## List of Symbols

Symbol	Page
$G$ Graph . . . . .	7
$N$ Nodes . . . . .	7
$E$ Edges . . . . .	7
$K$ Commodities . . . . .	11
$n$ Number of Nodes . . . . .	11
$m$ Number of Edges . . . . .	11
$f$ Interface enumeration . . . . .	11
$k$ Commodity enumeration . . . . .	11
$i$ Source node enumeration . . . . .	11
$j$ Destination node enumeration . . . . .	11
$e$ Edge enumeration . . . . .	11
$u_{ijf}$ Edge capacity . . . . .	11
$e_{co}$ Operation edge cost . . . . .	12
$e_{cc}$ Congestion edge cost . . . . .	12
$x_{ijf}^k$ Flow percentage . . . . .	12
$P$ Path . . . . .	12
$d^k$ Commodity Demand . . . . .	12

## List of Abbreviations

Abbreviation		Page
DoD	Department of Defense . . . . .	1
IC	Intelligence Community . . . . .	1
IP	Internet Protocol . . . . .	1
IPS	Intrusion Protection System . . . . .	2
AES	Advanced Encryption Standard . . . . .	5
NIST	National Institute of Standards and Technology . . . . .	5
NSA	National Security Agency . . . . .	5
SHA-2	Secure Hash Algorithm 2 . . . . .	23

# Dynamic Network Topologies

## 1 Introduction

### 1.1 Premise

A network is a collection of devices, known as nodes, which exchange information over a means of communication, known as edges. Network devices are workstations, servers, routers, switches, and other network equipment. Edges are cables, radio waves, satellite links, or other methods of transmission. While much information exchanged over a network may be free and open to access (e.g. public web pages such as Wikipedia), some information (e.g. private communications or financial transactions) must be protected and limited to access. In the case of the Department of Defense (DoD) and the Intelligence Community (IC), networks contain information that is pertinent to national security. To protect this information, the network must also be protected.

Network defense consists of a series of mechanisms designed to protect information traveling through a network from any potential adversary that desires that information. The defense is multi-layered, depending both on application and transmission methods. Commonly, applications employ their own set of hardening techniques. A common technique is encryption, which is used to mask the actual content. At the network level, firewalls and various protocols, such as IPSec, can automatically reject data outright based on Internet Protocol (IP) address or port number. Still, at the lowest level, data must flow between devices over the network for the network to be of any use. More importantly, a valid data flow path from source to destination must be known for data to be properly received. This data flow across a network establishes the basic topology of the network, creating a virtual landscape that can be scanned and probed for vulnerabilities.



A network topology is the mapping of the interconnections between network nodes. Typically, enterprise networks are designed to have a certain topology which rarely changes. Updates and changes are generally time-consuming, costly, and must be executed in a precisely controlled fashion to avoid impacting network functionality. Other networks are much more flexible but can be much less reliable, depending on the transmission medium. The goal of this research is to allow changes at the base level of the network without incurring large costs and without adversely effecting network functionality. Most networks employed in commercial and government applications are statically defined. Data flows from device A to device B over a given static route Z. Once these networks are successfully scanned, an attacker can move to the act of actual exploitation on the assumption that the topology and traffic patterns will not change. The network is then infiltrated and information compromised.

An important key point is that a network topology, though physically stable, is also *virtual*. It is a landscape that can be changed at will by activating or deactivating network edges. Assuming that network traffic is properly re-routed to accommodate those changes, the nodes continue to operate as if no changes had occurred. Essentially, modifying the network topology on a regular basis provides an additional layer of defense for the network without compromising mission capabilities. Regular topology shifts force attackers to expend resources just to scan and re-scan the network, thus lessening the probability or severity of an attack. Such topology shifts can also accompany Intrusion Protection Systems (IPS) to automatically respond to an active attack.

## **1.2 Goals**

Before any network topology changes can take place, the changes must be known. Possible changes range from simple re-routing of a traffic path to physically disabling a connection. A new network topology must be generated and routes for network traffic re-established. The generation process must occur quickly enough to react to dynamic

network conditions, with or without human administrative assistance, at an enterprise level. If generating a new network topology takes days or even hours, the output is obsolete on arrival.

The new topology must not partition the existing network into multiple networks or fail to re-route any network traffic. Any inability to contact other users or use a network service renders the topology useless. Topology generation must also consider operational network costs and restraints: operation cannot consume excessive resources or use edges that do not exist. A new topology must also consider the configuration of previous generations because the entire goal of the process is to substantially modify the network topology; consistent re-use of a network link or a route simply misses the point. Similarly, if a solution only trivially changes the network topology as compared to previous generations, that solution should be rejected.

A final consideration is the security of the entire generation process: an attacker should be forced to re-scan a network to re-identify targets. If an attacker can perform the same calculations and arrive at the same solution, the process will quickly be reverse engineered and defeated. Given that an attacker can observe both the previous and current configurations, the generation process must execute with additional information.

The goal of this research, then, is to create a dynamic network topology generation and verification process. Implementation and testing of the output of the process on a physical network is beyond the scope of this thesis. Previous research in this area considered many of the factors discussed above, but to be truly useful all factors must be evaluated [1]. This research is faster and able to consider larger networks than previous research while maintaining significant confidence that the described constraints are accounted for appropriately. Additionally, this research provides flexibility to network administrators by incorporating flexible tolerance levels in the described constraints.

Two methods are used to modify the network topology: one based on cryptology algorithms, and one based on graph theory algorithms. A single method is used to re-route network traffic through the new configuration of network connections, and is used to finalize the network configurations for both topology change methods. The two topology change methods are compared for computational speed, operational cost, and security.

The remainder of this thesis is organized as follows: the second chapter provides background information and reviews previous research, the third chapter examines the methodology of this research, and the fourth chapter provides the results. Finally, the fifth chapter offers conclusions and suggestions for further research.

## **2 Background and Previous Research**

### **2.1 Introduction**

Several areas provide the background required for this research. These areas include both cryptology in general and the Advanced Encryption Standard in specific. Graph theory is also examined, including methods of finding the shortest path between a set of points. Additionally, linear programming and the multicommodity flow problem are examined, along with several solution methods for the multicommodity flow problem. Finally, previous research in the area of dynamic network design is reviewed.

### **2.2 Cryptology and the Advanced Encryption Standard**

This research uses the Advanced Encryption Standard (AES) in the modification of network topologies. It was accepted as a general standard in November 2001 by the National Institute of Standards and Technology (NIST) and shortly after by the National Security Agency (NSA) as the standard for encrypting classified information. The algorithm requires the plaintext, a substitution table, and a cipher key. The plaintext and the cipher key are provided by the user of the algorithm while the substitution table is provided by the algorithm itself. The algorithm follows four base steps in a series of iterative rounds to produce the ciphertext. The four steps are SubBytes, ShiftRows, MixColumns, and AddRoundKey. The number of rounds is dependent on key length and can be 10, 12, or 14 rounds for key lengths 128, 192, and 256 bits, respectively. Data is initially divided into blocks and each block is processed by the algorithm separately. Regardless of key length, data is always divided into 16 byte chunks, which are arranged in a 4x4 byte matrix [2] [3].

The SubBytes step uses a pre-generated table to modify the plaintext. The table is generated by applying an affine transformation over the multiplicative inverse of the finite

field  $GF(2^8)$ . The 16x16 byte table is static and is used in all implementations of AES. For each byte of plaintext data, a substitution is performed. The largest four bits of a given byte determine the row index of the table while the smallest four bits determine the column index. The entry at that spot on the table replaces the original data byte [2] [3].

ShiftRows is the second step. The 4x4 matrix of substituted data is grouped by rows and each row is circularly left shifted based on an offset. Each shift operates on a full byte of data. The offset starts at zero and increments with each row. This means that the first row is not shifted; the second is shifted by one, and so on [2].

MixColumns is the third step and re-arranges the columns. Each column of the shifted data matrix is treated as a polynomial over the finite field  $GF(2^8)$ , and is multiplied by a fixed polynomial,  $a(x) = (03)x^3 + (01)x^2 + (01)x + (02)$ , modulo  $x^4 + 1$  [2]. This operation modifies the data in place, using each byte of the column to affect the result of each final byte in the column. This provides diffusion in the output, adding additional security to the final ciphertext.

The final step is AddRoundKey. The user enters a password or pass-phrase at the beginning of the process. This password is used to generate a set of round keys; one round key is used per round of the process. The number of rounds dictates the required key size, and the round key generation process differs slightly, depending on these factors. The key generation process starts by generating a master key, and each round key consists of a set number of bits from this master key. These keys are generated before the AES algorithm begins and are used in this step. The AddRoundKey step itself consists of a basic XOR, using the round key and the mixed data matrix. This produces the final output of the round [2].

The pseudo-code of the process is shown in Figure 2.1. Note that a round key is added to the data before rounds begin, and that the columns are not mixed in the final

round. This is by design, to account for the accompanying decryption process. That decryption process is not used in this research.

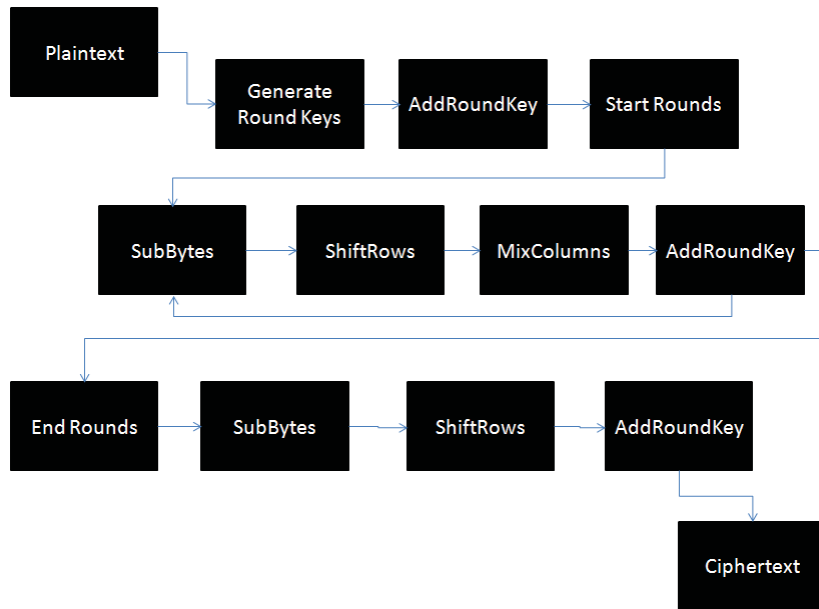


Figure 2.1: AES encryption process

This research utilizes the AES algorithm to modify the configuration of network edges. The methodology for this process is discussed in detail in Chapter 3.

### 2.3 Graph Theory

A graphical description of a network  $G$  includes the set of nodes  $N$  and the set of possible edges  $E$ . This relationship is described mathematically as  $G = (N, E)$ . An undirected graph consists of edges that are bidirectional between nodes while a directed graph contains edges that may only carry information in a single direction. This research assumes edges are bidirectional.

If the goal is to select edges to connect nodes, a number of shortest-path algorithms can connect given node-pairs one at a time. These algorithms include Dijkstra's and Bellman-Ford [4]. However, the goal is not only to connect a given list of node-pairs, but also to ensure the network is whole. Additionally, as the number of node-pairs grows, the time to generate the shortest-path list grows with it. If the selected algorithm connects pairs in sequence, such as Dijkstra's, rather than in parallel, this growth is linear.

Edges are described by weights, or costs, that are associated with that edge. For this research, all costs are assumed to be positive. For this phase, the capacity  $u_e$  of the edge is used as the edge weight. However, a higher capacity edge means more cost is associated with that edge. Practically, these operational costs include the cost of installing and maintaining the transmission medium as well as any power or administrative demands of the medium. For example, this would cover the cost to run a fiber line, the installation technicians, and the network administrators.

A minimum spanning tree aims to connect all nodes for the least amount of cost. By design, the network is only minimally connected, but the network is whole. However, the minimum connections may not be enough to carry all of the required network traffic. Additional edges must then be found to handle the network load using another algorithm, which defeats the use of a minimum spanning tree as a solution. As with the shortest-path algorithms, speed of the overall algorithm becomes a constraining factor.

A generalized Steiner tree finds the minimum-cost selection of network edges that connects all given node pairs. This is done by examining all node-pairs in parallel. The algorithm examines the connections between the set of source nodes and the neighbors of the source nodes. If this examination does not connect the node pairs, the process repeats. The original source node is replaced with the least-costly neighbor, and the algorithm uses the least-costly neighbor instead. The process continues until all node-pairs are connected,

not just the requested set of node-pairs. Additionally, the algorithm will identify a path between all node-pairs [5].

The generalized Steiner tree problem is essentially an optimization problem; network cost must be minimized. Linear programming is an established technique for solving such problems. Linear programming consists of an objective function and a set of constraints. Frequently, the objective function is a minimization or maximization problem. If a linear program is required by the constraints to have an integer solution, it is known as an integer program. A mixed integer linear program contains some constraints that are required to be an integer while other constraints are not under that restriction. A formal linear program is defined in the format [6]:

$$\min cx \tag{2.1}$$

Subject to:

$$Ax = b$$

$$x \geq 0$$

The first line defines the objective function. The matrix  $A$  contains the coefficients of the variables used in the model constraints. Similarly,  $b$  and  $c$  are corresponding vectors that contain equality information and the coefficients of the objective function, respectively. In this notation,  $x$  is also a vector of the model variables and non-negativity is imposed on the final solution values. A standard-form linear program is a minimization problem.

Also important to this research is the concept of dual linear programs. A linear program that can be inverted is known as a primal problem, whilst the inverse is known as the dual problem. For example, the primal minimization problem often has a corresponding dual maximization problem, and vice versa.



The Steiner tree algorithm is described as a dual linear program in [5]:

$$\begin{aligned}
 & \min \sum_{e \in E} u_e y_e & (2.2) \\
 \forall n \subseteq N & : n \in S & \sum_{e \in \delta(S)} \\
 \forall e \in E & : y_e \geq 0
 \end{aligned}$$

$$\begin{aligned}
 & \max \sum_{n \subseteq N: \exists i, n \in n_i} x_S & (2.3) \\
 \forall e \in E & : \sum_{N: e \in \delta(S)} y_e \leq u_e \\
 \forall n \in S & x_S \geq 0
 \end{aligned}$$

The primal minimization problem focuses on minimizing the total cost of all edges added to the network. Here,  $n$  refers to a given specific node,  $e$  to a given specific edge, and  $S$  to the set of edges comprising of the path. The variable  $y_e$  indicates if a network connection is present in the configuration, or not. The dual maximization problem increases the amount of flow,  $x_S$  on a given path until the capacity of the edge is reached. At that point, the path is added to the network. Use of the linear programming approach means the Steiner tree problem does not suffer from linear growth dependent on the length of the list of node-pairs as in the shortest-path approach. Instead, the growth is entirely dependent on the total number of nodes and increases exponentially. Since the linear problem considers the node-pairs in parallel, it also does not suffer from requiring additional edges later in the algorithm, as in the minimum spanning tree.

Typically, the generalized Steiner tree problem includes a phase to delete any extraneous edges. This phase is skipped in this research because the generalized Steiner

tree is only used to identify network edges, not paths for the network traffic. Removing edges at this step of the research limits the path selection later.

## 2.4 Multicommodity Flow

A transportation network is a network that carries goods or services over a set of edges where each edge has both a capacity and a congestion level. When multiple types of goods or services share the same infrastructure, it is known as a multicommodity flow network. Examples include airline route scheduling, fluids in pipes, and computer networks. A multicommodity flow problem takes a set of source and destination nodes, a set of edges between the nodes, and a set of network traffic demands to generate a set of paths for the network traffic. Without a path, the network information (or goods or services) cannot flow between the source and the destination. In the case of computer networks, demands include email traffic, streaming audio/video traffic, cloud applications, etc.. These demands are also known as commodities.

Even within computer networking, the applications of the multicommodity flow problem vary from basic network design to dynamic network topologies. A maximum concurrent flow problem is a set of commodities  $K$  that must all be routed through a network. The problem lies in defining the set of paths between every source and destination node pairs to carry the commodities so that no network edge is over its capacity.

Due to the variety of applications, a common set of terminology and symbology is required. A network is defined as a graph  $G$  with a set of nodes  $N$  and a set of node edges  $E$ . This relationship is described mathematically as  $G = (N, E)$ . The total number of nodes is  $n$  and edges is  $m$ . Network devices may be connected with multiple interfaces, enumerated by  $f$ .

Commodities are enumerated by  $k$ , a source node by  $i$ , and a destination node by  $j$ . For each edge  $e \in E$ , edge capacity,  $u_{ijf}$ , and cost information are known. Cost

information includes both the operational cost of including any given edge in the set of paths,  $e_{co}$ , and the congestion cost,  $e_{cc}$ , of using an edge  $e$  for another unit of flow from a commodity. Practically, these operational costs include the cost of installing and maintaining the transmission medium as well as any power or administrative demands of the medium. For this research, edge capacity and operational cost are fixed values. Edges are bi-directional and the costs and capacities may be different for each direction. The value  $x_{ijf}^k$  shows the percentage of flow over a given edge in path  $P$  for commodity  $k$  between source  $i$  and destination  $j$  over interface  $f$ . The demand that each commodity requires for transmission is also known, and is represented by  $d^k$ .

The multicommodity flow problem is an optimization problem; flow must be maximized or costs minimized. As a linear program, the concurrent multicommodity flow problem is formulated as:

$$\begin{aligned}
 \min \sum_{e \in E} \left( \sum_{k \in K} c_{cc} \right) + c_e & \quad (2.4) \\
 \forall e \in E : \sum_{k \in K} x_{ijf}^k \leq u_{ijf} & \\
 \forall k \in K : 0 \leq x_{ijf}^k \leq u_{ijf} & \\
 \forall k \in K : x^k = d^k &
 \end{aligned}$$

The formulation calls for the minimization of network costs while fully routing all commodities and not exceeding any edge capacity. Specifically, the objective function considers all commodity and edge costs. The first constraint ensures that the flow over an edge does not exceed the capacity of an edge. The second constraint ensures that commodities are not dropped, and the last constraint ensures that commodity demand is fully routed. The addition of constraining the flow to an integer would make this an integer program.

A number of properties are associated with linear programs. Most important to this research is the strong duality theorem. The strong duality theorem, given by [6], describes the optimality of a given pair of linear programs.

**Strong Duality Theorem** *If any one of the pair of primal and dual problems has a finite optimal solution, so does the other one and both have the same objective function values.*

Once a solution is found, optimality must be considered since a feasible solution does not guarantee an optimal solution. Optimality is important for both the general multicommodity flow problem and the dynamic network topology problem to ensure that all commodities are routed in the most efficient way possible. A dual linear program is created, so optimality is verified via the strong duality theorem. In this case, the dual to the minimum cost problem is the maximum flow problem.

However, finding the optimal solution often requires an unacceptable time to generate. The solution for the concurrent multicommodity flow problem is dependent on the number of nodes and number of commodities. Additionally, several paths may exist for a given source-destination-commodity combination. Approximation algorithms are required to find near-optimal solutions, guaranteed within a given margin of acceptability. This margin adds an error parameter  $\epsilon > 0$  to the problem. The parameter works with the network flow to provide a solution that is within  $(1 - \epsilon)$  of the optimal minimal cost.

Dantzig's simplex method given in [6] executes a series of operations, retaining a feasible solution to the problem after each step. The ellipsoid method in [5] improves on the simplex method and runs in polynomial time. The ellipsoid method allows for constraint violation during intermediate steps of the algorithm, but the final result is a feasible solution. The Ford-Fulkerson algorithm from [7] divides the network into two sets to find the maximum flow between two given nodes. However, these solutions all

assume a single commodity per network, and are inadequate for the maximum concurrent multicommodity flow problem.

This research's solution begins with randomized rounding to create an approximate solution. A network and its demands are first described as an integer program and relaxed to a linear program by removal of the integer constraints. The new linear program is solved, and the solution is rounded to obtain the integer solution. The rounding process is known as randomized rounding [5]. Young's work extends this by dropping the requirement of solving the linear program in favor of conditional probability operators that are independent of the optimal solution [8]. For the maximum concurrent multicommodity flow problem, the shortest path between a given pair of nodes is used as the independent operator. Garg and Konemann [9] extend Young's work further by routing additional units of flow per iteration of the algorithm. This allows the algorithm to run in  $(2k \log k) C_2 T_{sp}$  time, where  $k \geq \frac{m}{n}$ ,  $T_{sp}$  is the time to compute the shortest path, and  $C_2 = \frac{1}{\epsilon} \log_{1+\epsilon} \frac{m}{1-\epsilon}$ .

Fleischer further improves [9] when the number of commodities is large by selecting the  $\epsilon$ -approximately shortest path instead of the shortest path [10]. Fleisher first describes a linear program to maximize network flow:

$$\begin{aligned}
 & \max x & (2.5) \\
 \forall e : & \sum_{P: e \in P} x_{ijf} \leq u_{ijf} \\
 \forall k : & \sum_{p \in P^k} x_{ijf} \geq x d^k \\
 \forall P : & x_{ijf} \geq 0
 \end{aligned}$$

(2.5) focuses on maximizing the number of commodities routed, regardless of the network cost. However, the problem formulation is still bounded both by the capacity of each network link, and the commodity demands. The problem also demands that each

path chosen for a commodity has some flow over that collection of edges. The dual of this maximization problem is the minimization of the network costs:

$$\begin{aligned}
& \min \sum_{e \in E} u_{ijf} e_{cc} & (2.6) \\
\forall k, \forall e \in P^k : & \sum_{e \in P} e_{cc} \geq z^k \\
& \sum_{k \in K} d^k z^k \geq 1 \\
\forall e \in E : & e_{cc} \geq 0 \\
\forall k \in K : & z^k \geq 0
\end{aligned}$$

The dual introduces a new variable. Commodities are given weights,  $z^k$ , that control the shortest path between two nodes. Weights represent the cost of not routing the remainder of the commodity demand. The shortest path must be at least the weight of a commodity; that is, the commodity must be fully routed. Fleischer's work describes an approximation algorithm that solves the dual problem in  $O * (\epsilon^{-2}n(n + k))$  time [10].

Starting with a null solution, Fleischer's algorithm iterates until a proper solution is found. Each commodity is examined to route at least  $d^k$  units by performing a series of steps. The first step examines the shortest path as defined by the cost function  $l(e)$ , which will have some bottleneck  $u_{ijf}$ . Here,  $l(e) = e_{oc} + e_{cc}$ . The next step routes  $\min(d_k, u_{ij})$  along the path. The cost function is then multiplied by  $\left(1 + \frac{\epsilon \min(d_k, u)}{\max(e)}\right)$  to accommodate the change in the network that the routing imposes as well as the error parameter. When  $\sum_{e \in E} \max(e) l(e) \geq 1$  and all commodities are examined, the solution is considered proper and the algorithm stops. It is shown in [10] that the algorithm runs in  $O(\epsilon^{-2}e(e + k))$  time. In this way, the primal and the dual programs are solved together, and the dual solution is used to verify the approximate optimality of the primal.

The algorithm is upperly bounded by the total flow of the sum of the commodities. That is, once the traffic has been routed, the algorithm is guaranteed to stop [10]. The algorithm is lowerly bounded by the total flow divided by the number of commodities, which is at least one. The lower bound also serves to scale the optimal solution, if the algorithm fails to halt in the guaranteed time. The optimal solution and the lower bound are inversely proportional, and by manipulating the demands of the commodities, the algorithm retains the guaranteed run time [10] [9]. Further, Fleischer [10] proves:

**Fleischer Theorem** *An  $\epsilon$ -approximate solution to the maximum multicommodity flow problem can be obtained in  $O * (\epsilon^{-2}m(k + m))$  time.*

This research uses Fleischer’s algorithm for maximum concurrent multicommodity flow once the configuration of the network has been modified. By re-routing the network flow, this research ensures that the network remains mission-ready.

## 2.5 Previous Research

The dynamic network topology area builds from survivable network design, network load balancing and virtual network reconfiguration, and wireless communications.

Williamson and Shmoys define survivable network design as “low-cost networks that can survive failures of the edges” [5]. In a survivable network, traffic routes are adapted to compensate for the loss of an edge. Ho and Cheung present a generalized survivable network that is survivable independent of dynamic traffic [11]. Agarwal focuses on a survivable network with fixed capacities, and utilizes multi-commodity flows to create minimum-cost survivable networks [12]. This means that networks designed to be survivable are networks that can be effectively re-configured to generate multiple feasible topologies.

Network load balancing ensures that a given network edge is not over-congested. Traffic that congests an edge is dynamically re-routed to improve performance of the

network. In [13], fiber-optic networks are examined to iteratively distribute load among different light paths between nodes. Further, [14] examines the logical network topology to add and remove light paths to accommodate dynamic traffic demands. Tran, Casucci, and Timm-Giel examine a series of virtual networks sharing a physical network infrastructure. As virtual networks appear and disappear, they must be re-distributed to effectively utilize the physical infrastructure as efficiently as possible [15].

Communicating over a wireless network, by design, offers various strategies and protocols for adapting to networks whose nodes and edges can appear and disappear. Specifically, Erwin [16] and Compton [1] investigate mobile ad-hoc networks. Mixed integer linear programming is used to solve survivable network design problems, and both Erwin and Compton use mixed integer linear programming in their works [1, 16–18].

In both Erwin and Compton’s work, the following assumptions are made:

- There are a fixed number of nodes.
- Multiple commodities are routed on the network.
- Each commodity has a single source and destination node, for a unique source-destination-commodity tuple.
- Edges have fixed capacities.
- Nodes may have many interfaces.
- Two nodes are limited to one connection per interface.
- Nodes cannot connect to themselves.
- A node cannot possess more edges than interfaces.

The given assumptions also hold for this research. Compton additionally assumes that edges are bi-directional and have equal capacity in both directions. While this



additional assumption is convenient for his research, it is later shown that bi-directional equal-capacity edges are not a requirement for establishing a solution. From these assumptions, the following mixed integer linear program is developed by Erwin and expanded by Compton:

$$\min \sum_{(i,j,f) \in E} \left( \sum_{k=1}^K (v_{ijf}^k + b_{ijf}^k) x_{ijf}^k \right) + \sum_{(i,j,f) \in E} c_{ijf} y_{ijf} + \sum_{k=1}^K 1000 r^k m^k \quad (2.7)$$

subject to:

$$y_{ijf} = 0 \text{ or } 1 \quad \forall (i, j, f) \in E$$

$$y_{ijf} \leq a'_{ijf} \quad \forall (i, j, f) \in E$$

$$y_{ijf} = y_{jif} \quad \forall (i, j, f) \in E$$

$$x_{ijf}^k \leq y_{ijf} \quad \forall (i, j, f) \in E, 1 \leq k \leq K$$

$$x_{ijf}^k \geq 0 \quad \forall (i, j, f) \in E, 1 \leq k \leq K$$

$$\sum_{k=1}^K r^k x_{ijf}^k \leq cap_{ijf} \quad \forall (i, j, f) \in E$$

$$\sum_{j \in N} y_{ijf} \leq u_{if} \quad \forall i \in N, 1 \leq f \leq F$$

$$m^k = 0 \text{ or } 1 \quad \forall 1 \leq k \leq K$$

$$\sum_{j,f:(i,j,f) \in E} x_{ijf}^k - \sum_{j,f:(i,j,f) \in E} x_{jif}^k = \begin{cases} 1 - m^k & \text{if } i = s^k \\ m^k - 1 & \text{if } i = d^k \quad \forall i \in N, i \leq k \leq K \\ 0 & \text{else} \end{cases}$$

The objective function separately sums the usage and congestion costs for the network. The final term is added as a penalty for dropping commodities; any dropped commodity increases the final solution by a factor of 1000. The constraints restrict the solution to active edges of a network, restrict the variables to integer values, and ensure that capacities are not exceeded and that demands are fully routed. The final constraint,

added by Compton, restricts against simple loops. That is, a route traveling from node A to node B and back to node A is prohibited.

Erwin's network data contains up to 39 nodes, and each unique pair of nodes requires a commodity. Additionally, Erwin does not assume the network has enough capacity to route all commodity demands. A commodity-priority system is built into his algorithm to selectively drop commodities as needed. He begins by establishing a network backbone by use of a degree-constraining minimum spanning tree, and then expands the backbone into a full mesh network by use of several methods. The degree-constraining minimum spanning tree connects each node to no more than a given number of other nodes. A mesh network connects nodes in such a way so that each node can serve as an intermediate point between another pair of nodes. These methods include two link-adding heuristic approaches and solving the mixed integer linear program. The two heuristic approaches generate solutions significantly faster, but overall drop more commodities than the mixed integer approach. However, Erwin is unable to find a mixed integer solution for networks with more than 15 nodes within 30 minutes. Scalability of the mixed integer approach is a concern [16].

Compton expands Erwin's work by adding a metric to track significant differences in the generated network and a mechanism to periodically re-generate a solution. Compton's difference metric is used in this research to compare differences in generations of a particular network model. Additionally, Compton modified Erwin's objective function to include the term  $b_{ijf}^k$  as a penalty term for re-using edges. By penalizing edge re-use, the generated topology is more likely to route a commodity over a different route, thus increasing the differences between network configurations. This term is tracked by commodity so that a second commodity is not unjustly penalized for using a path another commodity previously used.

While Compton included Erwin's penalty term for dropping commodities, this research drops the term. If a network previously routed a set of commodities, modifications to the network configuration must not drop any commodities. Otherwise, the potential for significant mission impacts is present. Any modifications to the underlying network structure should not change existing network functionality.

Compton tests a number of network configurations after his modifications to Erwin's approach. Nodes number from 5 to 40, in 5-node increments, while interfaces are limited to a maximum of five. Nodes are the source of up to 3 commodities. Compton found that the number of nodes and interfaces contributed more to the generation time than the number of commodities [1]. Due to the long running times, several network configurations were not completed. For example, the slowest solution of the 30 node 4 interface 3 commodity configuration took 47 days to generate. However, network configurations with a high number of nodes and interfaces were found to contain edges that were not active across all generations. This ensures that an attacker listening on a particular network edge does not receive 100% of the traffic over that link. Compton's results show the security of adapting the network configuration, but the generation time and limited network size is unacceptable for operational use.

## **2.6 Summary**

This chapter discussed the underlying concepts and theory behind this research. The cryptology algorithm examined one approach to the broader dynamic network topology problem while the Steiner tree algorithm examined the second approach. Fleisher's multicommodity flow algorithm allows the re-routing of network traffic. Erwin and Compton's work serve as a baseline for comparisons. The next chapter will discuss the development of the methodology in detail.

## 3 Methodology

### 3.1 Problem

The physical network layer deals in circuits, transmission modes, and the design of transmission media; not in security. However, the raw bits of a packet must still pass over a physical connection. If a network edge is compromised, the most secure solution that preserves network operational status is to shut the edge off and reroute any traffic through the rest of the network. It is safest to keep even heavily encrypted data away from an attacker; while the field of cryptology steadily improves so does the field of crypto-analysis.

The more network edges that are compromised, the more difficult it is to keep network traffic from transmitting over a compromised edge. It is better to reconfigure the network so traffic is evenly balanced over secure network edges while maintaining network functionality. The reconfiguration must be generated and implemented quickly to minimize the amount of data sent over the compromised network connection. It must also take network and operational costs and constraints into account to maintain both network functionality, quality of service, and mission needs. Finally, the solution must apply enough change to the network topology relative to both current and past configurations such that an attacker must re-scan the network.

Speed is a vital factor, and directly ties to solution scalability. Ideally, the solution is fast enough to generate, verify, and implement a topology change during an active attack. The solution should also be fast enough to reconfigure the network at regularly scheduled intervals. However, operational networks range in size from small standalone networks servicing a specific mission need to large enterprise networks servicing entire agencies. As the network size and interconnectivity grows, the scale of the problem grows exponentially.

Operational costs and constraints include maintenance costs, connectivity requirements, service requirements, and service demands. Costs are both financial and man-power related, and also includes the cost of routers, switches, and other networking equipment. Connectivity requirements dictate which network nodes directly connect to other nodes. It may be impossible to directly connect two nodes due to distance or other limiting factors. Additionally, any given connection may be the compromised connection that administrators wish to avoid. Network services cannot be impacted by the change in network connections; each service requires a path between any two given nodes and consumes an amount of bandwidth over the edges in that path. For example, an email server has different network requirements than a streaming media server and both servers must be accommodated appropriately. Traffic must be carefully balanced to avoid over-saturating any given network edge.

Finally, the solution must force an attacker to expend resources and re-scan the network. The current and previous network configurations are considered when generating a new configuration to ensure the network is changed in a non-trivial manner. Insignificant changes allow an attacker to either ignore the changes or recall a previous configuration. Additionally, the solution algorithm itself requires a certain amount of security to ensure an attacker cannot perform the same calculations and arrive at the same set of network changes. This would allow an attacker to anticipate any changes and alter the attack accordingly.

This research assumes the network possesses redundant paths between network nodes. Any mission critical network where impacts to availability cause severe damage will contain redundant measures by DoD and NIST regulations to maintain approval to operate the network.

## 3.2 Algorithm

There are two main phases to the dynamic network topology algorithm presented here: the network edge shift phase, performed either by the AES cryptology algorithm or the Steiner tree algorithm, and the network routing phase, solved as a maximum concurrent multicommodity flow problem.

*3.2.1 AES.* The AES algorithm is used to 'encrypt' the network topology. Crucial to the security of the AES algorithm is the secret key. This is perhaps even more crucial in this application due to the difference in the outputted 'ciphertext'. In standard cryptology, the resulting ciphertext is nonsense data that is generally useless without access to either the key or the plaintext. In this application, it must be presumed that an attacker scans both the original 'plaintext' network topology as well as the re-configured 'ciphertext' network. Without proper key handling, reversing the process becomes trivial and an attacker will be able to anticipate network shifts. Each new network topology modification uses a secure hash chain to generate a new set of keys. The keys are used only once to prevent any mathematical analysis on the algorithm using the 'plaintext' and 'ciphertext' network topology information.

The hash begins with a user-generated pass phrase. The pass phrase is hashed using the Secure Hash Algorithm 2 (SHA-2) with a 512 bit block size. The resulting hash is chained 5000 times for additional security. The final result is used to generate the AES round keys.

The network topology itself is represented as a dynamic two dimensional array, with an optional third dimension to represent multiple network interfaces. Array indexes represent nodes while the value at that index represents the on/off status of the connection between each set of nodes. A one represents an active edge between nodes; a zero represents an inactive edge. Allowing for multiple network interfaces allows for the representation of routers, switches, and other network equipment that makes multiple

connections. Workstations and servers can have multiple interfaces if multiple network cards are available. Complete network information is stored in each half of the array; the status of the connection between nodes A and B is the same as status of the connection between nodes B and A. Connections between a node and itself are off by default. Symmetry is maintained between the halves to show bidirectional connections. While maintaining symmetry is convenient for this research, it is unnecessary for the general algorithm. A second network array is maintained to represent the availability of the connection between each set of nodes that are possible to connect. This adjacency matrix is checked before any network connection is modified to ensure that any network change is feasible. For example, a node may not connect to itself.

The AES algorithm is slightly modified for use in this research: the algorithm round computations are modified to account for three-dimensional space. The modification repeats the row shift step for the array slice containing the network interface information. This extra step is only performed when the input network contains multiple interfaces. It is not enough to perform the AES algorithm separately on each interface, as in typical two-dimensional cryptological functions. If the algorithm is performed separately on each interface, network connections would only shift along the same interface. Allowing connections to shift between network interfaces increases the number of possible network configurations.

It is important to address the impacts this modification has to the cryptanalysis of AES. In [19], the creators address differential and linear cryptanalysis through the propagation of bit patterns over the algorithm steps. Daemen and Rijmen state the ShiftRow has the properties:

- The column weight of a pattern at its output is lower bounded by the maximum of the byte weights of the columns of the pattern at its input.

- The column weight of a pattern at its input is lower bounded by the maximum of the byte weights of the columns of the pattern at its output.
- Byte weight is invariant as there is no inter-byte interaction.

This research copies the ShiftRow step to create the ShiftInterface step and the same properties hold. In Daemen and Rijmen's analysis, the bit patterns and column weights before and after the ShiftRow and MixColumn steps are examined. The additional step changes output patterns and weights to include the ShiftInterfaces step. The weights determine effectiveness of differential and linear cryptanalysis. The ShiftInterfaces step provides further diffusion of the bit pattern, since the step moves bits of one layer into another. The weight impact on each layer is nominal and the analysis provided by Daemen and Rijmen holds.

The algorithm is performed across the entire network array. One half of the network array is designated as the primary half; as modifications are made to the primary half, the secondary half is updated to reflect the primary half. This maintains the bi-directionality of the network connections. Bi-directionality is not a strict requirement; if edges are not bi-directional, the entire network array is required for the complete network configuration.

Table 3.1 shows how the AES algorithm is applied to the network array. Figures 3.1 and 3.2 show the input and output of the algorithm in graphical form. The substitution, shift row and modified mix column steps are applied to the network array. If required, the additional shift interfaces step is applied after the mix columns step. Finally, the round key step is applied. The number of rounds is dependent on network size.

Two verification steps check to ensure the network remains a single network with all nodes connected. The first verification checks to ensure all nodes are connected to a network by examining the values in the network array. If a node is found without any connections, an edge is added to the network array to establish a connection. The check



then searches for the node with the most connections, and a connection is removed from that node. This maintains a constant number of active network edges.

The second verification checks to ensure the network remains a single network instead of multiple independent networks. A breadth first search begins with any node and checks connectivity to all other nodes. If any node cannot be reached from the chosen root node, a sub-network is found. These sub-networks are connected to the root network by adding the appropriate edges to the network array. Again, edges are removed from the most connected nodes to maintain the number of active network edges. Both verification steps must pass for the network topology to be valid, and any corrections are re-verified to assure no additional topology issues are introduced.

The number of active links is maintained to control the verification steps, which must add links to fix any inconsistencies the network AES reconfiguration process. While a single network containing all nodes is required, a network topology with every network edge active is undesirable. The fully connected network does not allow further reconfigurations without removing network connections, and removal does not occur automatically. To preserve the general network state during the verification stage, it is best to remove links in equal number to those added. Otherwise, the configuration must begin again to find a valid network configuration.

Ten generations are performed per execution. This allows for the method to use the output from one generation as the input to the next; which in turn allows for the tracking of differences between generations.

*3.2.2 Steiner.* The second approach follows the Steiner tree approach and is implemented in a similar way to the AES approach. Two additional matrices are needed; one to store the distance between node pairs and one to store the next step in the path between node pairs. Only the 'next step' is required; at each step in the path, there exists the path between the 'next step' node and the destination node.

	A	B	C	D
A	0	0	1	1
B	0	0	0	1
C	1	0	0	0
D	1	1	0	0

(a) Original Network

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

(b) SubBytes

	A	B	C	D
A	0	0	1	1
B	0	0	0	1
C	1	0	0	1
D	1	1	1	0

(c) ShiftRows

	A	B	C	D
A	0	0	1	1
B	1	0	0	1
C	0	0	0	1
D	1	1	1	0

(d) ShiftColumns

	A	B	C	D
A	0	0	0	1
B	0	0	1	0
C	0	1	0	1
D	1	0	1	0

(e) AddRoundKey

Table 3.1: AES Process in a Network

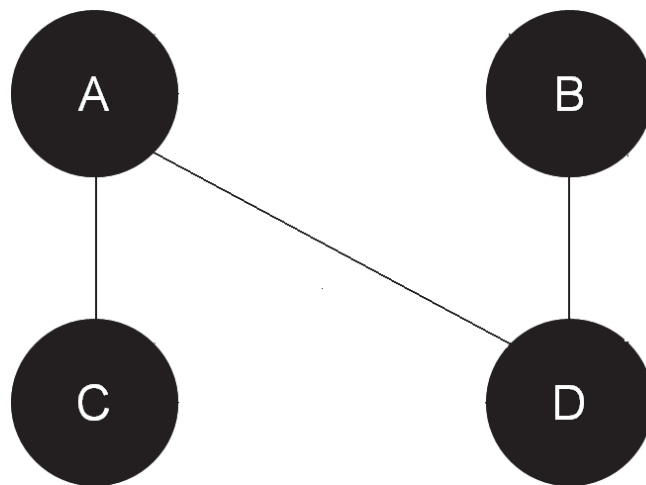


Figure 3.1: Original Network

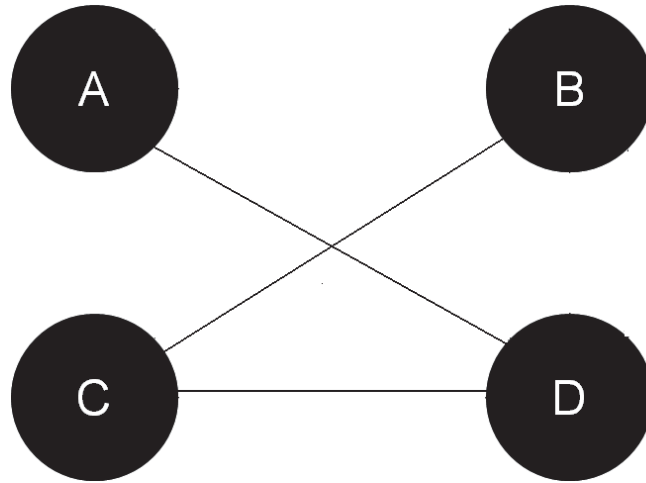


Figure 3.2: Shifted Network

The approach begins by hashing a user-provided pass phrase using the SHA-512 hash algorithm. The hash is hashed again creating a hash chain until enough bits to populate the network matrix have been generated. These bits are assigned to the network matrix to create a starting set of edges. If the hash produces a whole network, the algorithm moves immediately to re-routing the network traffic. The distance matrix is initialized to infinity, represented by a sufficiently large integer, and the next-step matrix is initialized to null. The distance matrix initialization depends on the presence of an edge in the adjacency matrix. If an edge exists, the distance is set to the activation cost. Otherwise, the distance is infinity.

Otherwise, the node pairs are ordered by priority, determined by the total demand. A higher priority is granted to a node pair with a larger demand. The Floyd-Warshall shortest-path algorithm is used to generate the set of shortest paths, using the set of possible edges as a reference. If a pair of nodes is not already connected, edges are added to the network as determined by the path found by Floyd-Warshall.

The same verification steps performed in the AES verification process are used here. If a node is in the network, but not part of the randomly-generated node pairs, it may or may not be included in the final network. The verification steps ensure that every node is connected and the network is whole. In practice, each node will be part of a node-pair and will be accounted for by the Steiner-based part of the algorithm. However, the verification steps remain a best practice.

Similar to the AES approach, ten generations are performed per execution. After each generation the distance matrix is re-initialized, based on the final configuration of the previous generation. Any included edge is initialized to 1000 instead of infinity. The value of 1000 was chosen as a sufficiently large number greater than any possible network demand. This weights the edge and discourages it from use in the next generation.

Edges unused in the generation are re-set to either the activation cost of the edge or infinity, dependent on the adjacency matrix. The next step matrix is re-initialized to null. This mirrors the original initialization. Each generation uses a new hash value. This sets the initial set of edges based on the user's pass phrase, with the phrase serving as a 'secret key'. As with the AES approach, this protects the results of the algorithm while allowing the algorithm itself to be open.

*3.2.3 Network Traffic Routing.* The second phase re-routes the network traffic over the re-configured network topology. Network traffic, like the network topology, is represented as a dynamic three column array. The three columns represent the source of the traffic, the destination, and the required bandwidth. The length of the array is determined by the number of unique source-destination pairs. Multiple commodities between the same source and destination are grouped together. The required bandwidth for a commodity is based on the type of traffic between the source and destination. Traffic types can be text, video, audio, etc.. This information, combined with the network

topology and network connection capacity information, defines the maximum concurrent multicommodity flow problem used in this research.

This research follows Fleischer's algorithm, detailed in Chapter 2 [10]. The network is initialized with a null solution and loops until the objective function value is at least one. The algorithm then examines each commodity to see if the demand has been met. If the commodity has not yet been routed, Dijkstra's algorithm is employed to find the shortest available path between the source and the destination. Note that the shortest available path may not be the shortest path; the algorithm considers path feasibility and will not route over edges that are not available or are overly congested. From there, the bottleneck capacity is determined and routed. The bottleneck capacity is defined as the minimum value between the available capacity on the edge and the commodity demand. Finally, the objective function value is updated.

The paths generated from Fleischer's algorithm are then examined and a set is selected to cover all network commodities. This function starts by examining each commodity's source node, and the edges leading from the source node to a path to the sink node for that commodity. The flow of the commodity is randomly split between appropriate paths. It is possible for half the packets from a commodity routed through one path and the other half routed through another path. All packets arrive at the destination node. Commodity splitting depends on the current capacity of an edge and the demand of the commodity. A commodity cannot be dropped for convenience; it must be routed. The loop moves to the next node in the path and routes the commodity from there, stopping when the next node is the destination. Once all commodities have chosen final paths, the output is checked for both exceeding edge capacities and for failing to route commodity demands, or dropping a commodity.

Paths are not weighted between generations. The paths chosen for one generation have equal opportunity to be re-chosen for the next, providing a similar set of edges is available.

The result is a set of paths and flow for all commodities through the modified network configuration.

### **3.3 Testing**

To test the described methods, sample networks are generated. Each method reconfigures the sample network, and a set of metrics are measured.

*3.3.1 Experiments.* Each network is described by the network size, number of interfaces, and maximum edge capacity. Network size is the number of nodes that are in a given network while number of interfaces determines the maximum number of physical network connections a given node can make. Every network contains at least one network interface to connect the network together; for example, a four-port switch is capable of four network interfaces. For this research, network sizes of 5, 10, 15, 30, 50, 100, and 1000 nodes are tested with 1, 2, 3, 4, or 5 interfaces.

The Network Simulator software is used to generate adjacency matrixes for each combination of network size and number of interfaces. For statistical significance, each combination is tested 5 times. The actual starting configuration of each method is generated via a hashed user pass phrase.

At minimum, each network node can be expected to communicate with at least one other node. Practically, each node will communicate with many other nodes. Minimum communication covers only a single standard network service, such as e-mail or file sharing. For this research, it is assumed that each node will communicate with one other node in the network. The source and destination nodes are randomly paired and pass a randomly generated amount of network traffic between the source and destination. The

amount of network traffic does not exceed the given maximum edge capacities. Demands of different types of traffic, such as text or video traffic are not considered individually. Rather, all traffic traveling from a source to a destination is grouped into a single commodity. A source node can share both a low and high demand traffic type with the destination; the generated amount of network traffic between the nodes is the total communication between the nodes.

In practical use, the possible network edges, the base network description, and the traffic requirements will be known from physical network configurations and customer demands. These values can be directly inputted into each method.

The accuracy of the network routing portion can also be varied, ranging from 0 to 1. However, the more accurate the selection of best paths, the slower the algorithm runs. For each combination with 100 or fewer nodes, the accuracy level is set at .3. For each combination with 1000 nodes, the accuracy level is loosened to .5 to accommodate the increase in nodes and routing requirements.

All experiments run on the same hardware. While the methods may perform faster with faster hardware, the hardware speed is not part of this research. The hardware contains an Intel i7 2.67 GHz quad-core processor with 6 GB of random access memory. Each processor contains two virtual processors for additional parallel processing. The operating system is Ubuntu version 12.04, 64-bit.

*3.3.2 Metrics.* A primary concern of this research is the minimizing the cost of a network topology generation while maintaining a valid output. To that end, several metrics are considered. The network topology generation process is tested for speed and scalability while the generated network topologies are examined for topological difference, security and cost. It is not enough to evaluate the generation process alone; the generation algorithm can produce and fast results for large networks without producing a usable operational solution. All metrics must be considered in validating the algorithm.

Speed and scalability are linked. The AES method runs in  $O((i^2n^3) + (\epsilon^{-2}e(e + n)))$  time while the Steiner method runs in  $O((n^3) + (\epsilon^{-2}e(e + n)))$  time. The second term in the notation represents the running time of the network routing phase, where  $\epsilon$  represents the accuracy level of the routing. The terms are separated because each phase runs in sequence and not in parallel; the network routing phase is dependent on the completion of the re-configuration phase. Each phase of algorithm execution measures run time of that phase, and the total run time of the method is captured.

Each step in each method affects the overall running time. Each step in the AES method runs in  $O(i^2n^3)$  time, for each of the three steps: ShiftRows, ShiftColumns, and ShiftInterfaces. The remaining steps, SubBytes and AddRoundKey, perform in  $O(in)$  and  $O(n)$  time, respectively. The three Shift steps are individually complex, requiring  $O(in)$  time per action for  $O(in^2)$  worth of data. By contrast, the Steiner method runs in  $O(n^3)$  time, but each action is only  $O(1)$ .

A formula for measuring topological difference is developed by Compton [1]. Essentially, each edge in a commodity's path is examined to determine the amount of difference for each edge-commodity pair between the original and updated topologies. The formula is weighted based on the bandwidth requirements for each commodity and the network size; changes to a high-bandwidth commodity in a small network are more significant than changes to a low-bandwidth commodity in a large network. The amount of change in the route for a given commodity is found by summing the absolute values of the differences in each edge within the route. Absolute values are used to account for both positive and negative changes over a particular network edge.

Here, the binary variable  $y_{ijf}$  represents the inclusion of the  $ijf$  path in the set of edges within a network. The different network configurations,  $\omega$ , are measured at different times,  $t_1$  and  $t_2$ .



$$\Delta(\omega_{t_1}, \omega_{t_2}) = \frac{\sum_{k=1}^K \left[ r^k \sum_{i,j \in N} \sum_{f=1}^F |x_{ijf}^k(t_2) - x_{ijf}^k(t_1)| \right]}{\left[ \sum_{k=1}^K d^k \right] \left[ \sum_{i,j \in N} \sum_{f=1}^F \frac{y_{ijf}(t_2) + y_{ijf}(t_1)}{2} \right]} \quad (3.1)$$

Security of the network is difficult to quantifiably measure. However, the goal of the network reconfiguration is to move the paths over which data flows, thus preventing the ease at which an attacker may gain access. To that end, security is measured through the difference in the network topologies. Over a series of generations of a given configuration, the amount of time any given edge is active is measured. Generational topological differences are also compared using (3.1). It is essential that each generation's topological difference is compared not only to the immediately previous generation, but to a set of previous generations. If the original and third network topologies are similar to each other, and it is known that a network shifts topologies every hour, an attacker simply has to wait three hours before re-attempting the attack.

Network node connectivity is measured to provide a framework for evaluating topological differences. Configurations with high network connectivity have many options for routing network traffic through the nodes and may display high topographical differences. Similarly, a configuration with low connectivity has fewer routing options and may display lower topographical differences. Basaras, Katsaros, and Tassiulas describe the  $\mu - PCI$  metric for calculating connectivity in [20].

The  $\mu - PCI$  metric is found by first finding the node degree values. From there, each node is compared to other nodes that are directly connected to it. The  $\mu - PCI$  value of a node is equal to the number of directly connected nodes with at least that amount of degrees. For example, a node with a degree of 5 and neighbors with degrees of 3, 1, 4, 2, and 5 would be assigned the  $\mu - PCI$  value of 3 because there are 3 directly connected nodes with degrees of at least 3. In this way, the metric looks at both centralness and vitalness of a node; a bottleneck node that connects two highly-connected nodes receives a

high  $\mu - PCI$  value while a central hub node connecting lowly-connected nodes receives a lower value.

By comparison, network cost is remarkably easy to measure. The cost is calculated using (3.2) and displayed as output for the network administrator's evaluation.

$$\omega_{cost} = \left[ \sum_{k=1}^k \left[ d^k \sum_{i,j \in N} \sum_{f=1}^F x_{ijf}^k e_{oc}^{ijf,k} \right] + \sum_{i,j \in N} \sum_{f=1}^F e_{cc}^{ijf,k} \right] \quad (3.2)$$

Essentially, the costs of the network traffic demands are weighed with the number of active edges in a given configuration. A configuration with high demands and a high amount of connectivity will cost more than a network with low demands and low connectivity. For comparison, the total possible cost with complete connectivity is provided.

### 3.4 Summary

The methodology and evaluation approach is defined in this chapter. The AES and Steiner methods are detailed along with all required parameters. Of those parameters, only network size and network interfaces are varied. These two factors are sufficient to gather the metrics necessary for determining the effectiveness of the reconfiguration process. These metrics include CPU time, generational topographical differences, and network costs. Results and analysis of the factors and metrics are presented in the next chapter.

## 4 Results and Analysis

This chapter details the results of the experiments discussed in Chapter 3. Results include run times for each phase of the algorithm, topographical differences, and network cost. The AES and Steiner methods are compared to establish which method performs with better results. Additionally, the two methods are both compared to previous research.

### 4.1 AES Results

*4.1.1 Execution Time.* As predicted by the algorithm run time,  $O((i^2n^3) + (\epsilon^{-2}e(e + n)))$ , the speed of the AES method is dependent on both the number of nodes and the number of interfaces. The dependency on number of interfaces is most acutely seen with the 1000 node test network. The method was able to execute on a network with 1000 nodes and 1 interface, but failed to execute on a 1000 node, 2 interface network within an acceptable time frame. For this reason, the AES approach only includes the 1000 node case with one interface.

The 1000 node, 2 interface case performed a single re-configuration; the execution time was 12.5 days. As discussed during the goals of this research, a configuration that takes days to generate is obsolete upon arrival. The 1000 node, 2 interface case fails to execute in an acceptable amount of time. No analysis of other factors is required is performed on this case. The remainder of the results for the AES method omits the 1000 node case where the number of interfaces is greater than one.

Figure 4.1 shows the total average run time for all 10 generations across the number of performed experiments. Table 4.1 contains the standard deviation of the execution times. Following the 1000 node, 2 interface case, the most significant spike in execution time is between networks with one and two interfaces. This is due to the additional computation time required to handle multiple interfaces; this is the point where the matrices containing network information become three-dimensional. The added

complexity of shifting interfaces as well as rows and columns corresponds to the most drastic increase in total execution time. After the ShiftInterface step is introduced, the additional time to compute 3, 4, and 5 interfaces is exponential, as predicted by the  $O()$  characterization.

Results shown as 0 seconds executed in sub-millisecond time.

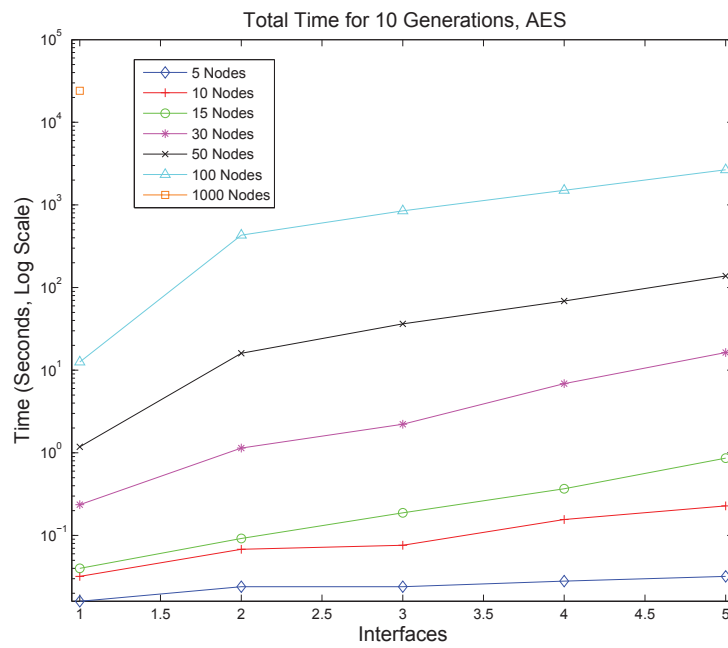


Figure 4.1: AES total times shows speed improvement over previous methods

Figure 4.2 shows the total average time per configuration, across all 50 configurations. Each experiment is performed 5 times, with 10 generations each experiment, giving a total of 50 configurations. Again, it is shown the most drastic increase in execution time for each generation is between one and two interfaces, no matter how many nodes are in the network.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.0089	0.0089	0.0089	0.0110	0.0110
10 Nodes	0.0110	0.0179	0.0167	0.0167	0.0363
15 Nodes	0	0.0110	0.0228	0.0110	0.1774
30 Nodes	0.0607	0.2338	0.1479	0.8065	1.4231
50 Nodes	0.3338	2.3441	6.7026	6.1676	24.0309
100 Nodes	2.5602	23.7758	7.7009	55.6639	117.5725
1000 Nodes	1563.014	NA	NA	NA	NA

Table 4.1: Standard Deviation for Total Time for 10 Generations, AES, in units of network flow

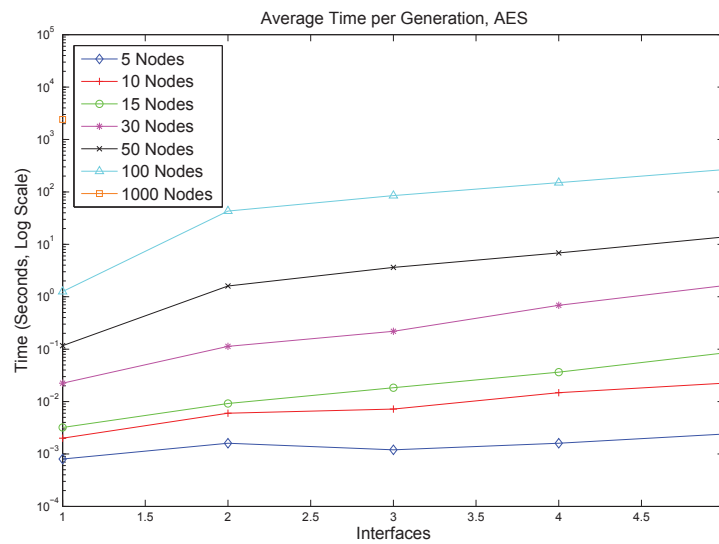


Figure 4.2: AES generational times do not have large deviations from the mean

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.00168654	0.00386436	0.00193218	0.00337309	0.00429987
10 Nodes	0.00210818	0.00432049	0.00590291	0.00269979	0.00337309
15 Nodes	0.00559364	0.00597773	0.00571936	0.0078768	0.01736407
30 Nodes	0.00469515	0.01655160	0.0460260	0.05475602	0.12518856
50 Nodes	0.03198888	0.18894090	0.27712973	0.36084862	0.46111767
100 Nodes	0.13874100	1.49313718	1.57110967	2.77089367	14.7630
1000 Nodes	48.34405	NA	NA	NA	NA

Table 4.2: Standard Deviation for Average Time per Generation, AES, in seconds

Figures 4.3 and 4.4 show the average time per generation for both the edge re-configuration and traffic routing phases of the algorithm, both edge re-configuration and traffic routing. In figure 4.3, values shown as  $10^{-10}$  represent a sub-millisecond time. The figures show that the two phases represent approximately half of the total run time for a generation when the number of interfaces is low, but the time to route the network traffic exponentially increases as the number of interfaces increases. This is due to the routing phase finding paths through the network for all of the traffic. The AES re-configuration only guarantees that the network is whole; it does not establish paths between the source and destination nodes. The routing phase finds all possible paths before routing the traffic; this enumeration and selection of routes takes time and slows the total speed of the algorithm exponentially.

*4.1.2 Network Costs.* Network cost is calculated via Equation 3.2 and shown in Figure 4.5. Both the dynamic routing costs and the static edge costs are considered for the network. The total costs for each network are high, even for small networks. This is mainly due to high traffic demands; a streaming video service consumes greater power and resources than an e-mail service. If these high-demanding traffic sources are eliminated by

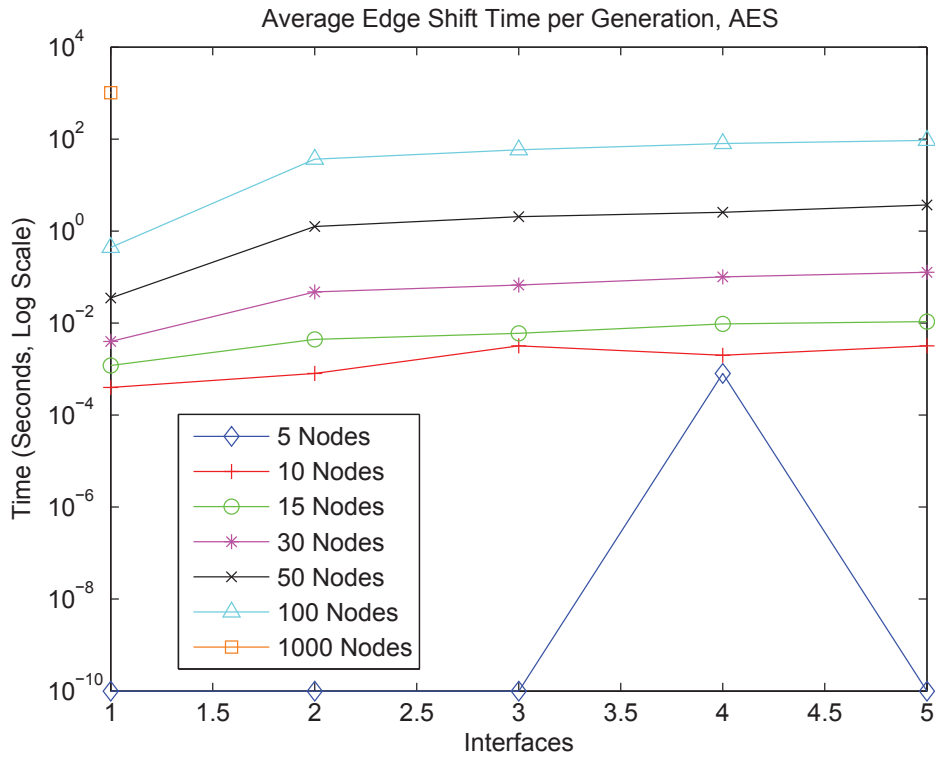


Figure 4.3: AES edge re-configuration occurs with no growth across interfaces

business policy, as is the case with many firms, including the DoD, these costs would drop. The figure shows that traffic demands are more significant than the number of network nodes and interfaces; networks with lower nodes and interfaces are shown to have higher costs than larger networks. This is most drastic in the case of 30 nodes, where the costs are higher than both the 50 and 100 node cases when the three networks have one interface. Examination of the network traffic demands show that a higher overall demand of the network in the 30 node, 3 interface case than in the 50 and 100 node 3 interface cases.

Ideally, the traffic costs of a particular configuration should not exceed the total cost as generated by the adjacency matrix and the list of traffic requirements. However,

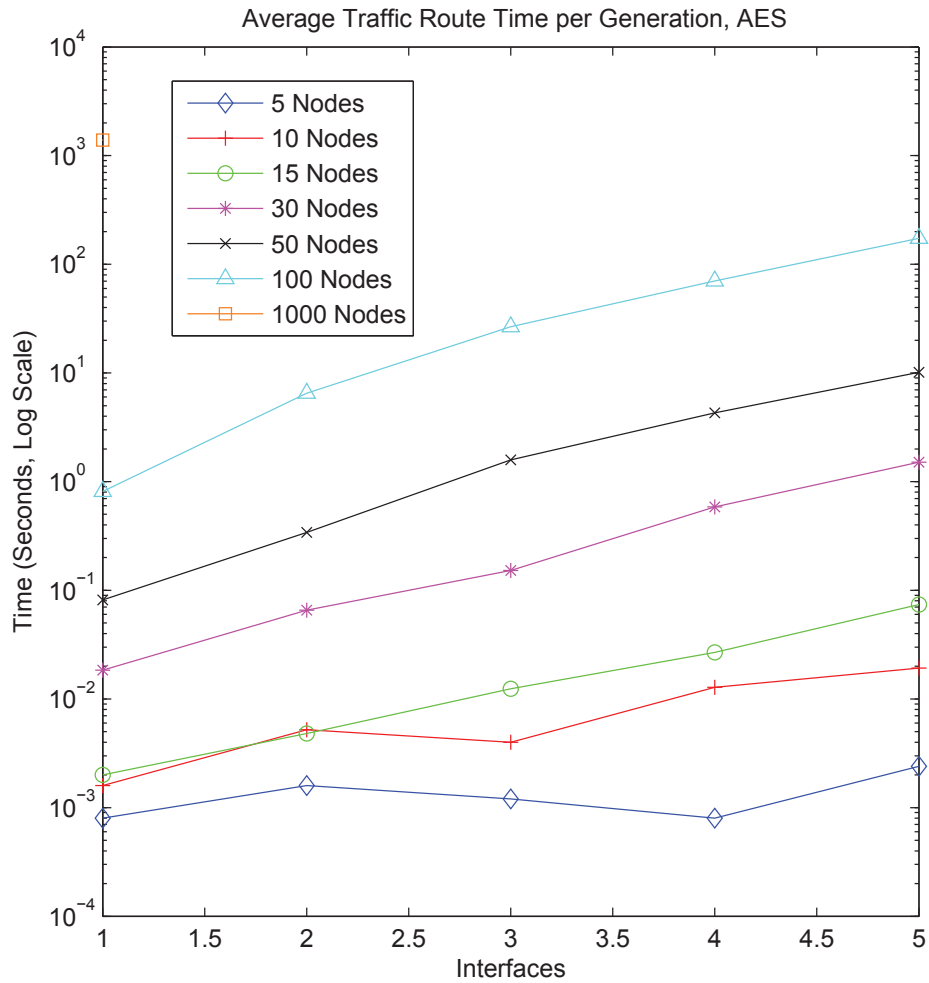


Figure 4.4: AES routing time dwarfs edge re-configuration time exponentially

because the routing paths are not guaranteed to have the shortest path for a particular pair of nodes, the cost of a configuration is frequently higher than the expected cost. The total possible cost is calculated via:



	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0	0	0	0.001686548	0
10 Nodes	0.00126491	0.00168654	0.00413118	0.00388730	0.00413118
15 Nodes	0.00193218	0.00397771	0.00432049	0.00505964	0.00379473
30 Nodes	0.00326598	0.0035023	0.0049170	0.00976160	0.00888444
50 Nodes	0.0070047	0.14755850	0.18955104	0.1327621	0.33203052
100 Nodes	0.01053248	0.99405378	1.4596769	1.84599337	2.04530482
1000 Nodes	6.8710018	NA	NA	NA	NA

Table 4.3: Standard Deviation for Edge Shift Time per Generation, AES, in seconds

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.00168654	0.00386436	0.00193218	0.00505964	0.00429987
10 Nodes	0.00337309	0.00600704	0.01003410	0.00658709	0.00750427
15 Nodes	0.00752583	0.00995545	0.01003985	0.01293647	0.02115880
30 Nodes	0.00796113	0.02005398	0.0509431	0.06451762	0.13407301
50 Nodes	0.03899364	0.33649940	0.46668078	0.49361073	0.79314819
100 Nodes	0.14927349	2.48719097	3.03078659	4.6168870	16.8083448
1000 Nodes	55.21505	NA	NA	NA	NA

Table 4.4: Standard Deviation for Traffic Route Time per Generation, AES, in seconds

$$\omega_{possible\_cost} = \left[ \sum_{k=1}^k d^k + \sum_{i,j \in N} \sum_{f=1}^F e_{cc}^{ijf} \right] \quad (4.1)$$

This equation is run against the adjacency matrix to find all possible edge costs, to account for usable edges that are not included in a given configuration. The ratio between

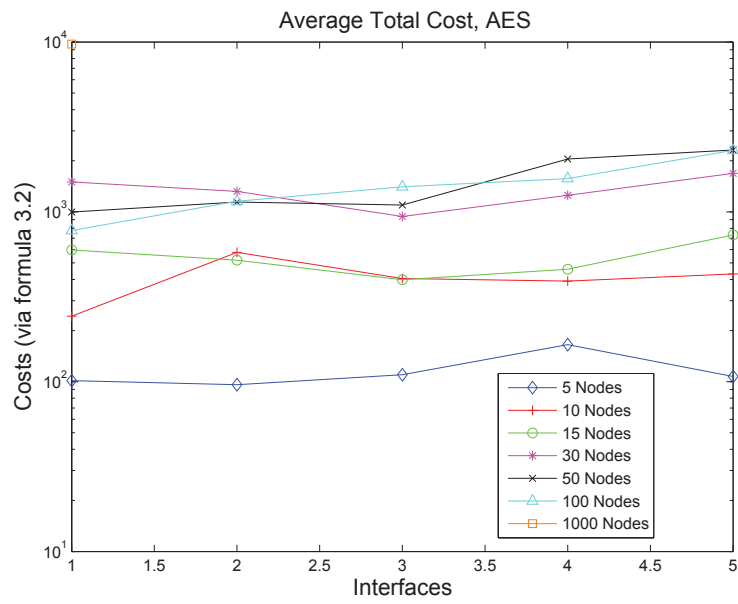


Figure 4.5: AES costs vary with traffic demands

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.0084	0.0054	0.0018	0.0056	0.0060
10 Nodes	0.0138	0.0117	0.0137	0.0086	0.0279
15 Nodes	0.0367	0.0312	0.0276	0.0260	0.0380
30 Nodes	0.1178	0.0889	0.0831	0.0967	0.1216
50 Nodes	0.1880	0.1656	0.1488	0.2657	0.2999
100 Nodes	0.2723	0.3610	0.4339	0.4831	0.6948
1000 Nodes	4.3306	NA	NA	NA	NA

Table 4.5: Standard Deviation for Average Total Cost, AES, by formula 3.2

the average calculated configuration cost and the total possible cost is given in Table 4.6. The table shows that as the network size increases, either by increasing the number of nodes or the number of interfaces, the relative cost of the network at each configuration decreases as compared to the possible cost of the network.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	18.778	4.9462	6.1031	6.8300	6.6291
10 Nodes	6.0404	14.8884	11.4425	7.3501	8.0124
15 Nodes	12.4686	10.1856	4.8587	6.3072	7.9257
30 Nodes	16.7472	9.0119	4.7109	5.2706	5.3081
50 Nodes	5.3305	4.2630	2.7469	3.3010	2.7337
100 Nodes	1.1037	1.0770	.8624	.6313	.6746
1000 Nodes	.3916	NA	NA	NA	NA

Table 4.6: Ratio of Configuration Cost vs. Total Possible Cost, AES, via formula 3.2

*4.1.3 Network Differences.* Generational topographical differences are calculated against all 50 configurations of a given combination of nodes and interfaces via Equation 3.1. Figure 4.6 shows the results of the equation. Networks with 3 interfaces are shown to have the most consistent differences, compared to the other possible network configurations. Figure 4.6 also shows that the cases of 5 nodes with 1 and 5 interfaces along with the 10 node and 15 node 1 interface cases to show no differences between each generation.

Examinations of the final network configurations of each generation for those four cases show that the network configuration does not deviate. The calculated configuration

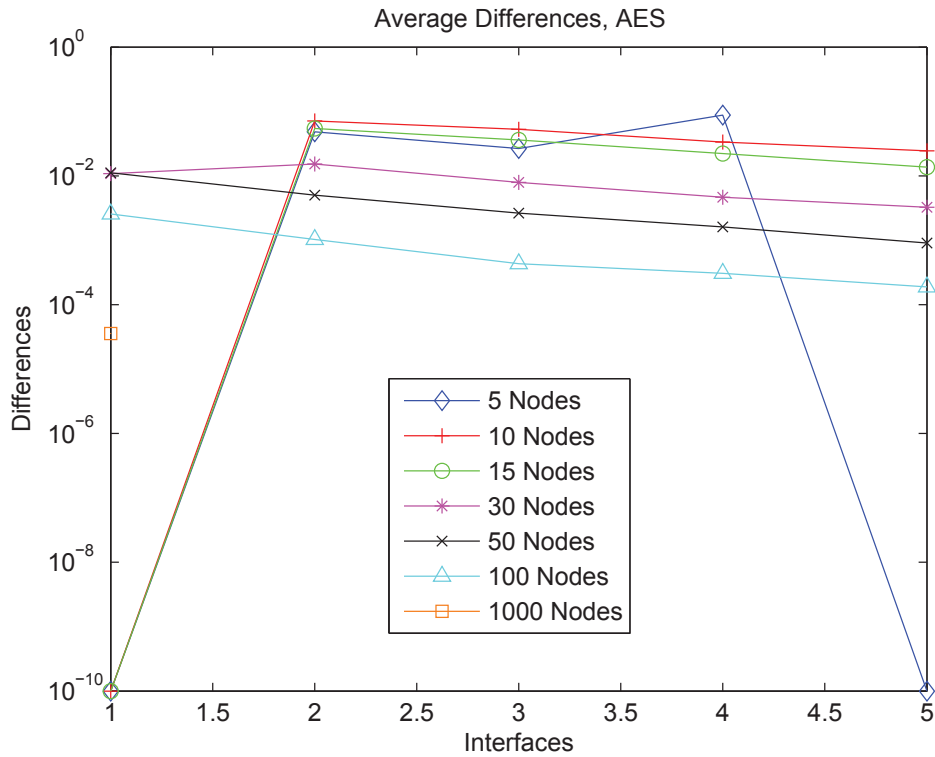


Figure 4.6: AES differences vary unpredictably over different configurations

is a subset of the total available edges as given by the adjacency matrix for each case, but the configurations do not change between generations as expected.

This is most likely due to the verification steps. Examinations of the network configurations of these cases before the verification steps do show differences in the overall configuration. However, the configurations were missing nodes or contained multiple sub-networks instead of a single network. The verification steps address these issues in a straightforward manner, and the resulting final configuration is the same in each case, no matter the starting configuration before the verification. Similarly, the routing phase selects paths in a programmatic way; given the same input of possible edges, the same paths are repeatedly selected because the algorithm does not weight paths.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0	0.0663	0.0566	0.0830	0
10 Nodes	0	0.0335	0.0249	0.0112	0.0078
15 Nodes	0	0.0215	0.0172	0.0080	0.0039
30 Nodes	0.0151	0.0051	0.0016	0.0007	0.0005
50 Nodes	0.0030	0.0018	0.0004	0.0002	0.0003
100 Nodes	0.0018	0.0001	0.0001	0.0000	0.0000
1000 Nodes	0.0000	NA	NA	NA	NA

Table 4.7: Standard Deviation for Average Total Topological Differences, AES, in units of network flow

Differences in topology can also be described by the percentage of edges that are active across all generations. That is, the more edges that are less active over all time, the greater the differences. Table 4.8 shows the number of edges that are active across every generation.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	8	10	14	10	14
10 Nodes	20	8	4	18	34
15 Nodes	28	4	8	14	24
30 Nodes	56	44	58	42	18
50 Nodes	78	18	2	6	350
100 Nodes	28	72	484	20	10
1000 Nodes	148	NA	NA	NA	NA

Table 4.8: Active Edges in All Generations, AES

In the cases of 10 and 15 nodes with 1 interface, the table shows that all generations have every possible edge active. This situation does not allow for further configuration changes; once a generation adds all possible edges, no edges are removed via the verifications steps. The shifting phase still executes but no visible changes are made if all the possible edges are already active.

In the other cases, the differences of the network are shown to decrease as the number of interfaces increases. Logically, the differences should increase, given additional dimensions to affect change. The connectiveness of a node affects the differences. If many nodes are minimally connected, changing a given node's connections may not have a great effect on the network as a whole. Table 4.9 shows the average  $\mu - PCI$  values. The  $\mu - PCI$  value shows that as the number of interfaces increase, the amount of overall connectiveness of a node actually decreases. Here, a node that connects to another node over three different interfaces is less connective than if it connects to three different nodes over the same interfaces. The average node connectiveness corresponds with the decrease in topographic differences as interfaces increase; the fewer connections a node has, the harder it is to make significant changes.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	1	1	1	1	1
10 Nodes	3	1	1	1	1
15 Nodes	1	1	1	1	1
30 Nodes	7	3	1	1	1
50 Nodes	6	1	1	1	1
100 Nodes	1	1	1	1	1
1000 Nodes	8	NA	NA	NA	NA

Table 4.9: Node Connectiveness, AES

## 4.2 Steiner Results

4.2.1 *Execution Time.* Unlike the AES method, the Steiner method is not dependent on the number of interfaces and runs in  $O((n^3) + (\epsilon^{-2}e(e + n)))$  time. For that reason the Steiner method executed on the 1000 node case with all tested numbers of interfaces, providing a more complete analysis of the method. Figure 4.7 shows the total time for 10 generations utilizing the Steiner method.

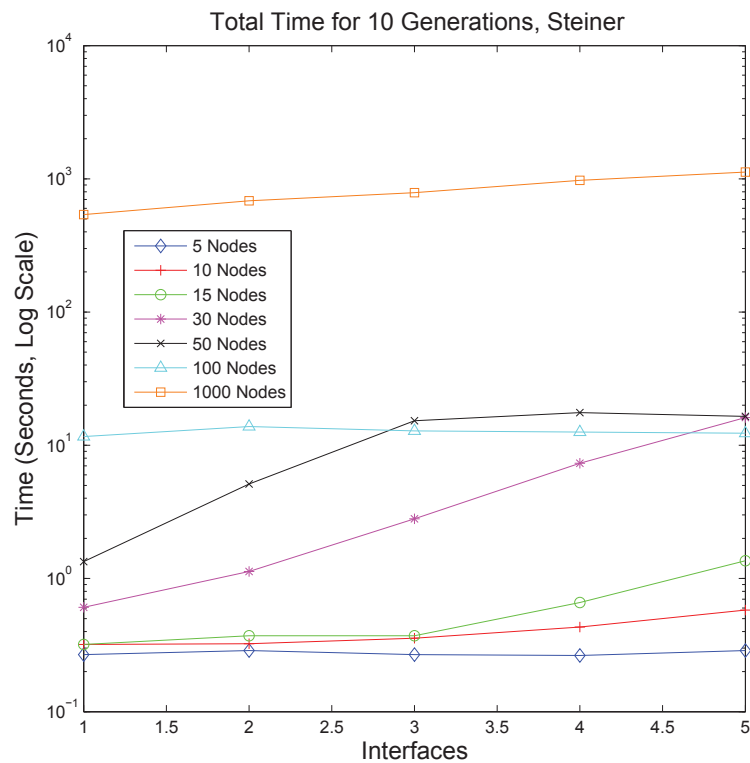


Figure 4.7: Steiner times show improvement over AES times

Like the AES method, the 1000 node cases require an order of magnitude of additional time to execute to completion. However, there is less of a pronounced difference in networks with multiple interfaces as compared to networks with one interface. Notably,

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.02683281	0.038987177	0.036331804	0.03286335	0.02683281
10 Nodes	0.02449489	0.03577708	0.068410526	0.0558569	0.14071247
15 Nodes	0.04	0.041472883	0.05761944	0.07874007	0.1104536
30 Nodes	0.09230384	0.25242820	0.272249885	0.24190907	2.01148701
50 Nodes	0.17663521	0.66819158	1.320348439	0.614068	0.7501199
100 Nodes	1.00365332	3.08821631	3.266355768	2.09654954	3.05486497
1000 Nodes	42.7001171	22.57653649	21.77154106	34.1203751	24.1909073

Table 4.10: Standard Deviation for Total Time for 10 Generations, Steiner, in seconds

the time required for the 100 node cases remained similar across all tested interfaces while the time required for the 50 and 30 node cases increased exponentially. This may lead to the conclusion that the random generation of the starting configuration of this method affects the running time of each generation. If the starting hash sequence does not produce a connected network, additional time is required to connect the set of node pairs.

However, if the hash sequence produces a connected network, the overall time requirement is greatly reduced. This assumption is validated but the running times shown in Figure 4.7, but it is later shown that this assumption is not the case.

Figure 4.8 shows the total average time per configuration, across all 50 configurations. This figure mirrors Figure 4.7. This means that the execution time of each generation is approximately equal; meaning that no one generation causes a great effect to the total time.

Figures 4.9 and 4.10 show the average time per generation for the edge re-configuration and traffic routing phases of the algorithm, respectively. Figure 4.9 shows that the network re-configuration time of each generation is minimal in all but the 1000 node cases. This shows that the edge selection phase of the algorithm has little effect on



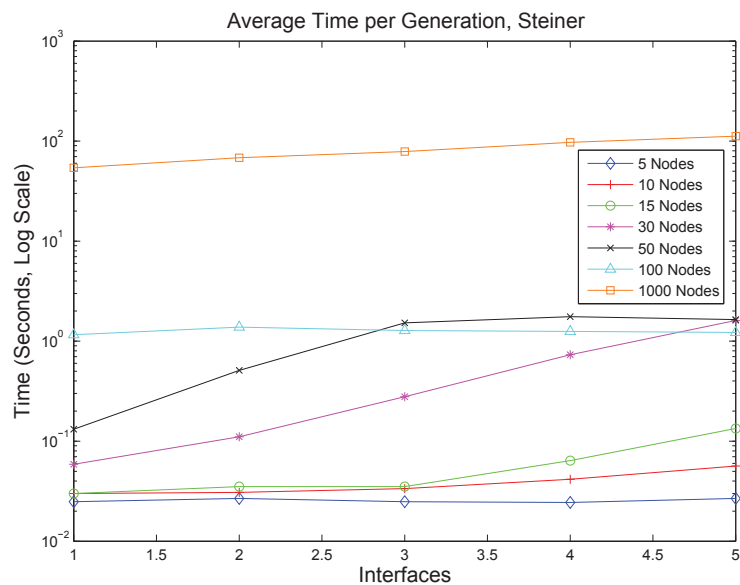


Figure 4.8: Steiner interfaces contribute to overall generation time

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.00367574	0.00567254	0.00559364	0.00478887	0.00597773
10 Nodes	0.00339934	0.00597773	0.00631048	0.00337309	0.0156005
15 Nodes	0.00541602	0.00647731	0.00413118	0.00326598	0.01897366
30 Nodes	0.00731512	0.01347260	0.03216899	0.06871001	0.17851187
50 Nodes	0.02868681	0.02614149	0.16677063	0.14956470	0.19563639
100 Nodes	5.29456324	5.13688621	5.04285633	7.17275400	7.71494653
1000 Nodes	5.29456324	5.13688621	5.04285633	7.17275400	7.71494653

Table 4.11: Standard Deviation for Average Time per Generation, Steiner, in seconds

the overall running time of the generation of the new configuration. Figure 4.10 reflects Figures 4.8 and 4.7. This shows that the total time is more affected by the routing time

than the edge selection time. As in figure 4.7, the 100 node case performs faster than the 50 node case at 3 interfaces, and faster than the 30 node case at 5 interfaces. This means the previous conclusion of the affect of the starting configuration on the run time is faulty; this difference is better explained by the complexity of the network and the number of possible paths for each commodity. The total computational time is increased by the number of possible paths because more flexibility is available. Greater flexibility requires more iterations for Fleisher’s approximation algorithm to produce a result within the desired error limit. This shows the previous conclusion that the random starting configuration affecting the execution time is incorrect.

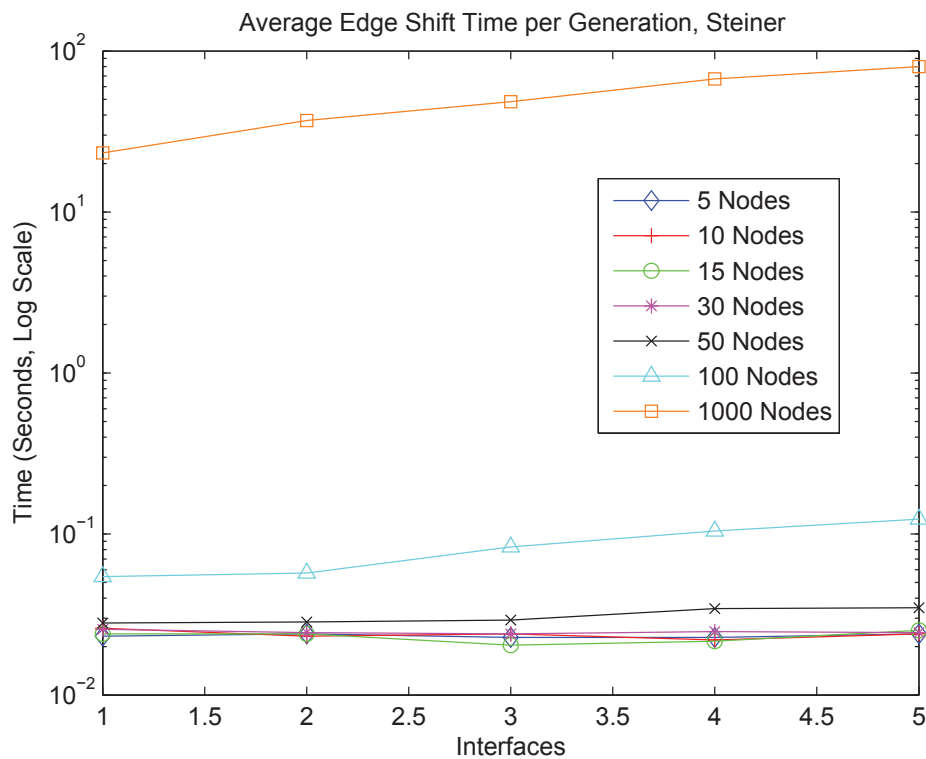


Figure 4.9: Steiner edge re-configuration times shows great improvement over AES method

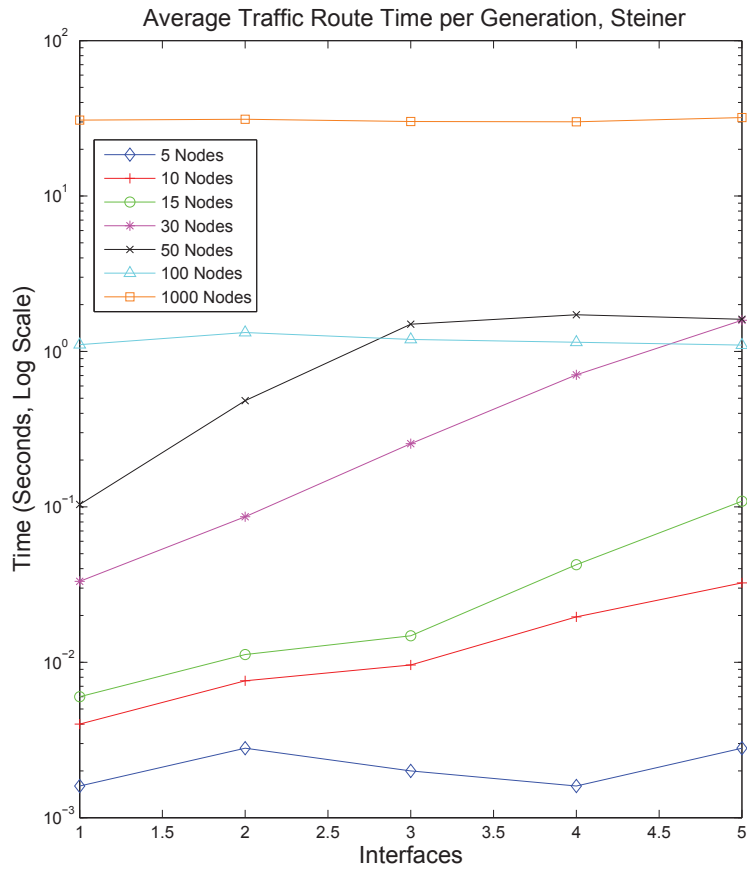


Figure 4.10: Steiner traffic routing contributes most to overall generational time

4.2.2 *Network Costs.* Similar to the AES method, the total cost of the networks generated by the Steiner method are high, even for small networks. The reasoning is the same; traffic sources with higher requirements cost more to operate than traffic sources with low demands. The costs are averaged over all iterations of experiments; that is, the total cost shown in Figure 4.11, is the average cost over all 5 experiments for a given node/interface combination.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.0049170	0.00565685	0.004237	0.00463800	0.00498887
10 Nodes	0.00432049	0.0049170	0.00498887	0.00282842	0.00326598
15 Nodes	0.00266666	0.00461880	0.00227058	0.00279682	0.00500666
30 Nodes	0.00386436	0.00478887	0.00461880	0.00526624	0.00478887
50 Nodes	0.00461880	0.00579655	0.00379473	0.00337309	0.00500666
100 Nodes	0.01018931	0.01320605	0.01930342	0.03224627	0.04689278
1000 Nodes	1.09969692	2.66999791	4.05566818	4.69231049	5.47089267

Table 4.12: Standard Deviation for Average Edge Shift Time per Generation, Steiner, in seconds

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.00859283	0.011329	0.009831047	0.00942688	0.01096661
10 Nodes	0.0077198	0.01089482	0.01129936	0.00620152	0.01886655
15 Nodes	0.00808269	0.01109611	0.00640176	0.0060628	0.023980328
30 Nodes	0.01117949	0.01826148	0.036787	0.07397626	0.18330074
50 Nodes	0.03330561	0.03193804	0.17056536	0.15293779	0.2006430
100 Nodes	0.11784225	0.15853878	0.21614289	0.15474253	0.23125764
1000 Nodes	6.394260177	7.80688413	9.09852452	11.865064	13.1858392

Table 4.13: Standard Deviation for Average Traffic Route Time per Generation, Steiner, in seconds

The figure shows minimal differences between the costs over the number of interfaces in a given network. The greatest differences in the cost come from adding additional nodes to the network. The anomalies are the 50 and 100 node cases, which are

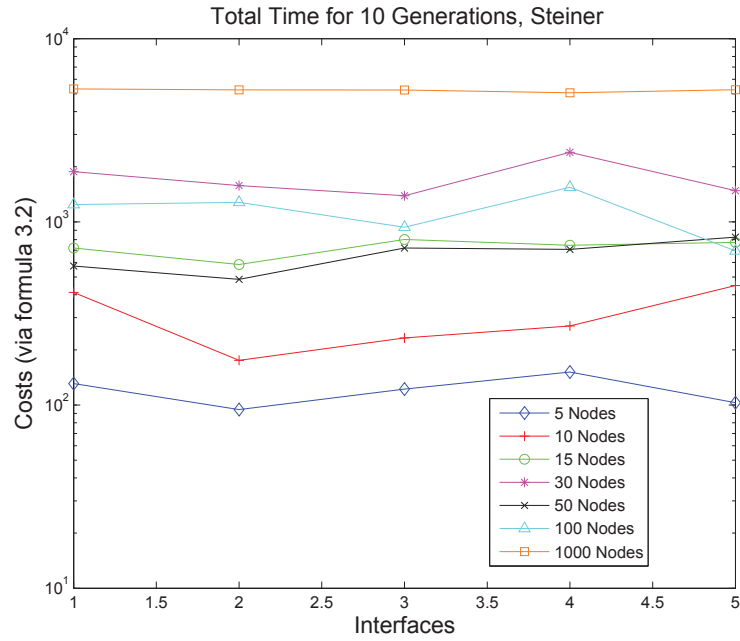


Figure 4.11: Steiner costs are similar to AES costs

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	68.06964	183.2627	196.1926	1072.750	1131.2941
10 Nodes	3997.59901	23682.4348	46.735432	135.1845	357.3613
15 Nodes	800.30723	1027.8742	4108.57842	23411.093	65.0724
30 Nodes	82.65524	167.2656	436.03921	1546.4331	2995.3226
50 Nodes	23379.1335	59.7165	139.23153	371.9814	414.4486
100 Nodes	1553.0178	4741.2014	22581.06839	46.84479	155.3167
1000 Nodes	138.5919	1025.84670	1706.11418	2447.7096	23459.6333

Table 4.14: Standard Deviation for Average Total Cost, Steiner, via formula 3.2

shown to be less costly than the 30 node cases. While the 50 and 100 node cases have more commodities to route, examination of the generated commodity demands show that many commodities for the 50 and 100 node cases are low demand. Further examination of the 30 node cases shows high-demand commodities.

The same cost ratio analysis as in the AES analysis is performed. Table 4.15 shows similar results as the AES method: as the number of interfaces or nodes increases, the proportion of actual cost to theoretical cost decreases.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	8.8456	6.3846	5.8761	7.0731	6.2837
10 Nodes	11.5772	4.5893	5.4021	5.8269	7.1074
15 Nodes	17.1274	9.4009	11.8164	8.7333	7.4068
30 Nodes	19.9478	11.5733	7.5770	10.1409	4.5425
50 Nodes	3.21015	1.8424	1.7222	1.1705	.9451
100 Nodes	1.7287	1.1961	.5553	.6291	.1976
1000 Nodes	.2142	.0812	.0812	.0350	.0257

Table 4.15: Ratio of Configuration Cost vs. Total Possible Cost, Steiner, via formula 3.2

*4.2.3 Network Differences.* Generational topographical differences are calculated against all 50 configurations of a given combination of nodes and interfaces via Equation 3.1. Figure 4.12 shows the resulting network differences of the equation. Smaller networks are shown to have less overall differences than larger networks, with the exception of the 50 node cases after 3 interfaces.

These results follow logical reasoning: Smaller networks are more sensitive to change, and show the biggest differences for the smallest amount of overall changes. As the number of network nodes and interfaces increase, the amount of change at the network level decreases. For example, take the 100 node case and examine only the edge

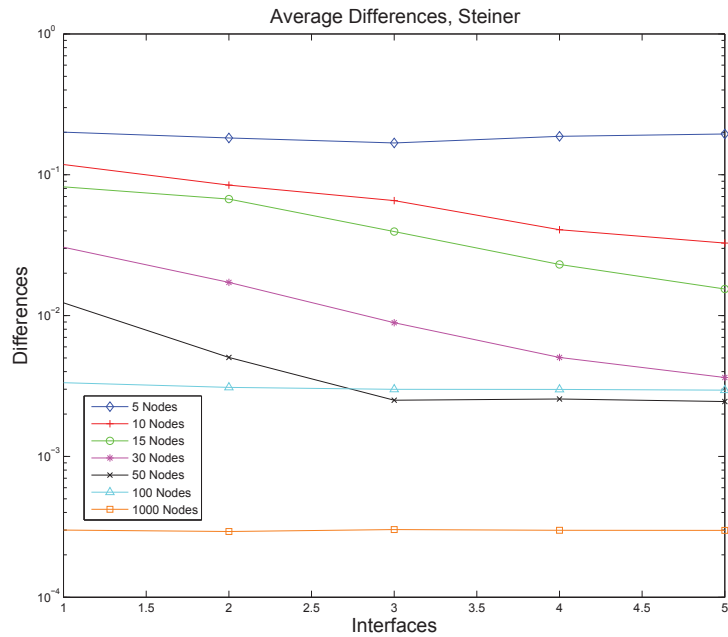


Figure 4.12: Steiner differences exponentially decrease as interfaces linearly increase

re-configuration phase. Modifying 10 edges over one interface results in a 10% change. However, modifying 15 edges over two interfaces only results in a 7.5% change, despite the fact that more changes occurred. Consistent change is shown across interfaces in the 5, 100, and 1000 node cases while in the other cases greater differences are shown with a fewer number of interfaces.

The average number of active edges and the connectivity of the network correspond with the variations in the topographical differences. Table 4.17 shows the percentage of active edges in a given network with respect to the adjacency matrix. Table 4.18 shows the connectiveness of the nodes.

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0.1148	0.0741	0.0490	0.0678	0.0637
10 Nodes	0.0362	0.0275	0.0162	0.0088	0.0058
15 Nodes	0.0218	0.0157	0.0068	0.0032	0.0015
30 Nodes	0.0048	0.0017	0.0010	0.0005	0.0005
50 Nodes	0.0023	0.0007	0.0002	0.0003	0.0003
100 Nodes	0.0003	0.0004	0.0003	0.0003	0.0003
1000 Nodes	0.0000	0.0000	0.0000	0.0000	0.0000

Table 4.16: Standard Deviation for Topological Differences, Steiner, in units of network flow

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	0	0	0	0	0
10 Nodes	0	0	0	0	0
15 Nodes	0	0	0	0	0
30 Nodes	0	0	0	0	0
50 Nodes	0	0	0	0	0
100 Nodes	0	0	0	0	0
1000 Nodes	0	0	0	0	0

Table 4.17: Number of Active Edges in All Generations, Steiner

The sparse network reflects the lack of differences shown in Figure 4.12. A fully connected network is required; if a node has few neighbors, there are few options to modify the connection.



	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
5 Nodes	2	1	1	1	1
10 Nodes	3	1	1	1	1
15 Nodes	2	2	1	1	1
30 Nodes	4	2	1	1	1
50 Nodes	3	1	1	1	1
100 Nodes	4	1	1	1	1
1000 Nodes	60	30	7	9	3

Table 4.18: Node Connectiveness, Steiner

### 4.3 Comparisons

The AES and Steiner methods presented in this research are evaluated under the same metrics. Comparisons between the methods are accomplished via a t-test.

The Steiner method provides greater topological differences in all but the 50 node 2 interface case at a 95% confidence level. Expanding the comparison to the 90% confidence level does not prove the Steiner method provides greater topographical differences in those cases.

Comparing the costs of the generated networks shows the methods are more equal. The AES method produces cheaper networks for the 5 node cases with all interfaces; the 10 node 5 interface case; the 15 node 2 and 5 interface cases; the 30 node 1, 2, and 5 interface cases; and the 100 node 2 and 4 interface cases at the 95% confidence level. The Steiner method; however, provides cheaper networks for randomly generated commodities at the 5 node 1 interface case and the 30 node 1 interface case at the 90% confidence level.

Each generation of networks is sparse. This is almost by design; examination of the starting adjacency matrices shows the set of possible network edges are also sparse

networks. To test the methods of creating network differences on a complete network, additional experiments are generated with the 50 node case. The adjacency matrixes are modified to include every edge; each generation still contains a subset these possible edges.

Figure 4.13 shows the amount of network differences for each method, on both a complete and sparse set of possible network edges for the 50 node cases. Each line shown represents the amount of differences for that case as a result of the topological difference formula given by formula 3.1. The figure shows that a fully connected network generates fewer differences than the sparse networks. Again, this is due to the availability of additional edges. When more edges are available, more changes are required to show a significant difference at the network level.

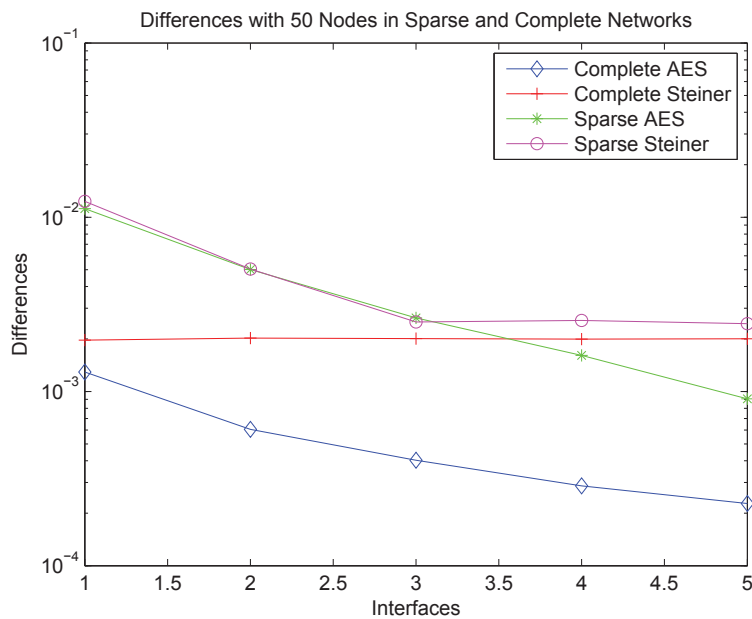


Figure 4.13: Complete possible networks only assist differences if generated configuration is not sparse

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
AES, $10^{-4}$	0.706	0.3495	0.2252	0.1387	0.1331
Steiner, $10^{-3}$	0.1921	0.1911	0.2311	0.1574	0.1229

Table 4.19: Standard Deviation, Differences in units of network flow

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
AES	1	1	1	1	1
Steiner	6	1	1	1	1

Table 4.20:  $\mu - PCI$  of Complete Graphs

	1 Interface	2 Interfaces	3 Interfaces	4 Interfaces	5 Interfaces
AES	10	8	0	40	110
Steiner	0	0	0	0	0

Table 4.21: Connectiveness of Complete Graphs

Tables 4.20 and 4.21 show that despite the relaxed adjacency matrix, the generated networks remain sparse. Comparing the table information to the Figure 4.13, the sparsity of the generated networks explains the lack of differences between generations. However, examining the number of active edges present in all generations decreased from the networks generated from the sparse adjacency matrixes. This means that while the calculated differences are small, the original goal is still partially achieved. The number of edges active 100% of the time over a group of generations is perhaps a better measurement of network differences; however, further experiments are required.

The Steiner method continues to perform better than the AES method, however. At a 95% confidence level, the Steiner method generated networks with greater differences than the AES method across all interfaces.

Both methods are compared against Compton's work in regards to network differences for the 5, 10, and 15 node cases at the 1, 2, 3, and 4 interface level as well as the 30 node case at the 1 and 2 interface level. In both methods, no significant differences are observed in the differences between generated network topologies at the 90% confidence level using a t-test [1].

#### **4.4 Summary**

The results presented here detail the effectiveness of the AES and Steiner methods with Fleisher's routing method. The results show that topographic differences are dependent on the network infrastructure; the more options a node has to connect to the network, the greater the amount of differences can be made. Additionally, the greater the amount of redundancy, the easier it becomes for network administrators to make changes without network downtime.

The overall speed of the Steiner method allows for quick changes in the face of an active attack. This is important because attacks such as distributed denial of service attacks ramp up quickly. The ability to modify the network configuration as well as block the appropriate incoming address may serve to continue to allow legitimate traffic while the attack is ongoing. Not only would the attackers have to modify the incoming addresses, they would have to take the time to modify the attack vector to account for the shifting topology.

## 5 Conclusions and Future Work

Demand for effective network defense capabilities continues to increase as cyber attacks occur more and more frequently and gain more and more prominence in the media. Many defense strategies exist, but more are needed to meet the challenge of increasingly sophisticated attacks. This research addresses dynamic network topologies as a means of network defense in response to both ongoing cyber attacks or the threat of potential cyber attacks. The prototype application developed dynamically shifts the network topology and re-routes network traffic accordingly. This network shift, in theory, expends the resources of an attacker, as the network requires continually re-scanning to determine appropriate attack vectors.

The goal of this research is to provide another layer of defense for use by network administrators. An algorithm was successfully developed using two different methodologies. The first methodology utilized the Advanced Encryption Standard as a means to encrypt a network topology. The second methodology uses graph theory and Steiner trees to generate a network configuration from the basic network information. Both methods use the Fleisher routing algorithm to ensure the network traffic is routed properly through the new network configuration.

This research improved on previous research by addressing a greater number of nodes and network commodities. The results for networks with nodes 50, 100, and 1000 have not been addressed previously, and the number of commodities was previously limited to 3 commodities per network. The sample networks used in this research are closer in scale and scope to operational networks currently in use.

However, improvements can be made to this research before it is used in operational environments. Additional steps can be added to the algorithm to group nodes together for networks larger than 1000 nodes. This would also assist in modifying an extensive enterprise network such as the DoD's networks. Each base or other installation would

have the ability to modify the local network without affecting the connections between installations. Enterprise administrators can then independently modify those connections. This scales in the opposite way as well – an installation can independently modify the networks of various tenants, then on the structure of the greater installation infrastructure. The scalability allows for isolation of the changes, if required; it is needless to potentially disturb the entire network when only a certain location is targeted.

The algorithm itself can be improved in a number of ways. The execution runtime can be shortened if the algorithm were to execute in a parallel manner, instead of in a serialized way. Currently, both phases and methods of the algorithm execute iteratively, but each step is not dependent on the surrounding steps. The only dependency is that the network edges must be modified before the network routing takes place. Additionally, this algorithm focuses mainly on network infrastructure devices with multiple network interfaces. General workstations only possess one connection to a single infrastructure device. While this case can be addressed via the set of possible edges, separation of the infrastructure devices and workstations may produce better results.

This research also assumed the network edges are bi-directional. In an operational sense, this is not always a safe assumption. Read-only connections are occasionally required for security of other purposes. Additionally, measures to weight the network paths in the route phase of the algorithm can be improved upon. Currently, no preference is given to routes that were not used in previous generations, if the same edges are still present. Additional topographical differences can be achieved with various routing paths. Network traffic demands can also be prioritized; it may be more important to guarantee e-mail and message traffic flow before streaming video, depending on the needs of the network and the mission.

The work presented here also assumes that the network is clean; that is, there is no malware present on the network. Potentially, malware already operating on the network

can automatically adapt to any network changes made by way of sending updated information back to the attacker. Examination of packets after network modifications may serve to identify infected machines. This functionality would jointly serve with an IPS as an enhancement to the IPS network sensors.

Adding security measures to the physical layer of the network adds to the overall security package of a network. Current security practices stop after data encryption and network address filtering. Security at the lowest level of network infrastructure allows for greater control of how the network traffic flows around the network. Greater control over the network means more options are available for defense when the network is attacked.

## Bibliography

- [1] M. Compton, “Improving the quality of service and security of military networks with a network tasking order process,” Ph.D. dissertation, AFIT, 2010.
- [2] “Specification for the advanced encryption standard (aes),” National Institute of Standards and Technology, FIPS PUBS 197, November 2001.
- [3] L. C. W. Wade Trappe, *Introduction to Cryptography with Coding Theory*, 2nd ed. Pearson Prentice Hall, 2006.
- [4] R. R. C. S. T. Cormen, C. Leiserson, *Introduction to Algorithms*, 2nd ed., M. Press, Ed. McGraw-Hill, 2001.
- [5] D. P. Shmoys, David B. Williamson, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
- [6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, P. Janzow, Ed. Prentice Hall, 1993.
- [7] E. Kleinberb, Jon. Tardos, *Algorithm Design*, C. Leyba, Ed. Pearson Education, Inc, 2006.
- [8] N. E. Young, “Randomized rounding without solving the linear program,” *CoRR*, vol. cs.DS/0205036, 2002.
- [9] N. Garg and J. Koenemann, “Faster and simpler algorithms for multicommodity flow and other fractional packing problems,” *SIAM Journal on Computing*, vol. 37, no. 2, pp. 630–652, 2007.
- [10] L. K. Fleischer, “Approximating fractional multicommodity flow independent of the number of commodities,” *SIAM J. Discrete Math*, vol. 13, 2000.
- [11] K. S. Ho and K. W. Cheung, “Generalized survivable network,” *Networking, IEEE/ACM Transactions on*, vol. 15, no. 4, pp. 750–760, 2007.
- [12] Y. Agarwal, “Survivable network design using polyhedral approaches,” in *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, 2011, pp. 1–6.
- [13] A. Narula-Tam and E. Modiano, “Dynamic load balancing in wdm packet networks with and without wavelength constraints,” *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 10, pp. 1972–1979, 2000.
- [14] P. Tran and U. Killat, “Dynamic reconfiguration of logical topology for wdm networks under traffic changes,” in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, 2008, pp. 279–286.



- [15] P. N. Tran, L. Casucci, and A. Timm-Giel, "Optimal mapping of virtual networks considering reactive reconfiguration," in *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, 2012, pp. 35–40.
- [16] M. C. Erwin, "Combining quality of service and topology control in directional hybrid wireless networks," Master's thesis, Air Force Institute of Technology, 2006.
- [17] M. Dzida, M. Zagodzón, M. Pióro, T. Sliwinski, and W. Ogryczak, "Path generation for a class of survivable network design problems," in *Conference on Next Generation Internet Networks*, 2008.
- [18] A. K. Todimala and B. Ramamurthy, "Approximation algorithms for survivable multicommodity flow problems with applications to network design," in *IEEE INFOCOM*, 2006, pp. 1–12.
- [19] V. Daemen, Joan; Rijmen, "Aes proposal: Rijndael," September 1999. [Online]. Available: <http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf#page=1>
- [20] P. Basaras, D. Katsaros, and L. Tassiulas, "Detecting influential spreaders in complex, dynamic networks," *Computer*, vol. 46, no. 4, pp. 24–29, 2013.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.				
1. REPORT DATE (DD-MM-YYYY) 13-06-2013		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Jan 2010 – 13 June 2013
4. TITLE AND SUBTITLE Dynamic Network Topologies			5a. CONTRACT NUMBER	
			5b. GRANT NUMBER	
			5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Lingg, Heather A, Ms.			5d. PROJECT NUMBER 13G192H	
			5e. TASK NUMBER	
			5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENY) 2950 Hobson Way WPAFB OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-13-J-04	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Dr. Robert J. Bonneau 875 N Randolph St, Ste 325, Rm 3112, Arlington, VA 22203 (703) 696-9545 (DSN: 426-9545) Email: <a href="mailto:robert.bonneau@afosr.af.mil">robert.bonneau@afosr.af.mil</a>			10. SPONSOR/MONITOR'S ACRONYM(S) AFOSR/RTC	
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED				
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.				
14. ABSTRACT Demand for effective network defense capabilities continues to increase as cyber attacks occur more and more frequently and gain more and more prominence in the media. Current security practices stop after data encryption and network address filtering. Security at the lowest level of network infrastructure allows for greater control of how the network traffic flows around the network. This research details two methods for extending security practices to the physical layer of a network by modifying the network infrastructure. The first method adapts the Advanced Encryption Standard while the second method uses a Steiner tree. After the network connections are updated, the traffic is re-routed using an approximation algorithm to solve the resulting multicommodity flow problem. The results show that modifying the network connections provides additional security to the information. Additionally, this research extends on previous research by addressing enterprise-size networks; networks between 5 and 1000 nodes with 1 through 5 interfaces are tested. While the final configuration depends greatly on the starting network infrastructure, the speed of the execution time enables administrators to make infrastructure adjustments in response to active cyber attacks.				
15. SUBJECT TERMS Dynamic network topology, cryptology, graph theory				
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT  UU	18. NUMBER OF PAGES  80
a. REPORT	b. ABSTRACT	c. THIS PAGE		
U	U	U	19a. NAME OF RESPONSIBLE PERSON Dr. Kenneth Hopkinson, AFIT/ENG	
			19b. TELEPHONE NUMBER (Include Area Code) (937)255-3636, ext 4579 kenneth.hopkinson@afit.edu	