UNF UNIVERSITY of NORTH FLORIDA

UNF Digital Commons

UNF Graduate Theses and Dissertations

Student Scholarship

2015

A Targeted Denial of Service Attack on Data Caching Networks

Jeffrey B. Gouge
University of North Florida

Suggested Citation

Gouge, Jeffrey B., "A Targeted Denial of Service Attack on Data Caching Networks" (2015). *UNF Graduate Theses and Dissertations*. 575.
https://digitalcommons.unf.edu/etd/575

This Master's Thesis is brought to you for free and open access by the Student Scholarship at UNF Digital Commons. It has been accepted for inclusion in UNF Graduate Theses and Dissertations by an authorized administrator of UNF Digital Commons. For more information, please contact Digital Projects.
© 2015 All Rights Reserved

UNF UNIVERSITY of NORTH FLORIDA

A TARGETED DENIAL OF SERVICE ATTACK ON DATA CACHING NETWORKS

by

Jeffrey B. Gouge

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computing and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

August, 2015

Copyright (©) 2015 by Jeffrey B Gouge

All rights reserved. Reproduction in whole or in part in any form requires the prior written permission of Jeffrey B Gouge or designated representative.

The thesis "A Targeted Denial of Service Attack on Data Caching Networks" submitted by Jeffrey Gouge in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee: Date

____________________________ ____________

Dr. Swapnoneel Roy
Thesis Advisor and Committee Chairperson

____________________________ ____________

Dr. Sanjay Ahuja

____________________________ ____________

Dr. Karthikeyan Umapathy

____________________________ ____________

Dr. Anand Seetharam

Accepted for the School of Computing:

____________________________ ____________

Dr. Sherif Elfayoumy
Interim Director of the School

Accepted for the College of Computing, Engineering, and Construction:

____________________________ ____________

Dr. Mark Tumeo
Dean of the College

Accepted for the University:

____________________________ ____________

Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENT

I would like to thank so many people that have helped me on this thesis and my many years of graduate school. First, I would like to thank Dr. Swapnoneel Roy for being a great advisor and mentor to me. Thank you for the countless hours and advice throughout the thesis process. Thank you also to Dr. Anand Seetharam for advising me weekly and working so closely with Dr. Roy and me. Our weekly meetings were a great help along the process of working on this thesis and provided a great framework for learning.

I would also like to thank my committee members, Dr. Sanjay Ahuja and Dr. Karthikeyan Umapathy for working with me. The advice given from both of you in your respected fields helped me tremendously.

I would also like to send a special thank you to my wife, Elena Gouge. Without her, this thesis would not be possible. She was able to help me edit my paper through the many revisions which was a great service. She also put up with me through the many years of my graduate degree while raising our first child, Logan Gouge. She was able to put in many extra hours while I worked on my degree, and without her I would not be where I am today. Thank you also to my family for being there for me and supporting me throughout my many years of schooling.

Thank you to my many coworkers and fellow students who also were able to listen to me work out problems and work on ideas throughout the thesis process. A few people that were a huge help throughout my thesis were William Carle, and Geoff Whittaker. Thank you all for your time, thoughts, advice, and help.

CONTENTS

List of Figures........ix

List of Tables........x

Statement on Security Research........xi

Abstract........xii

Chapter 1: Introduction........1

1.1 Problem Statement........3

1.2 Contributions........4

Chapter 2: Background and Literature Review........6

2.1 Background........6

2.1.1 Information Centric Networks (ICN)........6

2.1.2 Peer-to-Peer Networks........7

2.1.3 MANET........9

2.1.4 Gnutella........9

2.1.5 Hybrid Networks........10

2.1.6 Content Centric Networking / Named-Data Networking........11

2.1.7 Caching........12

2.1.8 Computer Security........13

2.2 Known Results and Related Work........15

Chapter 3: Research Methodology........18

3.1 Design Science........18

3.2 Design Science Guidelines 18

3.2.1 Design as an Artifact 18

3.2.2 Problem Relevance..20

3.2.3 Design Evaluation 19

3.2.4 Research Contributions..20

3.2.5 Research Rigor..21

3.2.6 Design as a Search Process..21

3.2.7 Communication of Research..21

Chapter 4: Proposed Attack Theory and Methods 23

4.1 Objective and Scope..23

4.2 Proposed Attack Method..24

4.2.1 DoS Attack Method..26

4.3 Proposed Attack Implementation..32

4.3.1 Technology..32

4.3.2 Java ICN Simulator & Attack Implementation..32

Chapter 5: Evaluation 40

5.1 Evaluation Objectives 40

5.1.1 Evaluation Limitations 43

5.2 Experiment Scenarios..44

5.2.1 Scenario 1: Line Graph..44

5.2.2 Scenario 2: Square Graph 45

5.2.3 Scenario 3: Real World (Gnutella) Graph 47

5.3 Evaluation Results 48

5.3.1 Scenario 1 – Line Graph Results..49

5.3.2 Scenario 2 – Square Graph Results 51

5.3.3 Scenario 3 – Gnutella Graph Results 56

5.4 Discussion..61

5.5 Future Work..63

Chapter 6: Conclusion…..66

References...67

Appendix A: Source Code Examples 70

Node Class Example 71

Edge Class Example 71

Graph Class Example..72

Content Class Example 73

Packet Class Example 74

AttackerNode Class Example..75

Appendix B: Results Collected 80

Vita 121

FIGURES

Figure 1: Overview of Content Centric Networking Node..11

Figure 2: Named Data Networking Overview 12

Figure 3: Normal Data Caching Network...25

Figure 4: DoS Attack on Data Caching Network...26

Figure 5: Cache Size Estimation Algorithm from Dehghan *et al.* 28

Figure 6: Characteristic Time Estimation Algorithm from Dehghan *et al.* 29

Figure 7: Java Simulator Test Results Example..39

Figure 8: Line Graph 45

Figure 9: Square Graph..46

Figure 10: Line Graph Network Average Number of Hops 50

Figure 11: Line Graph Network Percent Increase..50

Figure 12: Square Graph - 25 Nodes, using LRU, and Attacker Request Rate of 1 52

Figure 13: Square Graph - 100 Nodes, using LRU, and Attacker Request Rate of 1 53

Figure 14: Gnutella Graph - 6301 Nodes, using LRU, and Attacker Request Rate of 1...58

Figure 15: Gnutella Graph - 8846 Nodes, using LRU, and Attacker Request Rate of 1...58

TABLES

Table 1: Development Machine Information..33

Table 2: Number of Unpopular Files Test 41

Table 3: Line Graph Percentage Increase Statistics 51

Table 4: Square Graph 25 Node (Zipfian=0.65): Percentage Increase Statistics..............55

Table 5: Square Graph 100 Node (Zipfian=0.65): Percentage Increase Statistics............56

Table 6: Gnutella Graph 6301 Node (Zipfian=0.65): Percentage Increase Statistics 60

Table 7: Gnutella Graph 8846 Node (Zipfian=0.65): Percentage Increase Statistics 61

STATEMENT ON SECURITY RESEARCH

This research was performed with local software simulation on hardware that was not connected to the Internet. All testing was performed within Java objects that only communicated with one another on one computer. No simulation between hardware devices was tested during this research.

This research was performed for purely academic reasons to study the security of data caching networks and a possible new attack vector. UNF does not promote the use of any attack outside of an approved testing environment designed for computer research. This research discusses the possibility of using this new attack on real world networks and what the effects of the attack would be. Any further research on this topic should be done locally and never used against live networks or computers not owned by the tester. Future research is promoted in this paper on the topic of finding methods to detect and prevent such data caching attacks in the future.

ABSTRACT

With the rise of data exchange over the Internet, information-centric networks have become a popular research topic in computing. One major research topic on Information Centric Networks (ICN) is the use of data caching to increase network performance. However, research in the security concerns of data caching networks is lacking. One example of a data caching network can be seen using a Mobile Ad Hoc Network (MANET).

Recently, a study has shown that it is possible to infer military activity through cache behavior which is used as a basis for a formulated denial of service attack (DoS) that can be used to attack networks using data caching. Current security issues with data caching networks are discussed, including possible prevention techniques and methods. A targeted data cache DoS attack is developed and tested using an ICN as a simulator. The goal of the attacker would be to fill node caches with unpopular content, thus making the cache useless. The attack would consist of a malicious node that requests unpopular content in intervals of time where the content would have been just purged from the existing cache. The goal of the attack would be to corrupt as many nodes as possible without increasing the chance of detection. The decreased network throughput and increased delay would also lead to higher power consumption on the mobile nodes, thus increasing the effects of the DoS attack.

Various caching polices are evaluated in an ICN simulator program designed to show network performance using three common caching policies and various cache sizes. The ICN simulator is developed using Java and tested on a simulated network. Baseline data are collected and then compared to data collected after the attack. Other possible security concerns with data caching networks are also discussed, including possible smarter attack techniques and methods.

Chapter 1

INTRODUCTION

The Internet has grown in the 21th century to become the main source of communication and data exchange. The rise in mobile smart phone usage has also increased the amount of data exchange and the number of people requesting content. Examples of content include documents, videos, images, audio, and metadata. As technology continues to get better, the byte size of the data also continues to increase. For example a standard definition video which, has an average resolution of 480p. A standard DVD disc can hold up to 4GB of data, which displays video in 480p resolution. Many popular cameras and video cameras in 2014 are capable of shooting videos in high definition, or 1080p resolution. A standard Blu-Ray disc can hold up to 50GB of data, which displays video in 1080p resolution. Similar types of file size increases can be seen with higher resolution images as technology continues to develop. With more people capturing large multimedia there comes a need to present this media quickly, and keep the high quality for remote users. The rise in technology has kept to “Moore’s Law” in doubling of technical capacity every two years [Schaller97], but many researchers suggest that the network technology is not keeping to the law [Coffman02].

Data or content caching was used as a way to increase the performance of data exchange over the network. Many peer-to-peer and mobile networks require data caching to increase network performance while providing the most energy efficient solution. To reduce the

amount of network traffic and help solve any issues related to the geographic distance of client and server data exchange, Content Delivery Network (CDN) architectures were put in place in heavy metropolitan areas. Content Delivery Networks place caching servers on the edge of networks so requests can be served in geographical proximity to users. Another networking architecture, Named Data Networking (NDN), has also been proposed where data caching would be done on routers. This type of network would require existing routers to be upgraded to faster and much more expensive hardware.

Network security has become a very popular topic in the world of computing since the invention and rise of the Internet. The Internet structure itself allowed for many security vulnerabilities to occur between remote computers. Research on the Internet (its vulnerabilities, attack methods, and security technology) has become a vital necessity to maintain the confidentiality, integrity, and availability of personal and corporate assets. Technologies such as firewalls and other security devices and methods allow for businesses to be protected within an “intranet” but still have the availability to connect to the Internet. Other security devices such as Intrusion Prevention Systems (IPS) allow security professionals to monitor and actively defend against security threats. Security research allows for such devices and methods to stay up to date on the latest vulnerabilities and attack methods. Newly found and evaluated vulnerabilities and methods allow for security devices and professionals to accurately detect and prevent such issues in the future. [Daya13]

1.1 Problem Statement

Data caching networks, such as an ICN, rely on the use of caching to increase network performance by minimizing the number of hops for given requests. As mentioned, many different networks have been created and researched to solve the problem of fast data exchange. ICNs have also introduced a new way to find content, based on the content data itself instead of an IP centric or host to host model [Brito13]. A main component of ensuring high performance in data exchange on networks is "data caching". Many companies and researchers use load testing to benchmark network limitations and show that data caching can greatly improve network performance [Bžoch12]. But what are the security implications of attacking the data cache itself? As shown in Chapter 2, security research is lacking in the field of data caching networks and attacks targeted at cache pollution. This thesis aims to show and discuss the security issues arising from attacking the use of data caching within networks and proposes a new type of DoS attack.

Any gap in research and knowledge can lead to possible security vulnerabilities that may not be well known until exploited. Take, for example, zero day attacks that use methods and vulnerabilities that are not known by the attacked company or technology until they are used against them. The recent vulnerability discovered in SSLv3 protocol is a perfect example of vulnerabilities that exist in the world today that are not known until research or live attacks are performed. These types of threats can be minimized by security research, testing, and general awareness. Data caching network security currently lacks an in-depth look at security attacks targeting the use of data caching and thus presents a problem.

1.2 Contributions

With the growing dependence on data caching within networks, a new DoS threat can be formed that targets the use of data caching strategies. The new creative and adaptive method targets the goals of data caching and uses prior research on inferring network traffic from data cache behavior [Dehghan13]. This targeted DoS attack would prove to decrease network throughput and increase network delay. This method is defined, implemented, and tested on a data caching network simulator. The scalability of the attack implementation is tested across 3 network topologies ranging in size from 5 nodes to 8,846 nodes. The attack implementation is also tested across 3 caching strategies and 5 different cache sizes per node. The network simulator was developed to test an ICN using custom object oriented classes that were developed to evaluate the new attack implementation. The network simulator software was developed to test a network using software testing limited to the local machine. No network hardware or external computers were used in the simulator testing. Three different caching strategies were also tested in the simulator using a Java implementation of the DoS attack. The scope of the research is limited to internal attacks. For internal attacks, it is assumed that an internal node in the network has already been compromised. No details are provided on how this node was compromised, nor will the security implications of attacks not related to data caching be discussed. This new attack and discussion of security issues concerning data caching networks aims to open up more research and awareness in the field of computer security.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

2.1 Background

In this chapter, various concepts and technologies that relate to a DoS attack on data caching networks are discussed and defined. The main topics include network technologies, data caching, and computer security. A defined understanding of these topics and terminology will be needed to fully understand the proposed new attack method.

2.1.1 Information Centric Networks (ICN)

The most widely used networks today are host centric networks that use an IP address route requests. Requests are bound to a physical geographic location that all requests must communicate to and from. Information or content centric networks aim to change this paradigm to a data-based approach for delivering content. This networking approach allows for faster information access regardless of location. For example, a user requests a video to download. The router takes this request, searches nearby hosts for this content, and routes the request to the closest host with that content to serve the request. The history of ICNs come from content delivery systems such as publish/subscribe architectures and peer-to-peer networks where the main goal was content dissemination.

Key differences between ICNs and host centric networks include naming, routing, security, and API. ICNs give names to unique pieces of content instead of giving names to hosts. Routing is performed by routing requests between a client requester and optimal content sources. Host centric networks route requests between source and destination nodes that are identified using a physical geographic location on the network with an IP address. Packet headers are used to send chunks of data, hop by hop, from the browser to the server and back. The packet headers contain the IP address which describes the location of the destination to all routers so they know where to forward the request [Brito13]. Clearly, if the end destination server is not geographically located near the client, the request could take some time to complete as it needs to travel from router to router, possibly across the globe. Security in host centric networks focus on having a secure communication channel between source and destination hosts. ICNs secure the integrity of the content itself and make sure it is not altered regardless of how the content is delivered. APIs exposed by ICNs create methods that allow for content to be published and consumed, whereas host centric networks allow data to be sent to given physical geographic locations [Tyson13].

2.1.2 Peer-to-Peer Networks

The roots of ICNs are found in the early peer-to-peer networks that allowed for a new and unique way to send and receive data. Peer-to-peer networks are a form of network architecture in which host computers send and receive data from one another without the need to go to a separate server. Each computer acts as both a client and a server in a peer-

to-peer network. A good definition of peer-to-peer systems was proposed by Androutsellis-Theotokis in 2004 [Androutsellis-Theotokis04],

> *Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.*

The most widely known and used peer-to-peer systems are used for sharing content, which first became popular with Napster. Napster was forced to shut down after violating the Digital Millennium Copyright Act. Shortly after that time, a file sharing protocol, Bit Torrent, became very popular and heavily used. The Bit Torrent protocol [Pouwelse05] "splits files into chunks and the downloaders of a file barter for chunks of it by uploading and downloading them. When a peer has finished downloading a file, it may become a seed by staying online and sharing the file." Users looking to download files must find a Bit Torrent web site that hosts directories of available files, called torrent files. The torrent file contains information on what tracker to use to download this file. The trackers [Pouwelse05] "keep a global registry of all the downloaders and seeds" of the given file. As seen, this protocol still relies on some servers in between the peers in order to share files across a network.

2.1.3 MANET

Over the years, many different peer-to-peer networks have been created that focused on fixed peer computers. A Mobile Ad hoc Network (MANET) aims to benefit from the advantages of peer-to-peer while using highly mobile peer computers. A MANET is made up of mobile hosts, or peers, that use a store-and-forward method to send and receive packets on the network via a wireless network connection. Since the nodes are mobile, all changes in physical location must be communicated to the entire MANET network so every node can update its network topology. The distance between the nodes also must be taken into consideration. If a mobile node is using a battery instead of a directly connected power source, then the node must see if the remaining battery power has enough power to send data to a node. It is possible that a node with a low battery may only be able to send data to a node very close to its physical location. In the original MANET networks, only the routes were cached for faster read and write access [de Morais Cordeiro02].

2.1.4 Gnutella

Another example of a peer-to-peer network topology that was used heavily in large scale deployments was Gnutella. Gnutella is an unstructured and decentralized peer-to-peer network that became popular in the early 2000s. The Gnutella topology was more robust and self-healing compared to other peer-to-peer topologies due to the lack of structure and decentralization. It was the first decentralized peer-to-peer network of its kind, which lead to many new network topologies since that time. The Gnutella topology was used heavily

with peer-to-peer file sharing clients such as Limewire. Research in Gnutella showed that the topology was scalable, which was needed for the file sharing peer-to-peer services that grew heavily in popularity with services like Limewire [Chandra10].

2.1.5 Hybrid Networks

Many devices have been produced with support for multiple communication technologies. A simple smart phone sold in 2015 would have the ability to connect to other devices using the cellular network, WIFI, Bluetooth and even RFID or NFC (Near Field Communication). Deploying a MANET network using devices such as current generation smart phones would allow for the use of a hybrid network. A Hybrid MANET network can be seen as a MANET network where certain nodes (or all nodes) have the ability to use another nearby existing network for communication.

Take a simple example of troops deployed to foreign countries. Each troop would be equipped with a communication device to make sure they can connect to all nearby troops. With the rise in use of cellular networks, there would be a strong possibility that the deployment location has a nearby cellular tower. Troops could then use that existing network to pass information between squads by using the cellular network regardless of physical geographic proximity. This would allow for much faster communication and data transfer than having to use satellite communication devices. If a General was in one location and gave out new orders or vital documents, the data could be transferred to all nearby troops using the MANET network, then passed to other squads using the cellular network. The same example could be used for troops requesting new information. The

request would be made within the MANET network and if it wasn't on the network, the backup cellular network could be used.

2.1.6 Content Centric Networking / Named-Data Networking

One form of ICN architectures is Content Centric Networking (CCN). Named content is the main component of content centric networking. The most distinguishing feature of CCN is the ability to divide content into chunks so that each chunk has a unique name, ordered identifier, and can be requested individually. There are two different types of packets in CCN: interest packets (I-packet) and data packets (D-packet). An interest packet is a packet that contains information on a certain chunk of content and information on the node with interest (the requester). The interest packet is then broadcasted on the network until a node is found with the specific chunk of content. If a node receives an interest packet and does not have the content stored locally, it forwards the interest packet to its neighbors until a node that has stored the requested data is found. The data packet is then sent back to the requester in response to the interest packet. Routers in CCNs use a content store (CS) to store content on the router using a caching policy such as least recently used (LRU). These routers also contain a pending interest table (PIT) and a forwarding information base (FIB). The PIT is used as a routing table to store interest packets that have been forwarded. This allows for data packets to be sent back correctly in response. The FIB is a database that stores the mapping between the content names and the output interface. See Figure 1 for an overview of a CCN node [Brito13].

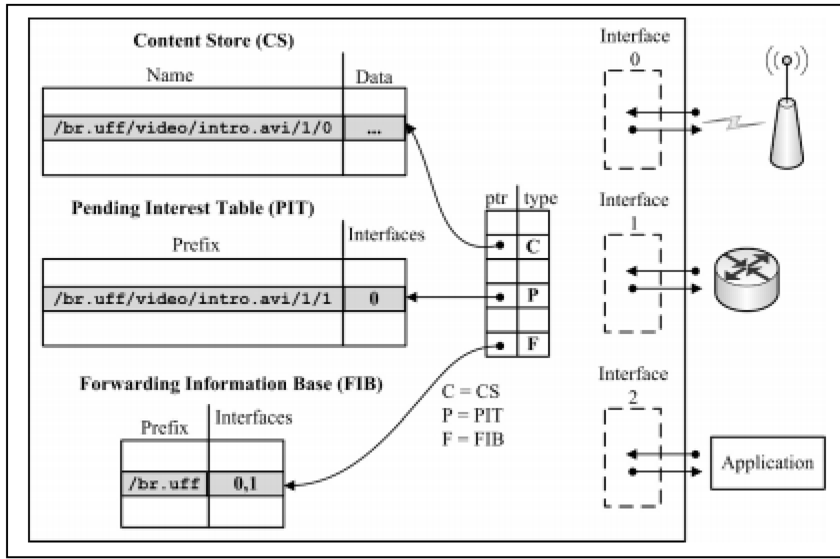


Figure 1: Overview of Content Centric Networking Node

NDN is a popular design architecture that implements content centric networking. It uses both interest and data packets as required by content centric networks (Using NDN notation, a "/" separates name components and fragments are created by adding another "/" and adding the fragment identifier). For example, fragment 3 of Jeff.jpg could be named /facebook/gougejeff/2014/photos/Jeff.jpg/3. Figure 2 shows an overview of NDN including the interest packet request and the received data packet [Conti13].

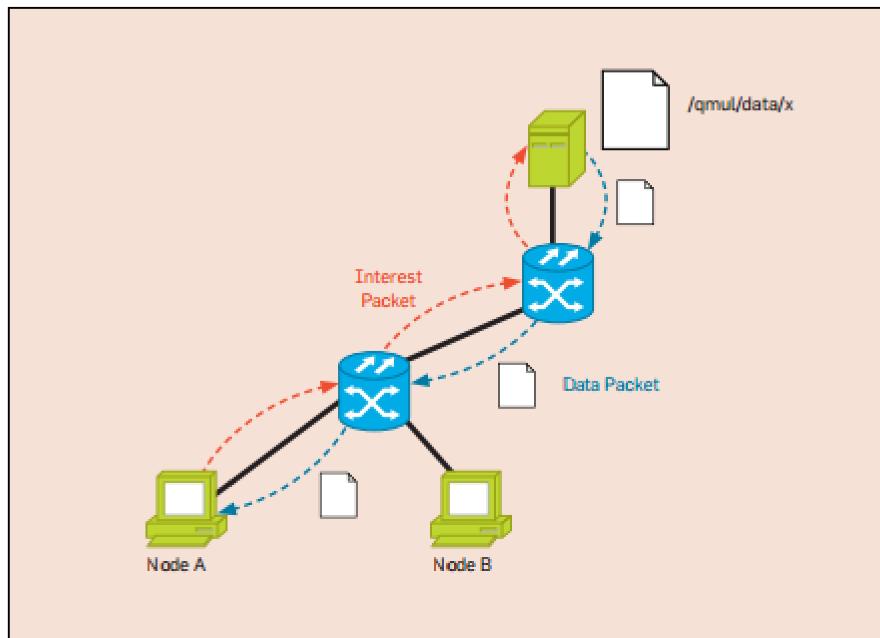


Figure 2: Named Data Networking Overview

2.1.7 Caching

Caching is a concept of storing data in memory to allow for faster data access. Routers implement route caching of routing tables to allow for faster querying. Web servers are also known for using caching to store web pages and multimedia. There are many different caching policies that are used, and each has advantages and disadvantages.

LRU or Least Recently Used caching policy is a commonly used caching policy [Bžoch12]. It allows for storing of the most recently used content, and data are purged from the cache using a strategy given by its name. Items are removed from LRU cache when the cache hits its maximum size and the item that was least recently used by that cache is removed from the cache. LRU tends to be one of the best caching solutions for caching of large files

[Bžoch12]. It is usually implemented using a priority queue, which using the last access timestamp as the priority. FIFO (First in First out) is another type of caching policy. Items are placed and removed from this type of cache based on a queuing strategy. Items placed into the cache are put into the tail or back of the queue and once its maximum size is reached, the item at the head or front of the queue is removed. Random cache is another caching strategy where items are place into the cache at the start of a list. When the maximum size is reached, a random index of that list is removed and the new item is put into that index [Bžoch12].

A form of caching content in different geographic locations can be seen in content delivery networks (CDN). The CDN is essentially just a collection of servers that store local copies of specific content on servers located strategically across the Internet. Content providers such as Netflix, Youtube and other companies use commercial CDNs to offload content hosting responsibility and allow for very high availability of content all over the world [Saroiu02]. Each CDN server can be seen as a data caching server that would serve requests for content instead of forwarding the request to the host.

2.1.8 Computer Security

In recent history, some of the most popular news stories surrounding the field of computing have been security related incidents. Every year, a number of large companies are affected by computer security incident(s) that lead to large data loss and sometimes business closures. The common core principles of information security are confidentiality, integrity,

and availability, which is commonly referred to as CIA. Confidentiality refers to ensuring that best practices and methods are used to ensure all information is only available to those who have the need, or the right, to see it. The integrity of a computing system is described as the prevention of unauthorized or improper modification of systems or information. The accuracy of data is very important, especially in systems that have dependences on the correctness of that data to make decisions and take actions. Availability refers to prevention of disruptions in service or productivity. The main goal of information security is to ensure each of these principles for all company assets including physical and informational assets [Hernandez09].

As technology changes every day, an increasing amount of security vulnerabilities are found in developer code. The most popular security attacks are targeted at remote servers and applications available on the Internet, which lead to web developers code being the most often attacked. Every year, the Open Web Application Security Project (OWASP) produces a top 10 list of the most critical web application security risks. The list is meant to be used as a resource to all developers to raise awareness of the most popular application security vulnerabilities and how to avoid them while developing applications. Top attacks in the last couple years include: Buffer Overflow attacks, injection attacks (SQL/QueryString paramaters/etc.), broken authentication and session management, cross-site scripting (XSS), using known vulnerable software components, and others [Stock13].

Attacks on application security is just one of the many attack vectors malicious users can target to compromise computer security. The availability of computing resources and

applications is another commonly attacked vector in computer security. Every day, malicious users throughout the world are targeting web sites and applications available on the Internet in an attempt to decrease performance or ultimately crash remote resources. This type of remote attack is called a denial of service attack (DoS). A DoS attack can originate from a single source, or from multiple sources all with the same attack destination which is labeled a distributed denial of service attack (DDoS). With the computing power available today, DDoS attacks are one of the most popular and effective ways of compromising the availability of remote services.

There are many tools and methods for protecting computers and data from security vulnerabilities. There are also many ways for securing the actual communication channel itself from attackers trying to take, decrypt, and change the data in route. Tools such as encryption, authentication mechanisms, intrusion detection/prevention systems, and firewalls are all examples of security tools that work to detect and prevent threats and attacks. Each of these tools depends on continuous research in order to stay current with new attacks and methods found in production and research environments.

2.2 Known Results and Related Work

There have been many research papers that discuss ICNs and caching related issues since heavy caching is one of the main concepts of ICN. The main motivation of this research was all started from a research paper, “Inferring Military Activity in Hybrid Networks through Cache Behavior” by Mostafa Dehghan, Dennis L. Goeckel, Ting He, and Don

Towsley [Dehghan13]. The research shows that mobile hybrid networks have a vulnerability where military network activity can be inferred by examining the cache hit/miss of a single adversary node. The paper discusses the concept of characteristic time (T*) which is used to describe the amount of time on average a single piece of content will stay in a single nodes cache. A method is discussed on how to determine the cache size being used by other nodes on the network as well, which is used as a basis for determining the characteristic time.

After taking a look at other research in this area, a gap was found in security research on caching. The previous paper stated that this research field was lacking and that the paper had hopes of starting new research in this field. While performing research, many papers on detecting attacks or vulnerabilities in networks including ICNs were found but only one paper was found on the details of such an attack.

The paper on detecting cache pollution attacks in NDN by Mauro Conti, Paolo Gasti, and Marco Teoli was found as the only paper describing the feasibility of the attack [Conti13]. This paper took a direct look at a new method of detecting cache pollution attacks, specifically in NDNs using concepts and methods used in this example of an ICN. The paper also shows that this type of cache attack is not only viable in smaller networks, as it was once shown to be, but also for much larger network topologies. The paper also suggests a way to improve a cache pollution attack found in a previous research paper by Xie et al. [Xie12]. A comparison is also used in this research paper to CacheShield [Xie12], which is the only known countermeasure to cache pollution attacks designed specifically

for NDNs. Using the concepts found in this research, some improvements are made on the proposed attack method, which can be found in Chapter 4. Also, the proposed attack is not targeted for NDNs, but any network architecture that uses data caching thus the detection methods specific to NDNs are viable.

Chapter 3

RESEARCH METHODOLOGY

3.1 Design Science

Design science research is the research methodology used in this thesis. The main goal of design science research is defined in its seven objectives or guidelines. The outcome of design science research in Information Systems disciplines is to create solutions (artifacts) that have been evaluated, and to share the results of the evaluation with the community [Hevner04].

3.2 Design Science Guidelines

In this section, the seven design science guidelines are discussed and defined. Each guideline is discussed and are used to help researchers conduct and evaluate research based on this methodology. All guidelines are just guidelines and are not strictly enforced [Hevner04].

3.2.1 Design as an Artifact

The first guideline of design science research states that an artifact must be created in the form of a construct, a model, a method, or an instantiation [Hevner04]. A clearly stated

and viable artifact must be created. For this thesis, the goal is to create a working method and implementation of performing a DoS attack on data caching networks. The details of the method is described in full detail in Chapter 4.

3.2.2 Problem Relevance

The objective of the problem relevance guideline is to ensure that the solution is important and relevant in the real world [Hevner04]. As stated in the introduction, data caching has become a very popular way of increasing network performance in many different networks. As more networks become reliant on data caching, a clear understanding of all security concerns need to be defined. If attacks and prevention techniques are not researched, then systems could be developed with future security vulnerabilities unknown to the general public. The reliance of this problem is established in Chapter 1.

3.2.3 Design Evaluation

The design evaluation guideline states that the artifact should be demonstrated via a well-executed evaluation method [Hevner04]. Five possible evaluation methods are observational, analytical, experimental, test-based, and descriptive. Observational evaluation is performed via case studies or field studies where multiple solutions are studied and discussed. Analytical evaluation involves performing analysis of various qualities of the solution. Experimental evaluation is performed with controlled experiments or simulations. Test-based evaluation is performed with the use of black-box

and white-box testing strategies where solutions are tested for expected results. Descriptive evaluation is performed using arguments, scenarios, or discussions on the solution. This thesis uses the experimental evaluation method with the simulation performed on a sample data caching network. An ICN software simulator is used as a simulation baseline and evaluated using three different scenarios as described in Chapter 5.

3.2.4 Research Contributions

The research contributions guideline states that the research must contribute to the designed areas of expertise in a clear and verifiable way [Hevner04]. The goal of this research is to create a new method and implementation of a DoS attack on data caching networks. The attack method created can be used to affect many devices that makes use of a data cache including routers, CDNs, webservers, and other hardware devices. A software ICN simulator was also developed to evaluate the DoS attack implementation. This artifact contributes to the overall security research field by providing a new documented attack that can be further studied and researched. Part of this research goal is to take the first steps into finding ways of detecting and preventing such types of attacks on data caching networks.

3.2.5 Research Rigor

The research rigor guideline states that the artifact should be constructed and evaluated through the application of rigorous methods [Hevner04]. The DoS attack created in this thesis was based on prior research methods developed and evaluated to infer military traffic. The method involved finding a unique way to use a data cache to infer information about cache usage. Using that research, combined with research on security attacks on networks, the DoS attack on data caching networks method was developed.

3.2.6 Design as a Search Process

The design as a search process guideline states that the artifact should be the result of a search process that is set to find the best solution to the problem [Hevner04]. The DoS attack on data caching networks method was developed with current research articles that were found in published articles and papers. The simulated attack was developed using existing technologies and network architectures in place in research and production environments today.

3.2.7 Communication of Research

The objective of the communication guideline is to present the solution to both a technology-oriented as well as a management-oriented audience [Hevner04]. The technical implementation details is described in Chapter 4 and working code samples are

provided in Appendix A. The thesis was presented to the public community of the University of North Florida in the form of a written document and a final defense.

Chapter 4

PROPOSED ATTACK THEORY AND METHODS

4.1 Objective and Scope

The objective of this thesis is to design a new method and implementation for a DoS attack on data caching networks. The new targeted attack method is described in detail in section 4.2, followed by a description of the Java implementation of the attack. The DoS attack is designed to be an internal attack. The attack can be started and run using a malicious node or a set of malicious nodes within a data caching network. The attack is not designed to spread or replicate from node to node, but the impact of the attack increases as the number of malicious nodes increases, as seen in most DoS attacks. The impact of the attack could also increase as the distance in hops between the requester and the content custodian increases. The attack is also designed to run for an infinite amount of time or until the attacker chooses to stop the attack. While the attack is running, network performance is expected to be adversely affected and should not improve until the attack stops.

This DoS attack can be used on any data caching network. This includes networks that have any node that implements data caching. One example of a data caching node could be a web server that hosts or caches video content. The attack would cause the cache to become filled with unpopular content that was not previously in the cache. This would cause requests for valid and popular content to be affected by the attack and thus take more

hops to complete. Another example would be any network that has data caching on nodes along the path to the content custodian. This type of network would cache content on nodes as requests pass through the node. Any piece of content that is not in the node's cache, will be added to the cache so if a similar request is received it will respond with the content in its cache. This type of network can be seen in a hybrid mobile ad-hoc network (MANET) or ICNs as described in Chapter 2.

The scope of this DoS attack is limited to looking at the attack and its risk impact on data caching networks. The security of the network activity over the wire is not discussed in this thesis. The process of obtaining control of the malicious node is also outside of the scope of this thesis. The DoS method described in section 4.2 assumes that the node has been compromised or that a malicious user is now in control of the node. Security attacks that attempt to alter the state of the content in the cache or vulnerabilities in the technologies used for content caching are also not discussed in this thesis.

4.2 Proposed Attack Method

Figure 3 shows a diagram of normal activity on a data caching network. Figure 4 shows a diagram of the DoS attack in a data caching network that uses content caching at each node. The following section provides a detailed description of the proposed attack method used for the targeted DoS attack on data caching networks. As stated in step 1, it is assumed that a malicious node has been compromised, yet undetected, and is used as the attack vector.

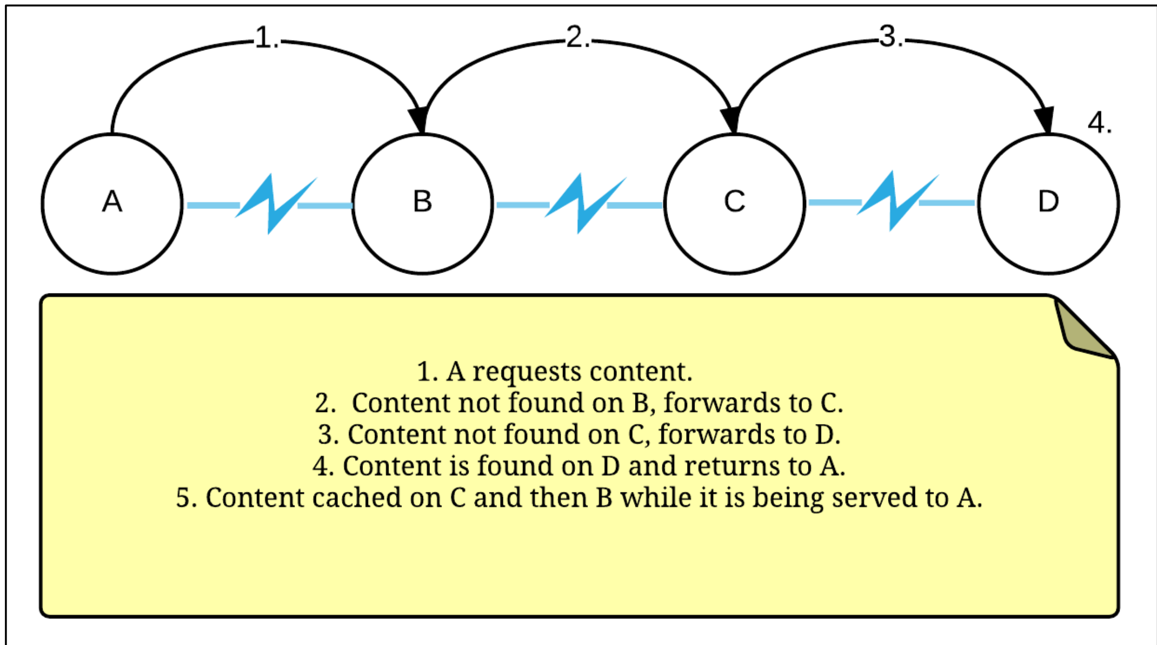


Figure 3: Normal Data Caching Network

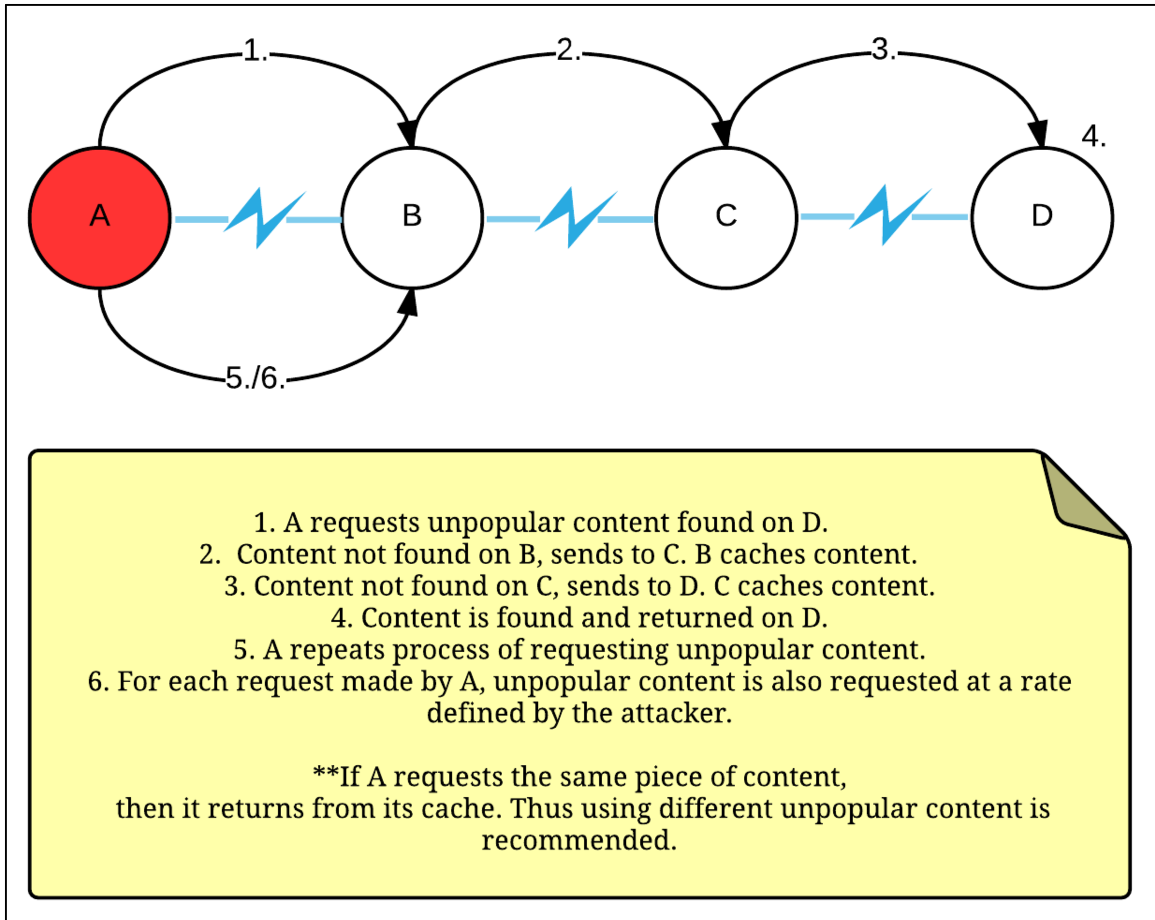


Figure 4: DoS Attack on Data Caching Network

4.2.1 DoS Attack Method

1. We assume that the malicious node on the network is compromised through an out of scope vulnerability or method that occurred in an undetected way. It is also assumed that this malicious node is known as non-malicious by other nodes on the network.
2. The malicious node then begins a search of all content on content custodians in the network. The malicious node creates a list of content ordered by the last access time stamp or by searching and getting all content on the custodian from a list of

keywords. Keywords are used to find content that is older, archived, or uncommon (eg. archived, old, 1/1/1999). The keyword list is then used to choose the most unpopular content from the list obtained. The top results from this list are then marked to be used for the attack. The attacking node can also get this list of unpopular content by polling the content that it routes. Keeping a list of all the content that is routed through the node gives the node a popularity distribution of content that it can use to identify the least popular content. This list of unpopular content is used to carry out the attack.

3. An estimate of the cache size used on the network is created. The research algorithm proposed by Dehghan, Goeckel, He, and Towsley's paper on "Inferring Military Activity in Hybrid Networks through Cache Behavior" is used to continuously get a better estimate of the cache size, which is seen in Figure 5 [Dehghan13]. If the attacker is knowledgeable about the targeted network in advance through security analysis, then the effects and speed of the attack are increased. It is also an option to skip this step since it can be a safe assumption that most nodes have the same cache size on a given ICN.

Algorithm 1 Cache Size Estimation

```
1: Let c be an initial guess for the cache size.
2: Pick c distinct files, and request them in rapid successions.
3: Request the c files again, but in reverse order.
4: Let δ be the number of requests that resulted in cache hits.
5: if δ = c then
6:     c ← 2 * c.
7:     goto 2.
8: else
9:     Estimate cache size as δ.
```

Figure 5: Cache Size Estimation Algorithm from Dehghan *et al.*

4. (Optional Smart Attack) An estimate of the characteristic time (T^*) is then created. The research algorithm proposed by Dehghan *et al.* is shown in Figure 6, which is based on a binary search [Dehghan13]. A slight variation of the algorithm is proposed below which starts by waiting a larger amount of time, then reducing the guess until an acceptable value is found. The goal of the variation in the algorithm is to develop an easier and faster implementation of the algorithm. The process is described below.

    a. The first stage of the characteristic time calculation is to send a request for a piece of unpopular content then compare the number of hops to the custodian to the number of hops the request actually took. The attacker is looking to see if the requested file was served by the content custodian or by a node in the route that has the content cached, thus resulting in a cache hit. It is important that the request is made for a content item that is not requested by other users or one that is seen as very unpopular. If the content requested is popular, then the newly proposed algorithm is altered and the

results will not be accurate. For this step, the attacker chooses the least popular file from the unpopular files list.

Algorithm 2 Characteristic Time Estimation

1: Select a file $f' \notin F$ unknown to other users.
2: Arbitrarily let τ be an initial guess for the characteristic time.
3: Let $l \leftarrow 0$ and $h \leftarrow \inf$
4: **while** $h - l > \Delta$ **do**
5:     **for** b times **do**
6:         Request f'. Record the hit/miss.
7:         Wait τ units of time.
8:     Let δ be the number of hits.
9:     **if** $\delta < b/2$ **then**
10:         $h \leftarrow \tau$, $\tau \leftarrow (l + \tau)/2$.
11:     **else**
12:         $l \leftarrow \tau$.
13:         **if** $h < \inf$ **then**
14:             $\tau \leftarrow (\tau + h)/2$.
15:         **else**
16:             $\tau \leftarrow 2 * \tau$.
17: Estimate T^* as $(l + h)/2$.

Figure 6: Characteristic Time Estimation Algorithm from Dehghan *et al.*

b. The first stage of the characteristic time calculation is to send a request for a piece of unpopular content then compare the number of hops to the custodian to the number of hops the request actually took. The attacker is looking to see if the requested file was served by the content custodian or by a node in the route that has the content cached, thus resulting in a cache hit. It is important that the request is made for a content item that is not requested by other users or one that is seen as very unpopular. If the content requested is popular, then the newly proposed algorithm is altered and the

results will not be accurate. For this step, the attacker chooses the least popular file from the unpopular files list.

c. Wait T* or the characteristic time guess, and repeat the request. The first iteration of the newly proposed algorithm should use a larger T* initial guess. Compare the number of hops taken in the request to the number of hops to the custodian.

d. If the second request was served by the custodian, then the attacker knows the content was purged from all caches in the path. Otherwise the content is still cached at a node in the path.

    i. If the request was served by the custodian, then the guess of *T** is too big and a lower value should be tested. Now, set the characteristic time guess to T*/2. Repeat step 4a with the new smaller *T** guess.

    ii. If the request was not served by the custodian and served from a node's cache, then the guess of T* is a good value and can be used as a best guess of the characteristic time. Once this step is reached, the characteristic time calculation is complete and the attack can be started.

5. Perform the Attack

    a. If the smart attack is used, the DoS attack starts once a good value for characteristic time is found. If the smart attack is not used, then the attacker sends requests for unpopular content from the list acquired in step 2. The attacker should only send requests for any content according to the rate

observed in normal activity or just above the normal rate. The attacker request rate was defined as the number of requests for unpopular content (attack requests) that the attacker node completed after a normal request. This thesis used attacker request rates of 1, 2, and 4. These three values were arbitrarily chosen as a good distribution of attacker request rates to show an impact, but to also prevent detection. Increasing the attacker request rate above 4 could be too much traffic to send in succession and could lead to higher chances of detection.

b. A single number increment is placed on each piece of unpopular content that is requested. For every normal request, any unpopular files that have been sent are incremented by one. If the smart attack is being used, the attacker can only choose unpopular files that have a number increment value less than the characteristic time (T*). The attacker waits at least the characteristic time (T*) between requests for unpopular files. This ensures the requested unpopular content leaves the cache of all nodes in the route path. It also ensures randomly requested content to help prevent detection. If the smart attack is not being used, then the attacker node picks a random unpopular file from its unpopular file list.

c. The attack continues sending requests for unpopular content until the attacker wants to stop the attack.

The concept of characteristic time, in theory, allows for the attack to better affect every node in the path between the requester and the content custodian. If the first node in the

path keeps a cached copy of the content, then requests for the same piece of content before the characteristic time would just return from that first node's cache. This is the main reason the concept of characteristic time is so important to this attack. Waiting a minimum of at least the characteristic time between repeat requests allows for the attacker to request the same piece of content in a looping fashion.

4.3 Proposed Attack Implementation

In this section, the technology used during development and details on the programming implementation are discussed and defined. A defined description of the technology used and the implementation details will be needed to fully understand the proposed new attack implementation.

4.3.1 Technology

There are many different technologies that can be used to implement this DoS attack. Java was chosen as the language to develop an implementation of this new attack and also used to create the data caching network simulator. Java presented data types that made using three different types of caching strategies easy to implement and code. Java also supported creation of a graph object with weighted edges. The Java Platform, Standard Edition (Java SE) Java Development Kit (JDK) version 8u25 was used, which was the latest version at the time of the development period.

IntelliJ IDEA community edition was chosen as the Java integrated development environment (IDE). IntelliJ also offers an enterprise edition of the IDE at a cost, but the free community edition was used for this thesis. The latest version of Java JDK was supported in the community edition. IntelliJ IDEA offered an environment for easy and fast Java development, compiling, and execution. GIT was used as the revision or source control system. IntelliJ IDEA offered free integration with GIT which made source control very simple. The latest version of GIT release code, version 2.2.1, was used at the time of development.

The development was done in a Microsoft Windows environment. All executables for IntelliJ, Java, and GIT were all run and tested on Windows 7 and Windows 8. Development of the source code was completed with IntelliJ and completed in weekly sprints. For testing, the simulator was run on the following computers described in Table 1.

Specifications	Machine 1	Machine 2
Operating System	Windows 7 x64	Windows 8 x64
CPU	AMD Phenom II X4 940 (3 GHz)	Intel Core i7 870 (2.93 GHz)
Memory	8 GB	8 GB
Hard Drive	256 GB	256 GB
Hard Drive Type	SSD	SSD

Table 1: Development Machine Information

4.3.2 Java ICN Simulator & Attack Implementation

Java was used as the platform to build an initial implementation of this proposed DoS attack on data caching networks. Java is an object oriented programming language, and many objects were used to create a network simulation with nodes and edges for evaluation. Basic classes were created including a node class, an attacker node class (extension of node), an edge class, a content class, and a graph class (collection of nodes and edges).

To create the network in Java, first a new graph was created. In the create graph method, nodes, edges, and content were created according to the input variables. As an example, a square graph was created that contained 25 nodes. Since it was a square graph, nodes were placed in a matrix pattern with 5 nodes wide and 5 nodes high. Edges were then created to connect every node to its neighbors (matrix) to the top, bottom, left, or right of the node. Diagonal edges were not created in the square graph matrix. Edges were created with a weight to supported weighted graphs and were created as directional edges. In order to connect node 1 to node 2 bi-directionally, an edge would need to be created from node 1 to node 2 and then also created from node 2 to node 1.

After creating all the nodes and edges on the graph, content custodians were chosen at random and assigned content using an equal distribution. In the 25 node matrix, 20% or 5 nodes were chosen at random locations and assigned as custodians. Changing the value of the number of custodians will change the results, but 20% was arbitrarily chosen as a constant. The node then stored the content locally on the node and thus was the custodian.

After all custodians were created, all other nodes on the network were updated to store the location of each piece of content. Each node stored a hash table of the content custodians which included a reference to the content object and a corresponding reference to the node object where it could be found.

All content on the network was then assigned a popularity value according to the Zipfian distribution. The Zipfian distribution assigned each content object a popularity value based on the value of alpha given as an argument. Two primary alpha values were tested during the evaluation phase, 0.65, and 0.85. These values were chosen as they were common values tested when implementing the Zipfian distribution. Changing the value of the Zipfian alpha will change the results, but 20% was arbitrarily chosen as a constant. The alpha would be used to distribute the values of the content popularity, which would assign them in ascending order. To appropriately distribute content randomly among the custodians, the values were shuffled and then assigned to each content object. After this was completed, the graph was fully created and ready to send and receive requests.

For each cache type and cache size, a series of tests were performed on the simulated network. The search class was used to perform all the work of processing the requests. The value of 100,000 was used as a constant in all evaluation testing for the number of request to perform for each test. The search class would create packet objects which would help the nodes route the request and store data about the request. The packet would start at the source node and perform Dijkstra's algorithm to determine the shortest path to the custodian that held the requested content item. That custodian would then become the

destination node and a path array was established. The packet would use the node's methods of send and receive data to route the request and check each nodes cache at every hop to see if the content could be served from the node's cache. If the content was found in the cache of a node in the path, then the request was served from that node and statistics of the completed request were stored in the packet class. If the content was not found in the cache of a node in the path, then that node would store that content in its own cache according to the caching policy being used on that network. The requested content object was selected according to the probability distribution of the content. More popular content was requested more frequently than less popular content. The source node, or requester, was selected at random for all nodes that were not custodians.

To ensure all nodes in the network had appropriate time to populate caches with content, also known as cache warming, no data were saved for the first 70% of the total test. Thus if the test called for 100 requests, only the statistics of the last 30 requests would be kept to allow for cache warming. After collecting the data from all packets, the data were stored in a packet tracer class to keep a history of all of the tests and statistics. This process was repeated until all cache types and cache sizes were fully tested according to the constant variables defined.

To ensure that a correct and valid evaluation was performed, a pattern was established during the first run of every unique cache type and cache size. The pattern consisted of the exact source, destination, and content of each request or packet. This method helped create a constant variable for when attacker node objects were added to the network. Having a

pattern allowed for the same tests and order of requests to be performed after adding attackers to the network. This ensured that the dependent variable, average number of hops, would be comparing the same sets of data while only changing the number of attacker node objects.

After testing a network with no attacker nodes, the same network structure was used. Random nodes were selected to add attackers to the network. The nodes were converted to an attacker node object that extends the node class. This allowed for the node methods to be overwritten to perform the attack. Once all attacker node objects were added to the network, Dijkstra's algorithm had to be recomputed on all nodes so every node would correctly find the shortest path to content. This ensured that all regular node objects would send requests for content without knowing the attacker node was malicious.

The attacker node object had three primary methods that made it different than a normal node: polling for content to determine popularity, sending the attack, and determining a good value for characteristic time. The warm up period was used by an attacker node to poll all content that it served. After polling for some time, the node would have a new popularity distribution of content based on what it had served on the network. This is one way the attacker node could know that sending a request for a given content item was indeed unpopular. The attack method was used when the attacker node was the source node of a request packet. The attacker would send its request for the content according to the pattern established. After that request was complete, the attacker node would then send X number of request packets for unpopular content. The value of X was kept as a constant

in the evaluation and was defined as the Attacker Request Rate. In order to prevent detection, small request rates were tested during the evaluation phase including the values of 1, 2 and 4. This meant that each time an attacker node was a source node in a packet, an additional 1, 2, or 4 packets would be created and sent over the network. By requesting unpopular content, popular content would be purged from the active cache stores of the node objects in the path to the unpopular file's custodian.

An improved attack method was also found and tested during the evaluation. This improvement was named the smart attack, which used the concept of Characteristic Time (T*) as described in section 4.2. When this attack was used, the node needed to calculate the value of Characteristic Time. The GuessCharacteristicTime method chose the most unpopular file, requested that file, waited a large amount of time (a multiple of the cache size value), re-requested the same file and saw if the custodian served the request. If the request was filled by the custodian, a smaller value was chosen as the guess and the process would repeat. This continued until the request was served by a node in the path (not the custodian), thus resulting in a cache hit. This final value was saved on the attacker node as the final characteristic time guess and used when determining which unpopular file to request during the attack. The attack method did not change, with the exception of only requesting an unpopular file that had not been requested in at least the characteristic time. The attacker node kept a hash table to store and update the unpopular file list during the attack.

All test results were stored and compared in a graph after all test runs were completed. An example of a completed test can be seen in Figure 7 below.

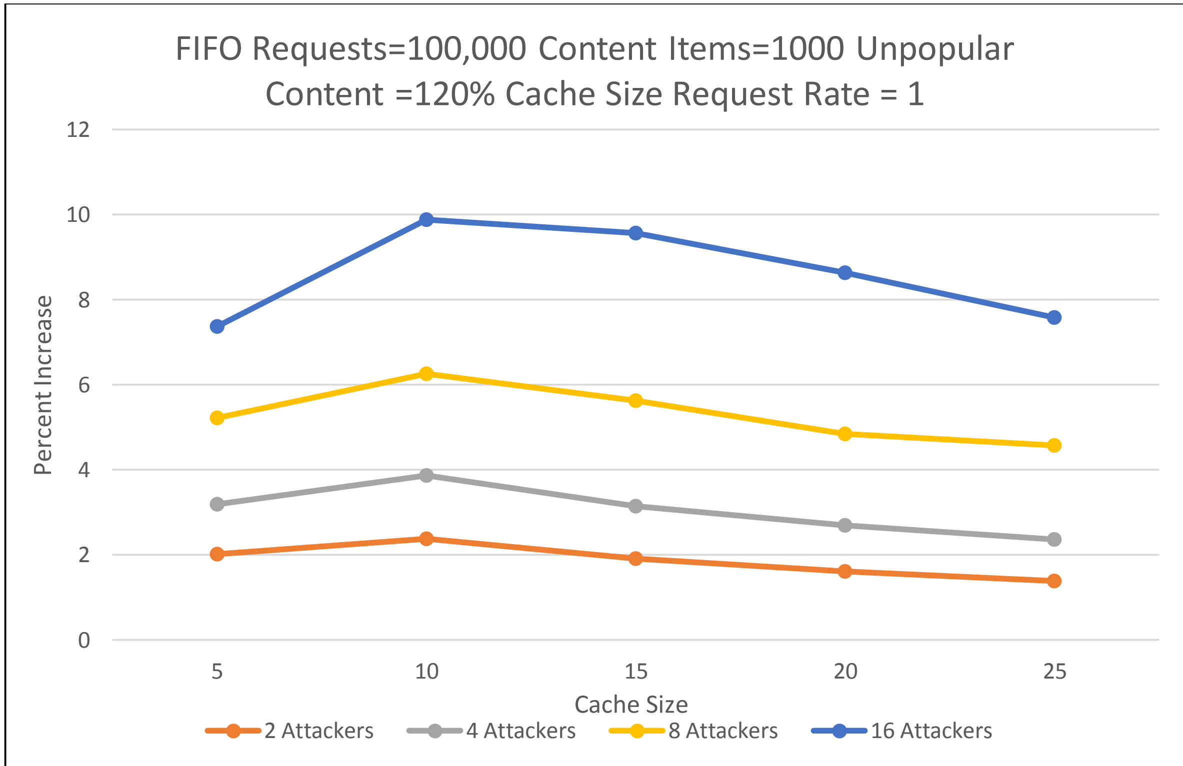


Figure 7: Java Simulator Test Results Example

Chapter 5

EVALUATION

5.1 Evaluation Objectives

The cache pollution attack described in the paper by Conti *et al.* [Conti13] was shown to produce positive attack results in a small network graph. This research was started to develop a specific implementation of the proposed cache pollution DoS attack. The simulator developed in Java was used to evaluate the implementation of the attack across multiple cache replacement policies (LRU, FIFO, Random), multiple cache sizes, and most importantly on multiple network sizes. The simulator was used to evaluate the average number of hops for each unique cache type and cache size on each of the three network scenarios. After a pattern was established on the first run of each unique cache type and cache size graph, attackers would be added to the network and the test would be run again with attackers. After running the complete test with and without attackers, the two values of average number of hops could be compared to calculate the percentage increase.

All other variables were created as constants including the number of tests run and the numbers of requests performed per test. The pattern established on the first test also remained the same as other tests were run with and without attackers. The location of the custodians and the content found on the custodians also remained unchanged for each test after the pattern was established. The number of unpopular files used in the attack was a

variable that was chosen after running tests with many different values. Since results were different depending on the number of unpopular files each attacker used, that number needed to remain constant. Statistics were used to determine which value was chosen as this variable. Table 2 below shows the results of those tests, in which different percentages of the currently tested cache size were used to determine the number of unpopular files.

Number of Unpopular files: Percent Cache Size (%)	Average Percent Increase (%)	Standard Deviation
80%	8.113225856	3.030205472
100%	10.11736493	2.80922496
120%	11.18972473	3.46343106
150%	8.792727194	2.491202425
200%	11.05813752	6.648555748

Table 2: Number of Unpopular Files Test

After evaluating the results of this test, 120% of the current cache size was chosen as the value to use in all tests. 120% of the current cache size means that the number of unpopular files used by each attacker changes as the cache size changes. For example, if the cache size being tested is 10, the number of unpopular files each attacker uses is set to 12 (120% of 10). Unpopular content items are those that have the lowest popularity value based on the Zipfian distribution that was established during the network graph creation. When a value of 12 was used as the number of unpopular items, the attacker would select the 12 content items with the lowest popularity value. This list could also be established by polling the network or by having the list of the entire content universe, but this was not how it was implemented in this simulator.

When evaluating the average hops taken to complete the requests in each test, only the requests that were served by a cache hit when the pattern was established were taken into consideration. This was done to ensure the results were truly evaluating the effectiveness of the attack on the data caches. If a request was served by the content custodian during the pattern simulation run, any attack on the data cache could not lengthen the hops taken to retrieve that piece of content. For example, if a node requested content “A” that was served by the custodian in say 7 hops, then that is the worst case scenario for the performance of that specific request. In this example, the content was served in a maximum of 7 hops due to the size of the network and how many nodes were between requesting node and the custodian node. The maximum number of hops will vary depending on the number of hops between a requester and a custodian. The data caching on the network for that request didn’t serve the request, and thus any attack on the data caching would be useless. For this reason, during the first run of a given test when the pattern is established, only the requests that are requested by a normal node (not an attacker) and served by a data cache (cache hit) are taken into consideration in the average number of hops in all tests. Any requests that are served by the custodian during the pattern simulation run are not included in the average hops calculation.

Another objective of this research was to develop a smarter cache pollution attack based on the previous research on characteristic time and caching. A smart attack was developed as described in Chapter 4, and was also evaluated against the normal cache pollution attack. To evaluate the smart attack, tests were run without the smart attack and then run with the

same graph structure with the smart attack. This ensured that the two different attacks were evaluated in a consistent manner.

5.1.1 Evaluation Limitations

While evaluating the large scale networks, memory limitations were found. All development and testing was completed on personal hardware that had at most 8GB of RAM. During the testing of the 25 and 100 node square graphs, all cache types, cache sizes, and number of attackers could be tested with a large loop function. The simulator allowed for looping through each specific test and the 100 node graph would use around 4-5GB of RAM after completing all tests. When testing the large scale Gnutella networks, tests needed to be looped one cache size at a time. Just looping with one cache type, one cache size, and one size of attackers would exceed 6-8GB of RAM. For this reason, batch scripts were created to loop through each specific test so all variable combinations could be tested. The amount of time taken to loop through this simulation should also be taken into consideration. When looping with the 6301 node Gnutella network with one cache type and one cache size, the test would take 40-60 minutes to complete using a machine with an Intel i7 quad core processor and DDR3 RAM. Running this same test on an AMD Phenom II quad core processor with DDR3 RAM took 90-120 minutes. This shows the simulator is very processor and memory intensive and hardware chosen to run testing should be considered carefully.

5.2 Experiment Scenarios

In order to fully evaluate the scale of the proposed DoS attack, three different scenarios were setup. The first experiment was a simple line graph setup with 5 nodes, including one requester, one custodian, and one attacker. The second scenario developed was a square graph that can also be seen as a matrix. The square graph had up to 100 nodes on the network, including 20 custodians (20%), and remaining nodes were requesters. Attackers were added to the graph and testing up to 16 attacker nodes (16%). The third scenario developed was the Gnutella peer-to-peer dataset from the Stanford SNAP database [Leskovec14]. Two different real world graphs were imported, including a 6301 node graph and an 8846 node graph. Each Gnutella graph included 5% custodians, and the remaining nodes were requesters. Up to 16% attackers were also added and tested on both of these networks.

5.2.1 Scenario 1: Line Graph

This evaluation scenario can be seen as the proof of concept for the targeted DoS attack on data caching networks. In this scenario, 5 nodes were created including one requester and one custodian. The requester was the first node in the line graph and the custodian was the last node in the graph when visualizing the graph from left to right. The other 3 nodes connected the requester to the custodian. Edges were set so the furthest left node would connect to the neighbor node to its right, and this would continue to the custodian. Only two content items were available on the custodian, one being the popular item and one

being the unpopular item. Each node was given a cache size of 1, and the requester always requested the popular file. An attacker node was added to the graph and its edge was connected to the second node in the graph, or the only node that the requester was connected to. This attacker node always requested the unpopular file and the requester always requested the popular file. Both nodes requested at the same rate, but that does not mean that the requests alternated as that would be the worst case scenario. Figure 8 shows a visual representation of this evaluation scenario.

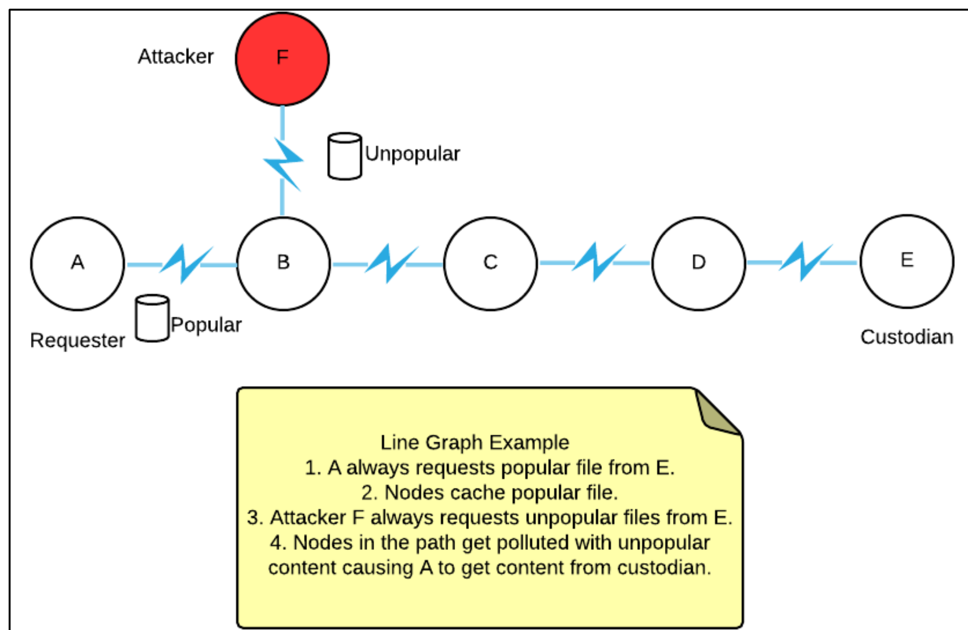


Figure 8: Line Graph

5.2.2 Scenario 2: Square Graph

This evaluation scenario was set up to test the effectiveness of the DoS attack on a small network topology. A simple matrix or square graph was chosen to implement this evaluation scenario. The square graph was tested with 25 and 100 total nodes on the

network, which tested a 5x5 graph and a 10x10 graph respectively. In this scenario, 20% of the total number of nodes were selected at random as the custodians and the rest of the nodes were selected as requesters. A total of 1000 content items were distributed equally to these custodians and popularity was randomized amongst all content as described in the implementation section in Chapter 4. Each node was connected to its neighbor to the top, bottom, left, or right of the node. If a node existed at one of those locations, an edge was created. This established a matrix square graph network that could be visualized as a grid as seen in Figure 9.

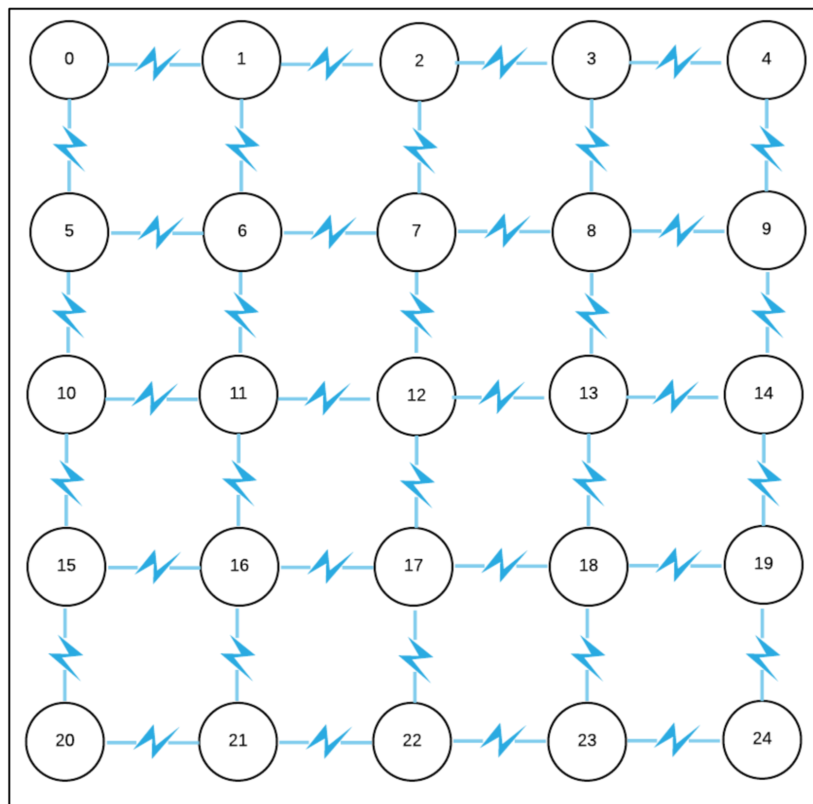


Figure 9: Square Graph

The following method was applied on scenario 2 and scenario 3 as mentioned earlier in Chapter 4. The first run of every test always started with 0 attackers, and requesters were selected at random. The node selected then requested content based on the Zipfian popularity distribution. Using a Zipfian popularity distribution and a constant alpha (values tested included: 0.65, 0.85) the requests were created and a pattern was established. When attackers were added to the graph, the same pattern was used as a constant variable. The pattern consisted of the source node, the destination node, and the content being searched for. The same pattern that was established with 0 attackers was used for every test after the first run to ensure correct statistics were being collected. Attackers were added to the graph, and when the pattern ran into an attacker node, the attacker would send the request for the popular content item and then send X number of attack requests (unpopular content) based on the constant variable for the attacker request rate.

5.2.3 Scenario 3: Real World (Gnutella) Graph

This evaluation scenario was set up to test the effectiveness of the DoS attack on a real world peer-to-peer network topology. The Stanford Large Network Dataset Collection was used to test two real world Gnutella peer-to-peer networks [Leskovec14]. From the dataset, a 6,301 node graph with 20,777 directed edges and an 8,846 node graph with 31,839 directed edges were chosen as test networks for this evaluation scenario. With this many nodes, 5% of the total number of nodes were selected at random as the custodians and the rest of the nodes were selected as requesters. A total of 2,000 content items were distributed equally to these custodians and popularity was randomized amongst all content

as described in the implementation section in Chapter 4. Each dataset from the SNAP dataset included a list of all edges in each network. This list of edges was imported from the downloaded file which created the unique network graph based on the Gnutella structure.

5.3 Evaluation Results

This research aims to analyze the impact of a DoS attack on data caching networks. Two independent sets of simulations (without attackers and with attackers) were run on the same graph to produce results shown in the dependent variable of average number of hops. As stated earlier, the average number of hops was only considered for normal requester nodes that returned a cache hit during the establishment of the pattern. This ensured that only the average number of hops resulting from a result of the attack were shown in the results.

For each scenario, the simulator measured the average number of hops for requests that when ran without attackers returned cache hits. Attackers were added to the graph using 2%, 4%, 8%, and 16% of the current graph size as attackers. Data collected on the line graph and the 25 node square graph were unable to perform tests for each of these percentages as they were too small in size. The line graph network results were collected for a proof of concept to show that the attack was valid. The 25 node square graph results show 4%, 8%, 16%, and 32% attackers since the minimum number of attackers on this small of a graph was 1 attacker or 4%. Data collected in each simulation are independent as results collected for one test do not interfere with results collected in another test.

5.3.1 Scenario 1 – Line Graph Results

In the line graph scenario, the requests from the normal node and the attacker node occurred at the same rate. This scenario aimed to show that the attack was possible and that a simple proof of concept line graph would show that the attack was valid. Each of the cache types of LRU, FIFO, and Random were tested in this scenario and both 1 and 2 attackers were also tested. Figure 10 shows that the average number of hops increased as the number of attackers on the graph increased from 1 to 2. Figure 11 shows that the percent increase of 109% for the average number of hops for normal requesters is very large when an attacker is on the network. As seen in Figure 10 and Figure 11, when the cache size is 0 all requests are served from the custodian which is 4 hops from the requester in this simulation. Also, when there is no attacker on the network and the cache size is 1, the neighbor to the requester always has the popular file in its cache and thus always returns the content in the first hop after the warm up period.

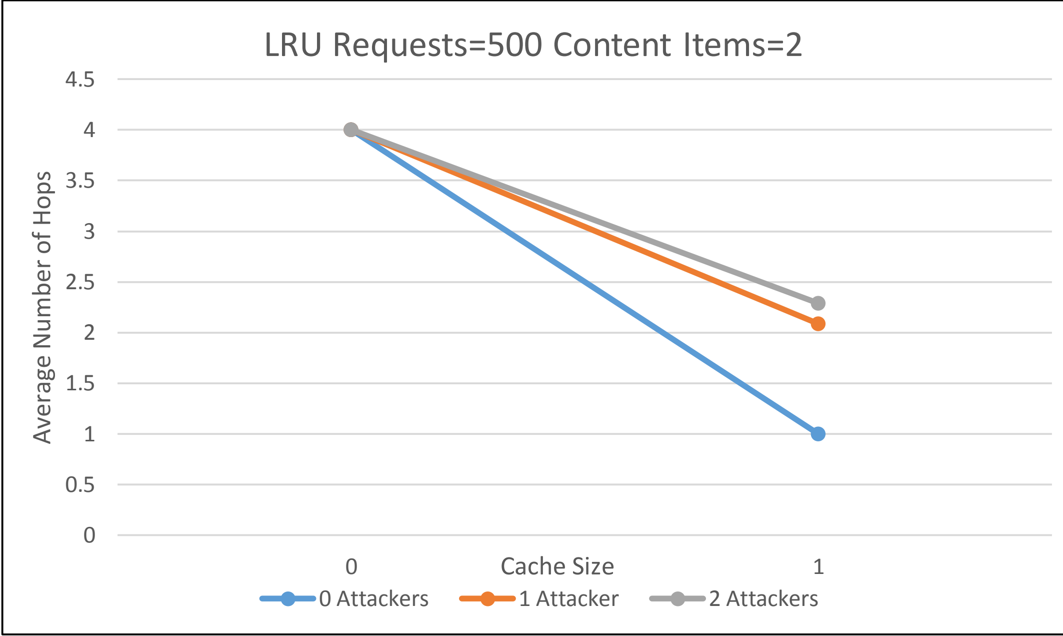


Figure 10: Line Graph Network Average Number of Hops

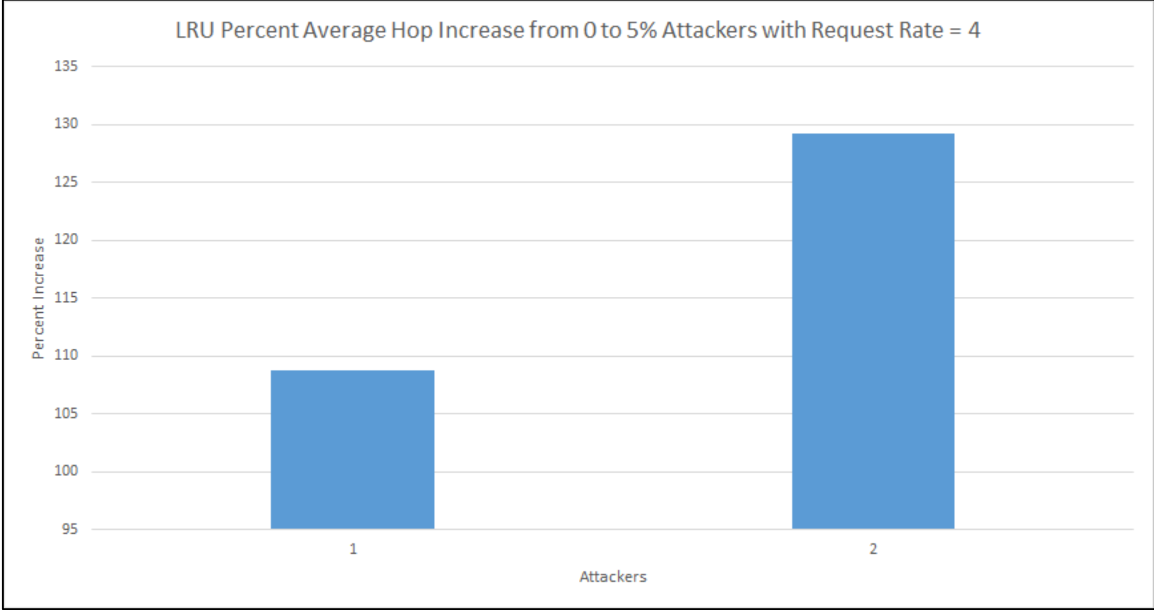


Figure 11: Line Graph Network Percent Increase

Table 3 shows that the attack was very effective during this line graph network scenario across all cache types. The results also show that as the number of attackers increased, the

average number of hops increased accordingly. This attack was also very effective due to the fact that only two pieces of content existed on the network. By defining that one content item was popular and the other was unpopular, the results showed that with limited number of content items on the network, the attack would be very effective. Three different cache replacement policies were tested in the line graph, and the results of this scenario did not show a difference in using one policy over another.

Cache Type	Mean: 1 Attacker	Std. Deviation: 1 Attacker	Mean: 2 Attackers	Std. Deviation: 2 Attackers
LRU	108.8	0	129.2	0
FIFO	111.2	0	116.4	0
Random	112	0	120.8	0
Grand Total	110.6666	1.3597	122.1333	5.3099

Table 3: Line Graph Percentage Increase Statistics

5.3.2 Scenario 2 – Square Graph Results

In the square graph scenario, a 25 node network and a 100 node network was tested which created a 5x5 and a 10x10 matrix network respectively. Each network was created using the same method, using the independent variables of 20% custodians, 10 tests, and 100,000 requests per test. The 25 node network was tested with 1 attacker (4%), 2 attackers (8%), 4 attackers (16%), and 8 attackers (32%). The 100 node network was tested with 2 attackers (2%), 4 attackers (4%), 8 attackers (8%), and 16 attackers (16%). Since the 25 node network was too small to test with 2% attackers, the best comparisons can be made using 4%, 8%, and 16% attackers. Each graph was also tested with two values for the Zipfian alpha (0.65 & 0.85) which changed the distribution of the content popularity.

Values of 1, 2, and 4 were used as the attacker request rate which defined how many unpopular files the attacker node would request for each attack request. All plotted charts can be found in Appendix B. Figure 12 shows an example of the percent increase found in the 25 node network using LRU caching strategy and an attacker request rate of 1. Figure 13 shows the percentage increase found in the 100 node network using LRU caching strategy and an attacker request rate of 1. As seen in the charts, the overall percentage increase was 2%-4% smaller as the size of the graph grew from 25 to 100 nodes. The results also show that the evaluation was performed with minimal error using LRU and FIFO cache replacement policies. The results in Appendix B show that using a random cache replacement policy increased the value of the error bars due to the randomness of the cache replacement policy.

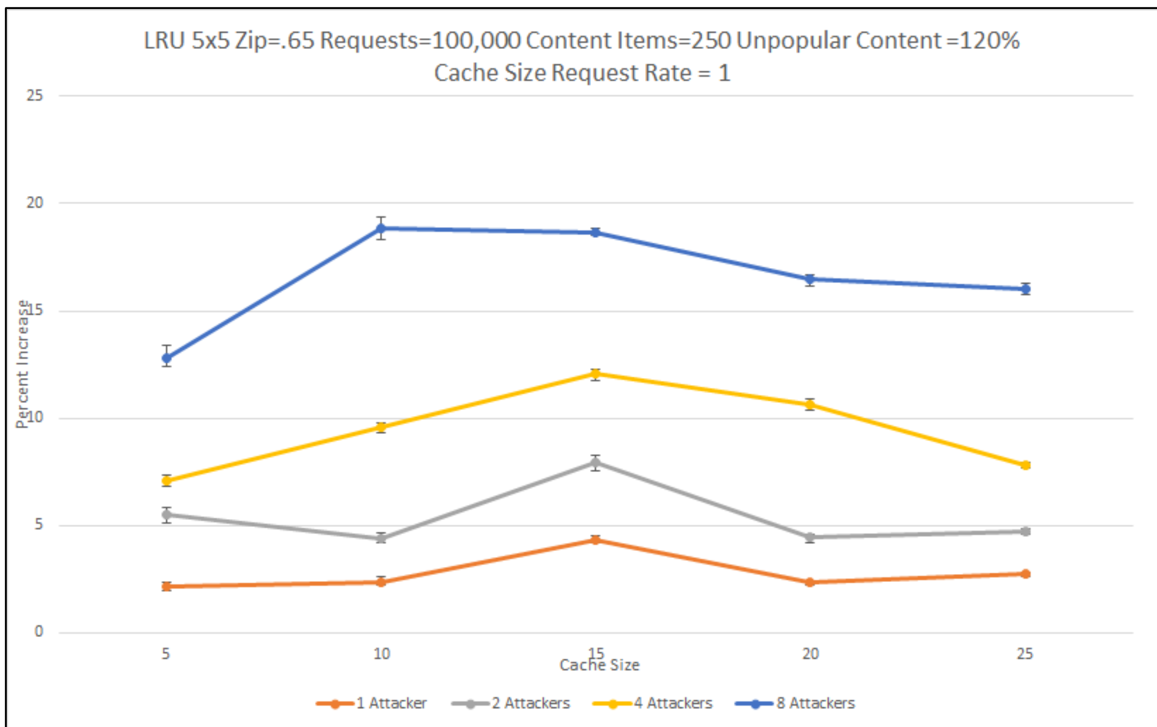


Figure 12: Square Graph - 25 Nodes, using LRU, and Attacker Request Rate of 1

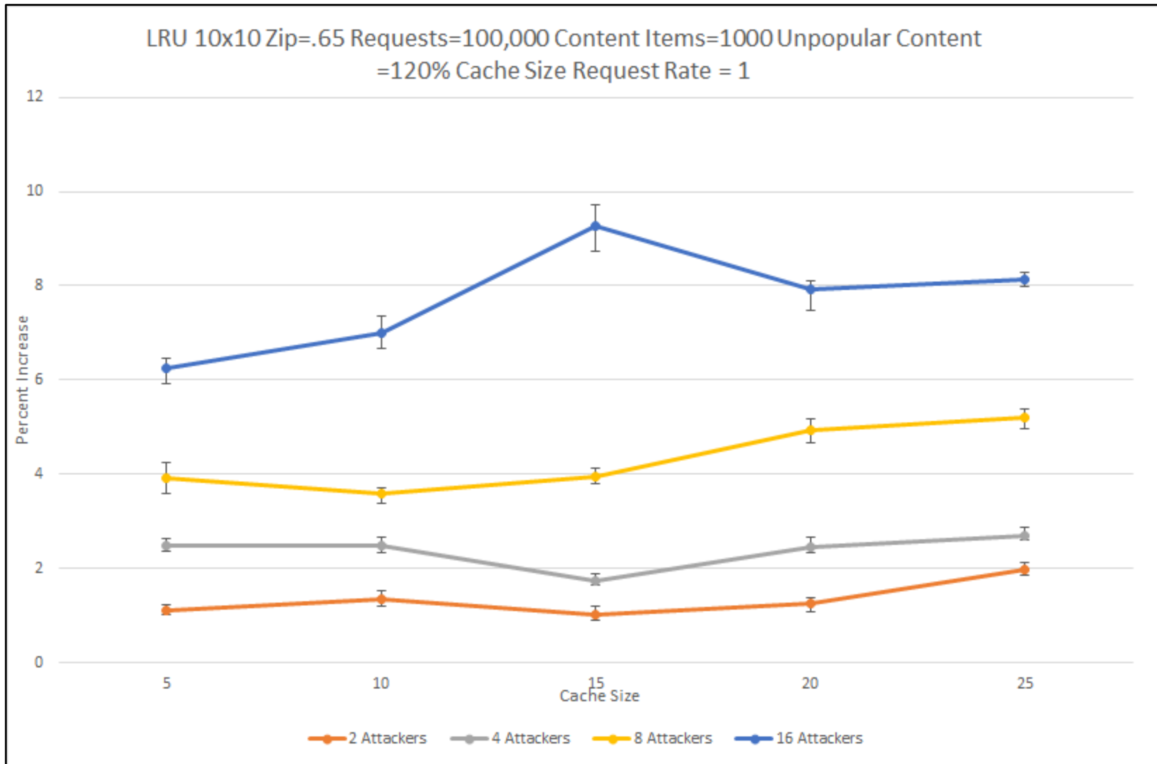


Figure 13: Square Graph - 100 Nodes, using LRU, and Attacker Request Rate of 1

The mean of all the percent increases collected was also calculated for each unique cache type, cache size, Zipfian alpha, and attacker request rate. The standard deviation of the entire population was also calculated for each of these unique combinations. All collected statistics can be found in Appendix B. Table 4 shows a combination of all stats collected on the 25 node square graph for 4%, 8%, 16%, and 32% attackers. Table 5 shows a combination of all stats collected on the 100 node square graph for 4%, 8%, 16%, and 32% attackers. All values in both tables are displayed as percentage increases calculated by the simulator using various numbers of attackers. Each table shows the results collected for the mean and standard deviation of LRU, FIFO, and random cache replacement policies using attacker request rates of 1, 2, and 4. The total rows shown at the end of each attacker request rate evaluated display the mean and the standard deviation of all the cache

replacement polices at the given attacker request rate. For example, the 1 total row and mean 4% attackers column shows the results of the average percentage increase across LRU, FIFO, and random cache replacement policies using an attacker request rate of 1 and 4% attackers. This value shows a good average that can be expected from the attack when using an attacker request rate of 1 and 4 attackers, regardless of what cache replacement policy is used. The grand total row is added to show the mean percentage increase of each evaluated number of attackers. The average percentage increase across all evaluated attacker request rates showed a decrease as the size of the network increased, which can be seen in the grand total row.

Rate	Cache Type	Mean: 4% Attackers	Std. Deviation : 4% Attackers	Mean: 8% Attackers	Std. Deviation : 8% Attackers	Mean: 16% Attackers	Std. Deviation : 16% Attackers	Mean: 32% Attackers	Std. Deviation : 32% Attackers
1	LRU	2.8021	0.8014	5.4001	1.3287	9.4280	1.8237	16.5540	2.1850
	FIFO	4.2650	1.5070	6.7171	1.4512	11.6098	1.4974	17.5756	0.8910
	Random	6.8465	2.2845	7.7570	2.5399	8.9012	2.2551	10.2906	2.2157
1 Total		4.6379	2.3466	6.6247	2.0907	9.9797	2.2195	14.8067	3.7235
2	LRU	4.6778	1.4546	7.9589	1.6079	13.5661	2.3762	21.9207	3.7898
	FIFO	4.9995	0.9143	8.2810	1.2684	15.2432	1.5159	23.0598	0.5477
	Random	4.6945	1.2789	6.3522	1.2567	8.0969	1.0437	11.8739	1.5567
2 Total		4.7906	1.2454	7.5307	1.6236	12.3021	3.5102	18.9515	5.5639
4	LRU	6.2265	1.6745	10.4373	1.0293	16.9176	2.3334	25.5914	1.8264
	FIFO	5.5726	2.1682	9.8396	2.8519	16.9229	2.2278	26.6797	2.1167
	Random	6.7405	1.7346	6.1768	2.5650	8.4563	3.0551	11.2954	2.7274
4 Total		6.1799	1.9321	8.8179	2.9672	14.0989	4.7434	21.1888	7.3636
Grand Total		5.2028	2.0194	7.6578	2.4658	12.1269	4.0116	18.3157	6.3250

Table 4: Square Graph 25 Node (Zipfian=0.65): Percentage Increase Statistics

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation : 2% Attackers	Mean: 4% Attackers	Std. Deviation : 4% Attackers	Mean: 8% Attackers	Std. Deviation : 8% Attackers	Mean: 16% Attackers	Std. Deviation : 16% Attackers
1	LRU	1.3405	0.3363	2.3684	0.3245	4.3202	0.6322	7.7166	1.0344
	FIFO	1.8617	0.3391	3.0553	0.5060	5.3064	0.5942	8.6093	1.0126
	Random	5.0120	2.1569	5.4562	2.2420	5.9421	2.2029	7.1760	2.1798
1 Total		2.7381	2.0634	3.6266	1.8837	5.1896	1.5211	7.8340	1.6222
2	LRU	1.9977	0.3909	3.6754	0.6713	6.3476	1.0592	10.2613	1.6947
	FIFO	1.8622	0.2527	3.3473	0.5085	6.0782	0.5963	9.7940	1.5589
	Random	5.2901	2.5830	6.0852	2.6980	7.0544	2.6648	9.0873	2.6381
2 Total		3.0500	2.1928	4.3693	2.0379	6.4934	1.7404	9.7142	2.0785
4	LRU	3.0606	0.6066	5.4929	0.8816	8.8349	0.5561	13.7887	0.9263
	FIFO	3.0641	0.7003	5.1621	0.6631	9.2712	0.4866	14.5649	0.9853
	Random	5.0844	1.7007	6.0817	1.9419	7.4762	1.8621	9.3821	1.7938
4 Total		3.7364	1.4693	5.5789	1.3443	8.5274	1.3864	12.5786	2.6252
Grand Total		3.1748	1.9787	4.5249	1.9537	6.7368	2.0756	10.0422	2.9017

Table 5: Square Graph 100 Node (Zipfian=0.65): Percentage Increase Statistics

5.3.3 Scenario 3 – Gnutella Graph Results

In the Gnutella graph scenario, a 6301 node network and an 8846 node network were tested. The graphs were imported from the Stanford SNAP dataset from the Internet peer-to-peer graphs [Leskovec14]. Each network was created using the same method, using the independent variables of 5% custodians, 10 tests, and 100,000 requests per test. Each network was tested with 2% attackers (126 & 176 nodes), 4% attackers (252 & 353 nodes),

8% attackers (504 & 707 nodes), and 16% attackers (1008 & 1415 nodes). Each graph was also tested with two values for the Zipfian alpha (0.65 & 0.85) which changed the distribution of the content popularity. Values of 1, 2, and 4 were used as the attacker request rate which defined how many unpopular files the attacker node would request for each attack request. All plotted charts can be found in Appendix B, and Figure 14 shows an example of the percent increase found in the 6301 node network using LRU caching strategy and an attacker request rate of 1. Figure 15 shows the percentage increase found in the 8846 node network using LRU caching strategy and an attacker request rate of 1. The results of the Gnutella scenario also show a small drop in percentage increase as the size of the graph grew from 6301 nodes to 8846 nodes. The results also show that the evaluation was performed with minimal error using LRU and FIFO cache replacement policies. The results in Appendix B show that using a random cache replacement policy increased the value of the error bars due to the randomness of the cache replacement policy.

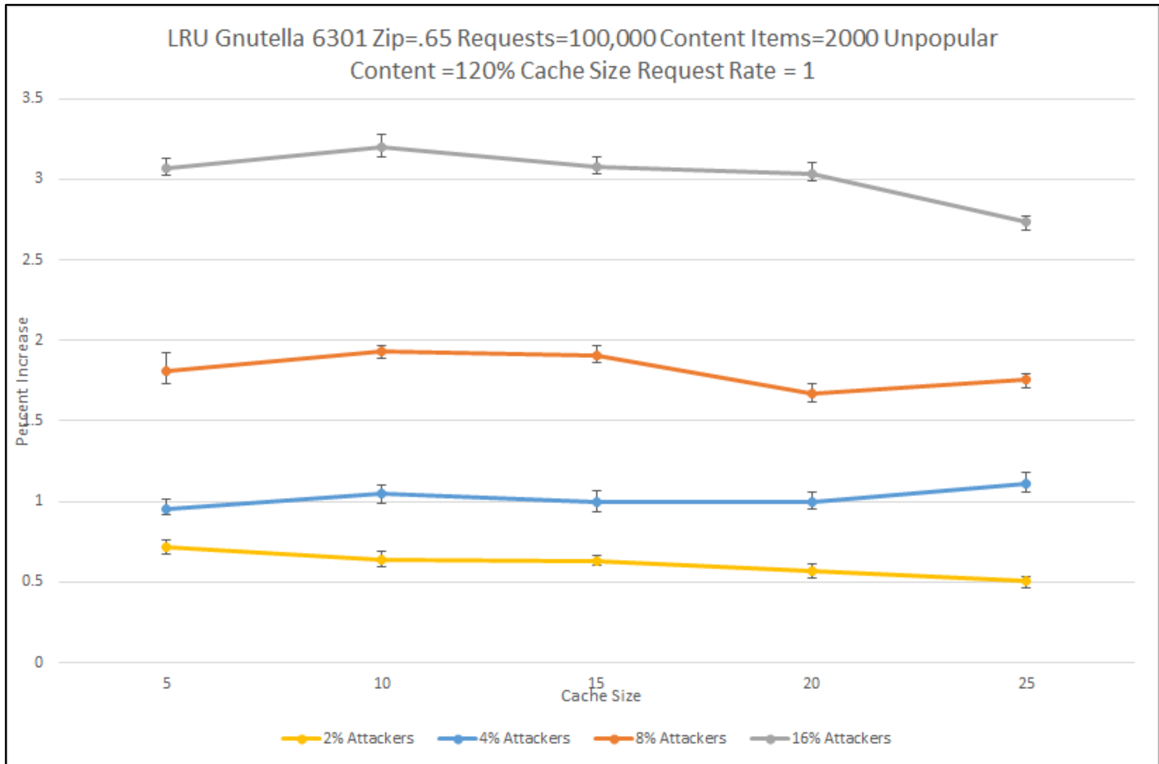


Figure 14: Gnutella Graph - 6301 Nodes, using LRU, and Attacker Request Rate of 1

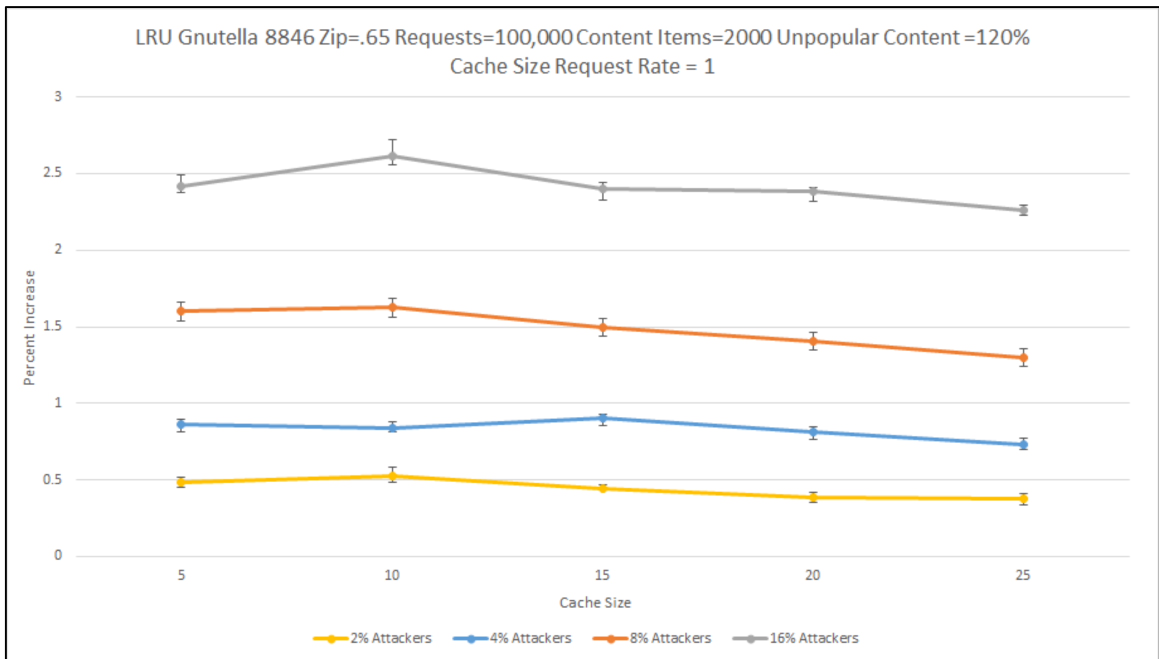


Figure 15: Gnutella Graph - 8846 Nodes, using LRU, and Attacker Request Rate of 1

Statistics were generated from all tests run on the two different size Gnutella graphs. When comparing the mean between the 6301 node graph and the 8846 node graph, a small decrease in the average percent increase can be seen. All collected statistics can be found in Appendix B. Table 6 shows a combination of all stats collected on the 6301 node Gnutella graph for 2%, 4%, 8%, and 16% attackers. Table 7 shows a combination of all stats collected on the 8846 node Gnutella graph for 2%, 4%, 8%, and 16% attackers. The total rows shown at the end of each attacker request rate evaluated display the mean and the standard deviation of all the cache replacement polices at the given attacker request rate. For example, the 1 total row and mean 4% attackers column shows the results of the average percentage increase across LRU, FIFO, and random cache replacement policies using an attacker request rate of 1 and 4% attackers. This value shows a good average that can be expected from the attack when using an attacker request rate of 1 and 4 attackers, regardless of what cache replacement policy is used. The grand total row is added to show the mean percentage increase of each evaluated number of attackers. The average percentage increase across all evaluated attacker request rates showed a decrease as the size of the network increased, which can be seen in the grand total row.

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation : 2% Attackers	Mean: 4% Attackers	Std. Deviation : 4% Attackers	Mean: 8% Attackers	Std. Deviation : 8% Attackers	Mean: 16% Attackers	Std. Deviation : 16% Attackers
1	LRU	0.6122	0.0709	1.0224	0.0515	1.8139	0.0952	3.0220	0.1516
	FIFO	1.1236	0.0972	1.7955	0.1299	2.7140	0.1696	4.1017	0.2510
	Random	0.8270	0.1139	1.1981	0.0688	1.7011	0.0354	2.3947	0.1116
1 Total		0.8543	0.2305	1.3387	0.3429	2.0764	0.4674	3.1728	0.7279
2	LRU	0.8594	0.0470	1.4177	0.0587	2.5109	0.0202	4.1999	0.1512
	FIFO	1.4053	0.1464	2.2980	0.1358	3.4862	0.2925	5.0724	0.2670
	Random	1.0516	0.0974	1.4507	0.0942	2.1109	0.0456	3.1348	0.1407
2 Total		1.1054	0.2493	1.7221	0.4198	2.7027	0.6025	4.1357	0.8159
4	LRU	1.1071	0.0698	2.0630	0.0467	3.4351	0.0580	5.3538	0.0681
	FIFO	1.7784	0.1684	2.8220	0.2614	4.2863	0.2365	6.1141	0.2380
	Random	1.2435	0.0855	1.7740	0.0985	2.5912	0.0823	3.8656	0.2549
4 Total		1.3763	0.3122	2.2197	0.4713	3.4375	0.7077	5.1112	0.9561
Grand Total		1.1120	0.3411	1.7602	0.5496	2.7389	0.8187	4.1399	1.1530

Table 6: Gnutella Graph 6301 Node (Zipfian=0.65): Percentage Increase Statistics

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation : 2% Attackers	Mean: 4% Attackers	Std. Deviation : 4% Attackers	Mean: 8% Attackers	Std. Deviation : 8% Attackers	Mean: 16% Attackers	Std. Deviation : 16% Attackers
1	LRU	0.4441	0.0575	0.8310	0.0572	1.4857	0.1225	2.4147	0.1155
	FIFO	0.8939	0.0685	1.4670	0.1234	2.2687	0.1196	3.5280	0.2095
	Random	0.7301	0.1098	1.0044	0.1551	1.4478	0.0629	2.1777	0.0742
1 Total		0.6894	0.2031	1.1008	0.2937	1.7341	0.3928	2.7068	0.6062
2	LRU	0.7130	0.0377	1.2734	0.0666	2.1661	0.1312	3.4845	0.1998
	FIFO	1.1960	0.0858	1.9668	0.1548	3.0387	0.1828	4.6194	0.2129
	Random	0.8487	0.0699	1.3241	0.0770	1.9098	0.0798	2.8567	0.1407
2 Total		0.9193	0.2143	1.5214	0.3332	2.3715	0.5025	3.6535	0.7531
4	LRU	0.9715	0.0785	1.6770	0.0781	2.8111	0.1438	4.5888	0.1307
	FIFO	1.4867	0.1110	2.4278	0.0921	3.6657	0.2104	5.4114	0.2733
	Random	1.0122	0.0696	1.5294	0.0583	2.3276	0.1183	3.4884	0.2909
4 Total		1.1568	0.2500	1.8781	0.4009	2.9348	0.5766	4.4962	0.8243
Grand Total		0.9218	0.2938	1.5001	0.4693	2.3468	0.6979	3.6189	1.0355

Table 7: Gnutella Graph 8846 Node (Zipfian=0.65): Percentage Increase Statistics

5.4 Discussion

After collecting all the results from all of the simulations, charts were created and are shown in Appendix B. The main contributions of this research were accomplished by creating an extensible ICN simulator, creating and implementing a DoS attack on data caching networks, and testing the scalability of the attack implementation. All charts

collected in the scalability testing showed a clear fall in the impact of the attack as the size of the network increased.

After analyzing all of the results collected, a few trends were found. The first trend that can be seen from the results is that the implemented cache pollution attack does not scale as Conti *et al.* discussed in their paper [Conti13]. As the size of the graph increased, the impact of the attack decreased. This can be seen in the steady decrease in values for in average percentage increase. This was seen clearly in the charts and shown to be true in the statistics collected for each set of graphs and shown in Tables 4-7 in the grand total row. The largest percent increases were seen on the smaller square graphs, which were similar to the graphs shown in previous research [Conti13].

One trend that was shown to remain true in the results was that as the number of attackers increased, the effect of the attack also increased. This was seen in the increasing values of the percentage increase as the number of attackers increased for a given graph. The results also showed a larger impact from the attack as the attacker request rate increased. This was expected as most DoS attacks show larger effects as the number of attackers increase in theory.

Another trend that was suggested by the results was that FIFO showed to be the consistently most affected caching strategy for the attack. Comparing the percent increase to LRU showed larger percent increases as seen in the 25 node square graph with an attacker request rate of 1. It was expected that LRU would be the most resilient caching strategy

for this data caching DoS attack since a main goal of LRU is to filter out unpopular content by prioritizing the most used content in the cache.

5.5 Future Work

As shown in Chapter 4, an initial version of the proposed smart attack was created and implemented in this research. A software based content centric network simulator was also developed to evaluate the proposed implementation. Initial results on the smaller graphs showed that this proposed improvement on the attack was inconsistent. The theory of the proposed smart attack should prove to be an improvement on the initial attack. One future improvement of this implementation of the DoS attack on data caching networks would be to improve the logic of the smart attack using the concept of characteristic time (T*).

Extending this research to evaluate the attack using hardware testing instead of the software simulator would also qualify as an extension of this research. This type of evaluation would require a dedicated lab of computers that each would represent a node or a collection of nodes. Connecting each of the computers to an internal network and building a new test would create a physical ICN in which testing of the attack could be performed. It would be recommended to ensure this network was disconnected from all other networks, including the Internet, to ensure the attack was limited to lab computers only.

The ICN simulator was built using custom Java objects that can be reused for many different types of data collection. Three optional variables that are not calculated or used

for extensive testing are edge delay, multiple values for weighted edges, and power consumption on nodes. The edge delay could be collected by making some small source code changes to the variables collected. The edge weight was set to a default value of 1 in the simulator, and thus made all edges un-weighted. Power consumption is another variable that a node could be assigned if the node is mobile. Performing analysis on new variables such as these would be a very good extension of this research.

Another subject that could be used for future work on this research is to apply this attack and test it on host to host networks. A node on a host to host network could turn into an attacker and then actively or passively poll web servers or other computers for content. This would allow the attacker to define a list of popular and unpopular content. A web crawler could be used on a web server to search a website for old or archived data and add that content to the list of unpopular content. Just sending out a large amount of requests for unpopular content would, in theory, fill up the web server's cache with unpopular content and begin removing popular content from the cache. This would have a similar effect on normal requests as they would no longer be able to retrieve content from the server's cache and thus performance would be affected. With enough attackers, it is possible that this attack could overload the server's hardware resources or database queries and cause the service to shutdown preventing all users from accessing the server.

Chapter 6

CONCLUSION

This research has provided an initial implementation of a DoS attack on data caching networks, or what is known as a cache pollution attack. The implementation of the ICN simulator and attack was developed using Java. The attack was evaluated for scalability by testing with 3 different types of graphs (line graph, square graph, Gnutella graph). The attack showed consistent improvement in performance in terms of percentage increase in average number of hops on cache hits. The results also showed that the attack does not scale as well as was initially thought in previous research. As the size of the graph increased, the average percent increase in average hops decreased. The greatest impact of the DoS attack was found on the line graph and on the 25 node square graph, which were the two smallest graphs in terms of number of nodes evaluated. The findings of this research contribute to the overall security research field by providing a description of a new attack that can be further researched.

We have identified three areas that can be considered for future research work. The improvement of the proposed smart attack using the concept of characteristic time (T*) could improve the initial version of the attack. Any logic that can be added to the attack should yield performance improvements on the attack algorithm and its effects. Creating and testing the attack on a physical ICN in a lab environment would allow for evaluation of the attack on physical hardware. This would be the next step in moving this type of

attack into a real world environment or test network. Expanding the current implementation of the Java simulator to include new variables and metrics such as delay, edge weights, and battery life is another option for extending this research. Evaluating the proposed DoS attack on host to host networks could be another area of future work on this research.

REFERENCES

Print Publications:

[Androutsellis-Theotokis04]
Androutsellis-Theotokis, S., amd D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies." ACM Computing Surveys, *36*(4), (2004), pp. 335-371.

[Brito13]
Brito, G. M. d., I. M. Moraes, and P. B. Velloso, "Information Centric Networks: A New Paradigm for the Internet." John Wiley & Sons, 2013.

[Bžoch12]
Bžoch, P., L. Matejka, L. Pešicka, and J. Šafarík, "Design and Implementation of a Caching Algorithm Applicable to Mobile Clients." Informatica, 36, 4, (2012), pp. 369-378.

[Chandra10]
Chandra, J., S. K. Shaw, and N. Ganguly, "HPC5: An Efficient Topology Generation Mechanism for Gnutella Networks." Computer Networks, 54, 9, (2010), pp. 1440-1459.

[Coffman02]
Coffman, K. G., and A. M. Odlyzko, "Internet Growth: Is there a “Moore’s Law” for data traffic?" Handbook of massive data sets, Springer US, 2002, pp. 47-93.

[Conti13]
Conti, M., P. Gasti, and M. Teoli, "A Lightweight Mechanism for Detection of Cache Pollution Attacks in Named Data Networking." Computer Networks, 57, 16, (2013), pp. 3178-3191.

[de Morais Cordeiro02]
de Morais Cordeiro, C., and D. P. Agrawal, "Mobile Ad Hoc Networking." Center for Distributed and Mobile Computing, ECECS, University of Cincinnati, Ohio, 2002.

[Dehghan13]
Dehghan, M., D. L. Goeckel, T. He, and D. Towsley, "Inferring Military Activity in Hybrid Networks Through Cache Behavior." Military Communications Conference, MILCOM 2013-2013 IEEE, (2013), pp. 1726-1731.

[Hernandez09]
Hernandez, S., Official (ISC) 2 guide to the CISSP CBK, CRC Press, 2009.

[Hevner04]
Hevner, A. R., S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research." MIS Quarterly, 28, 1, (2004), pp. 75-105.

[Pouwelse05]
Pouwelse, J., P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P File-Sharing System: Measurements and Analysis." Peer-to-peer systems IV, (2005), pp. 205-216.

[Saroiu02]
Saroiu, S., K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An Analysis of Internet Content Delivery Systems." Publications of the ACM, (2002).

[Schaller97]
Schaller, Robert R., "Moore's law: past, present and future." Spectrum, IEEE 34.6 (1997), pp. 52-59.

[Stock13]
Stock A., J. William, D. Wichers, OWASP top 10, OWASP Foundation, 2013.

[Tyson13]
Tyson, G., N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe, "A Survey of Mobility in Information-Centric Networks." Communications of the ACM, 2013.

[Xie12]
Xie, M., I. Widjaja, H. Wang, "Enhancing Cache Robustness for Content-Centric Networks." INFOCOM, 2012.

Electronic Sources:

[Daya13]
Daya, B. "Network Security: History, Importance, and Future", University of Florida Department of Electrical and Computer Engineering, http://web.mit.edu/~bdaya/www/Network%20Security.pdf, last revision 2013, last accessed 2013.

[Leskovec14]
Leskovec, J. and A. Krevl, "Stanford network analysis project (SNAP)", http://snap.stanford.edu, last revision June 2014, last accessed June 19, 2015.

[Stock13]
Stock A., J. William, and D. Wichers, "OWASP top 10", OWASP Foundation, https://www.owasp.org/index.php/Top_10_2013-Top_10, last revision August 2014, last accessed February 2015.

APPENDIX A

Source Code Examples

Git respository URL: https://github.com/gougej88/ICNDataCachingSimulator

Line Graph repository: https://github.com/gougej88/LineChartThesis

Node Class Example

```
public Packet receiveData(Packet p)
{
    //If this node is the dest
    if(p.dest.nodeID == this.nodeID)
    {
        p.hops++;
        p.referrer = this;
        UUID s = p.search.contentID;
        int f = content.indexOf(p.search);
        p.data = content.get(f);
        p.found=true;
        powerDrain(1);
        return p;
    }
    //Store content in cache then send along path if not on this node
    //Check in cache, if so sendData
    if(p.cacheEnabled && (searchCache(p.search.contentID) && !p.found))
    {
        //content found in cache send back to src
        p.found=true;
        //LRU cache move to front of hashmap since used
        if(cacheType==1) {
            cache.remove(p.search.contentID);
            addToCache(p.search);
        }
        p.hops++;
        p.referrer = this;
        p.data = cache.get(p.search.contentID);
        powerDrain(1);
        //System.out.println("Content Found in Cache!!!");
        p.cachehit = true;
        return p;
    }else{
        //Not found in cache, add to cache and forward to next hop
        //DISABLE/ENABLE CACHE HERE
        if(p.cacheEnabled)
            addToCache(p.search);
        p.hops++;
        powerDrain(1);
        return p;
    }
}
```

```
public Packet sendData(Packet p)
{
    //Send content to next route
    p = p.next.receiveData(p);
    return p;
}

public boolean searchCache(UUID contentID)
    {
    if(cache.containsKey(contentID))
    {
        return true;
    }else{
        return false;
    }
}

public void addToCache(Content x)
{
    //Add content to cache when new content is received
    if(cacheType==1 || cacheType==2)
        cache.put(x.contentID,x);
    if(cacheType==3)
        //if random cache has room. just add it
        if(cache.size() < cacheSize) {

            cache.put(x.contentID, x);
        }else{
            Random rnd = new Random(System.currentTimeMillis());
            Object[] values = cache.values().toArray();
            Object randomValue = values[rnd.nextInt(cacheSize)];
            Content c = (Content)randomValue;
            cache.remove(c.contentID);
            cache.put(x.contentID,x);
        }
}
```

Edge Class Example

```
public class Edge {
    public final Node target;
    public final double weight;
    public Edge(Node argTarget, double argWeight)
    { target = argTarget; weight = argWeight; }
}
```

Graph Class Example

```
public void createGraph(){

    dijkstraComputed = false;
    if(graphType == 1){
        length = (int)Math.sqrt(size);
        width = length;
    }

    int attackerindex = -1;
    Random rand = new Random(size);
    //Assign random requester as attacker
    if(numAttackers >0)  {
        for(int a = 0; a < numAttackers; a++){
        attackerindex = rand.nextInt(size);
            if(attackIndexes.contains(attackerindex))
            {
                a--;
            }else {
                attackIndexes.add(attackerindex);
            }
        }//end for
    }//end if

    for(int i = 0; i < size; i++)
    {
        if(attackIndexes.contains(i)) {
            AttackerNode att = new AttackerNode(i, cacheSize, cacheType, numUnpopularItemsPerAttacker,numRequestsPerTest);
            attackers.add(att);
            nodes.add(i, att);
        }else {
            Node a = new Node(i, cacheSize, cacheType);
            nodes.add(i, a);
        }//end else

    }//end for

    //Setup the rest of the graph
    //Create all edges and weights
    if(graphType==1)  {
        setEdgesSquareGraph();
    }
    if(graphType==2){
        setEdges(graphType);
    }
    if(graphType==3){
        setEdges(graphType);
    }
    //Make 20% of the nodes content custodians and assign content to each one
    createContentCustodians(percentCustodians);

//Pass a hashtable of all content and their respective custodian to all nodes
//All nodes must know where content lives in order to route correctly
distributeContentCustodians();

//Use Distribution to assign popularity to each piece of content in the graph
assignPopularityDistribution(alpha);
```

Content Class Example

```
public class Content {

    UUID contentID;
    String content;
    double probability;

    public Content()
    {


    }

    public Content getContent(UUID contentID) { return this; }

    public void addContent(UUID contentID, String stuff)
    {
        this.contentID = contentID;
        this.content = stuff;
    }


    public String showContent(int ContentID)
    {
        //Show the content stored on the node
        return content;
    }
}
```

Packet Class Example

```
public class Packet {
    Integer packetID;
    Node src;
    Node dest;
    Node referrer;
    Node next;
    Content search;
    Content data;
    Integer hops;
    Boolean cacheEnabled;
    Boolean cachehit;
    Boolean found;
    List<Node> route;
    double time;

    public Packet(Node s, Content k){
        this.src = s;
        this.search = k;
        this.hops = 0;
        this.cachehit = false;
        this.found = false;
        this.dest = s.contentCustodians.get(k);
        this.route = Dijkstra.getShortestPath(src,dest);

    }

}
```

AttackerNode Class Example

```
public Packet sendDataAttack(Packet p)
{
    //Check if done polling, if so time to attack or guess characteristic time
    //If not just send content along as usual
    if(donePolling && p.src == this) {
          //if not ready to attack then still need to guess characteristic time
            if(!readyToAttack) {

                p = GuessCharacteristicTime(target,cacheSizeGuess, p);
            }//end if not ready to attack
            //else ready to attack send attack request
            else {
                p = sendAttack(p);

            }//end else

        }//end if done polling

    else {
        //Send content to next route
        p = p.next.receiveData(p);
        }//end else

    return p;
}
```

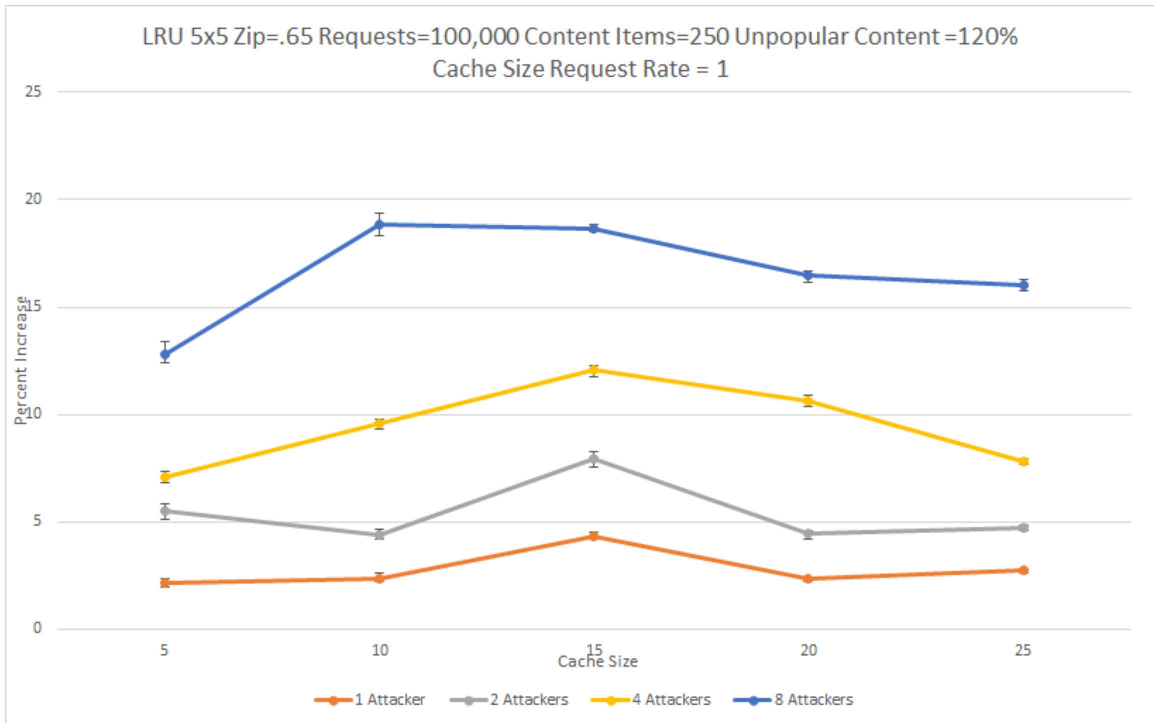
```java
public Packet GuessCharacteristicTime(Node target, int CacheSizeGuess, Packet p) {

    //Need to know where in the process we are
    //Possible values characteristicsTimeStatus
    //1 - request unpopular content first run
    //2 - waiting phase between requesting lists
    //3 - request unpopular content second run
    //4 - ready to decide on value
    if(characteristicTimeStatus == 2) {
        startWait++;
        //if number of requests seen more than characteristic time guess, then time to request again
        if(startWait >= characteristicTimeGuess){
            characteristicTimeStatus = 3;
        }

        //Send content to next route
        p = p.next.receiveData(p);

    }//end if waiting phase
    else {
        if (characteristicTimeStatus == 1 || characteristicTimeStatus == 3) {
            //Alter the request and request an unpopular file
            p.search = unpopularContent.get(characteristicTimeUnpopCounter);
            p.dest = this.contentCustodians.get(p.search);
            p.route = Dijkstra.getShortestPath(this, p.dest);
            p.next = p.route.get(1);

            p = p.next.receiveData(p);
        }

        //start waiting or set ready to attack
            //if done with phase 1, start waiting
            if (characteristicTimeStatus == 1) {
                startWait = 0;
                characteristicTimeStatus = 2;
            }
            //Else done guessing characteristic time
            if(characteristicTimeStatus == 3) {
                //ready to attack?
                //all from custodian check done outside of class in search class

                startWait = 0;
                characteristicTimeStatus = 4;
            }//end if
```

```
            if(characteristicTimeStatus == 4){
                //if all requests returned from custodian, then reduce T* guess
                if (allPacketsFromCustodian) {
                    characteristicTimeGuess = characteristicTimeGuess / 2;
                    characteristicTimeStatus = 1;

                    if (characteristicTimeUnpopCounter < (unpopularContent.size() - 1)) {
                        characteristicTimeUnpopCounter++;
                    } else {
                        characteristicTimeUnpopCounter = 0;
                    }//end else for increment
                }//end if all packets from custodian
                else {
                    finalCharTimeGuess = characteristicTimeGuess + 2;
                    characteristicTimeStatus = 1;
                    readyToAttack = true;
                }//end else

            }//end if

    }//end else for not in waiting phase
    return p;
}//end guessCharacteristicTime
```

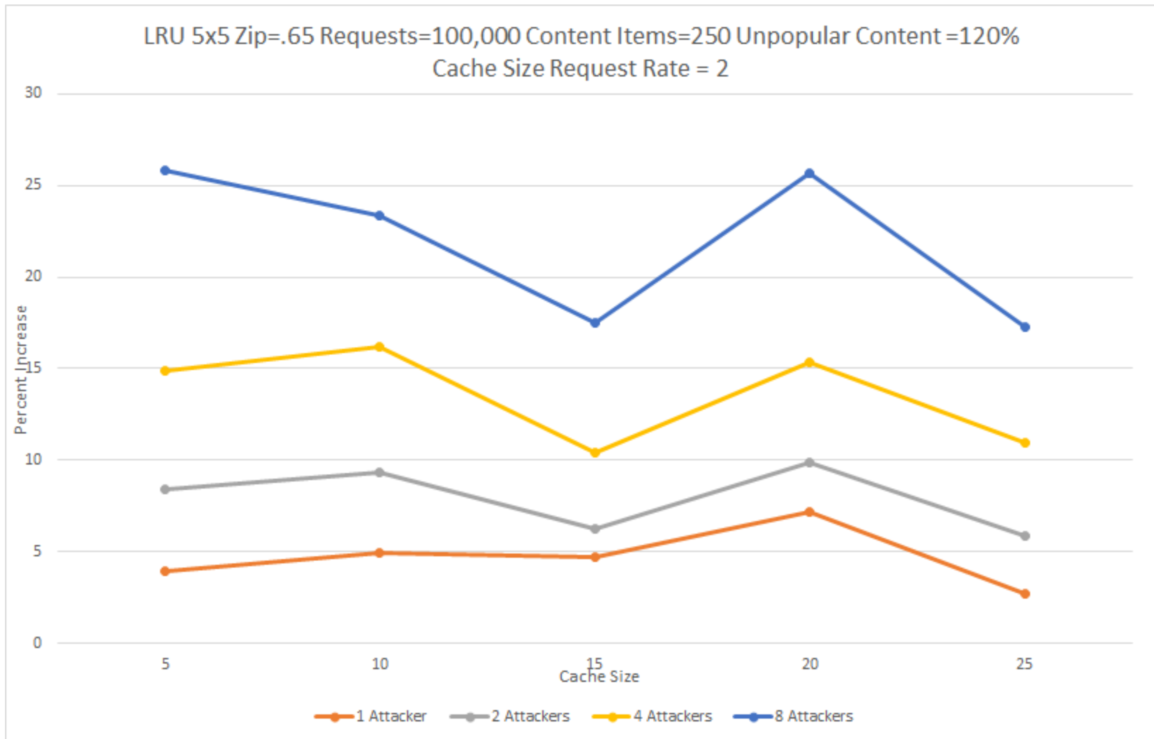
```java
public Packet sendAttack(Packet pack){
        numattacks++;
        Boolean foundItem = false;
        int index = 0;
        Content c = new Content();

        //Random file taken from list
        //Need to make sure the file has waited at least T*
        Random rand = new Random();
        while (!foundItem){
            index = rand.nextInt(unpopularContent.size());
            c = unpopularContent.get(index);
            if (attackList.containsKey(c)) {
                int waitedTime = attackList.get(c);
                if (waitedTime > finalCharTimeGuess) {
                    //Set content item back to zero since it is now being requested.
                    attackList.put(c,0);
                    foundItem = true;
                }
            } else {
                attackList.put(c, 0);
                foundItem=true;
            }
        }//end while

            //Alter the request and request an unpopular file
            pack.search = c;
            pack.dest = this.contentCustodians.get(pack.search);
            pack.route = Dijkstra.getShortestPath(this, pack.dest);

            //fix to ignore the nodes that cannot route to dest
            while(pack.route.size() ==1)
            {
                pack.search = unpopularContent.get(rand.nextInt(unpopularContent.size()));
                pack.dest = this.contentCustodians.get(pack.search);
                pack.route = Dijkstra.getShortestPath(this, pack.dest);
            }
            pack.next = pack.route.get(1);

            pack = pack.next.receiveData(pack);

    //Increment all attackItems by 1
    for(Map.Entry<Content, Integer> entry : attackList.entrySet())
    {
        Content k = entry.getKey();
        Integer i = entry.getValue();
        attackList.put(k,i+1);
    }

    return pack;

}//end sendAttack()
```

APPENDIX B

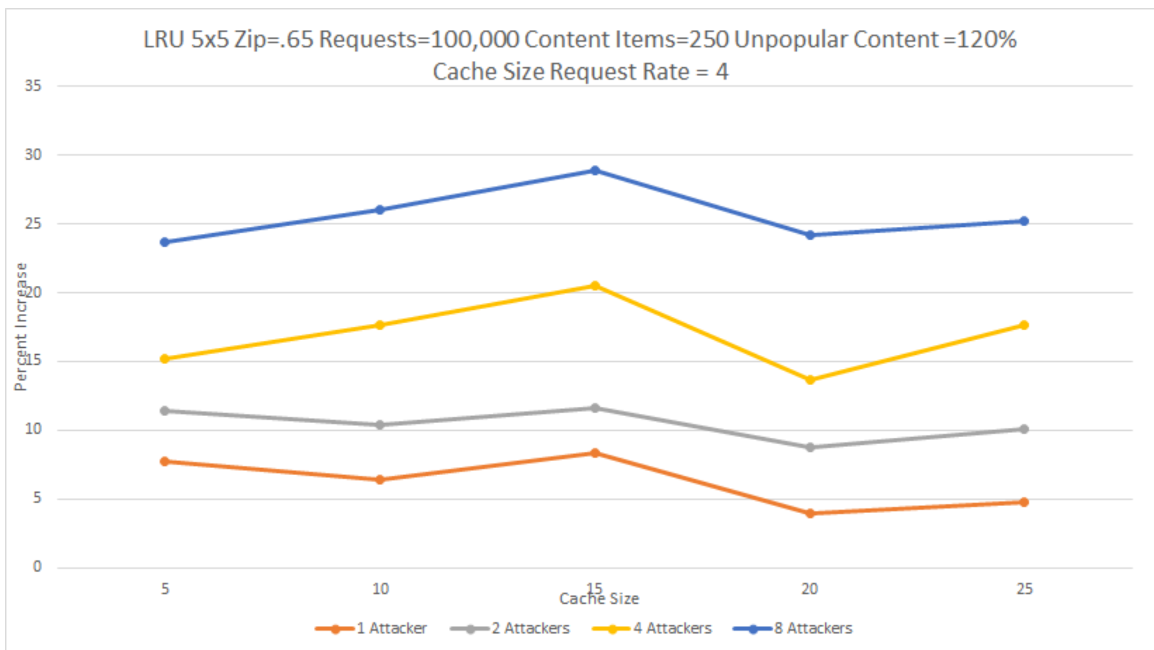
Results Collected



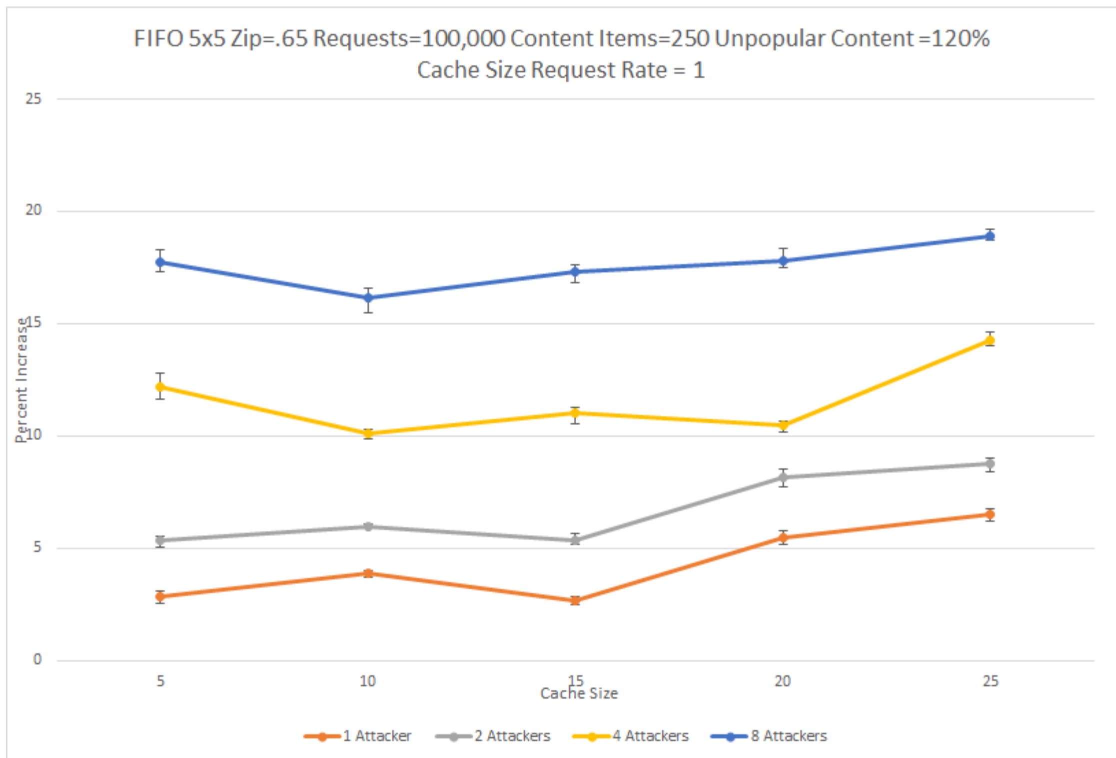
5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 1



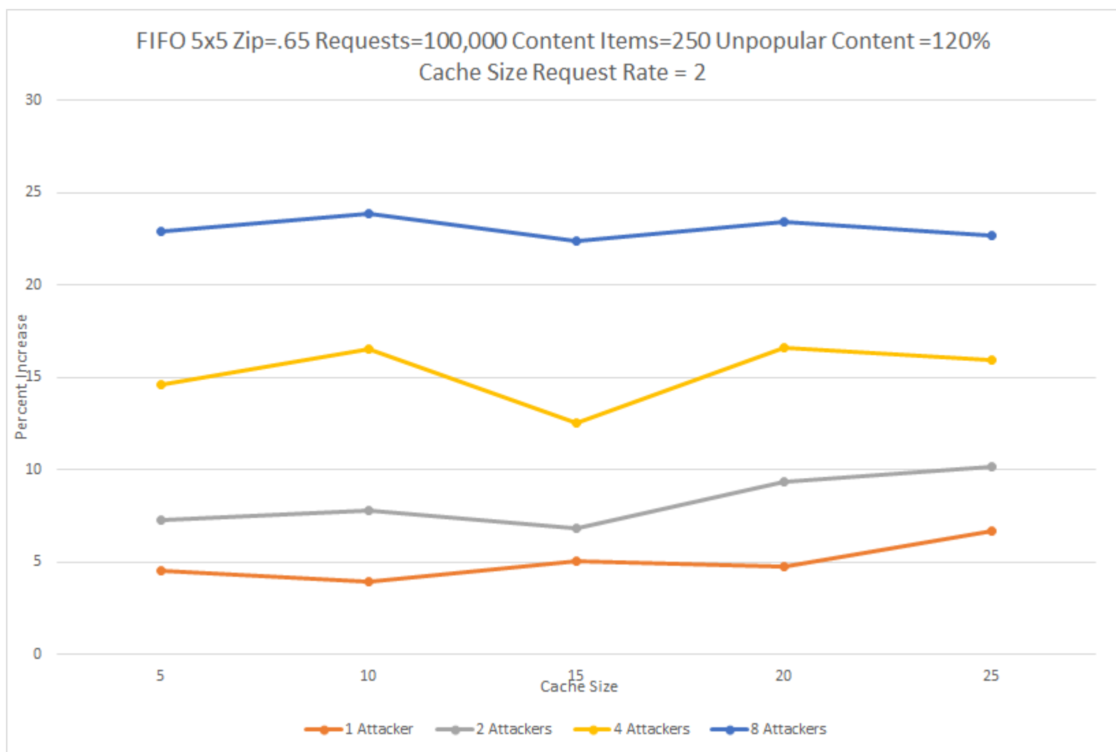
5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 2



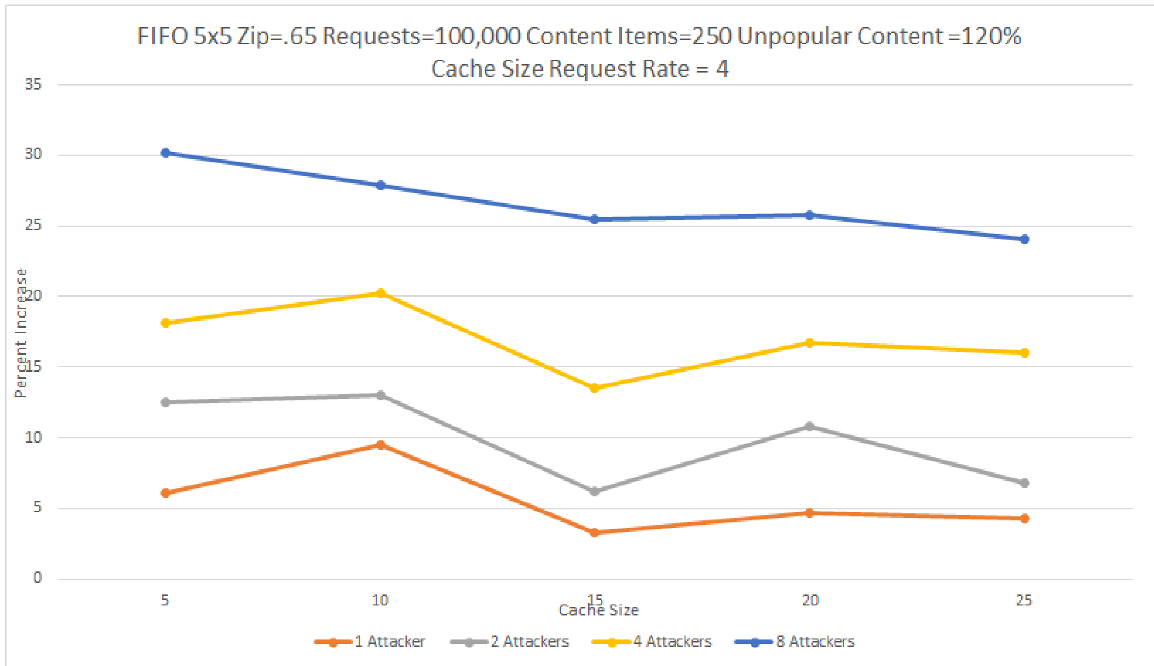
5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 4



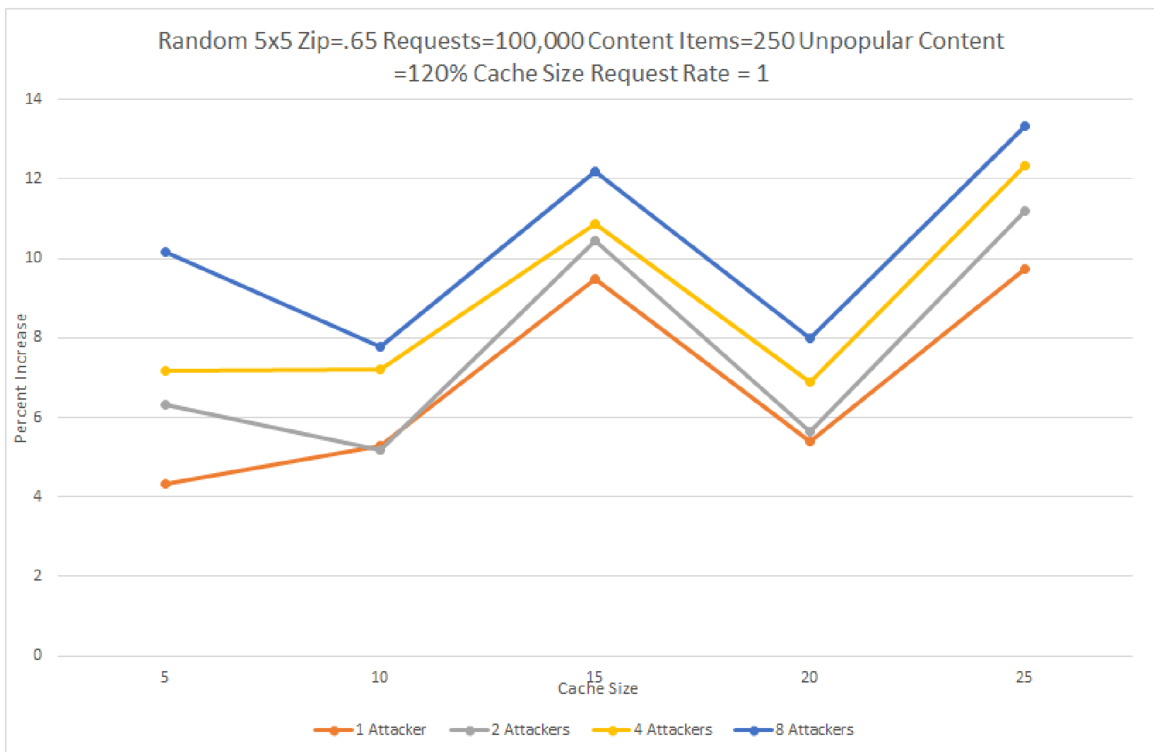
5x5 Square Graph. FIFO. Zipfian = 0.65. Request rate of 1



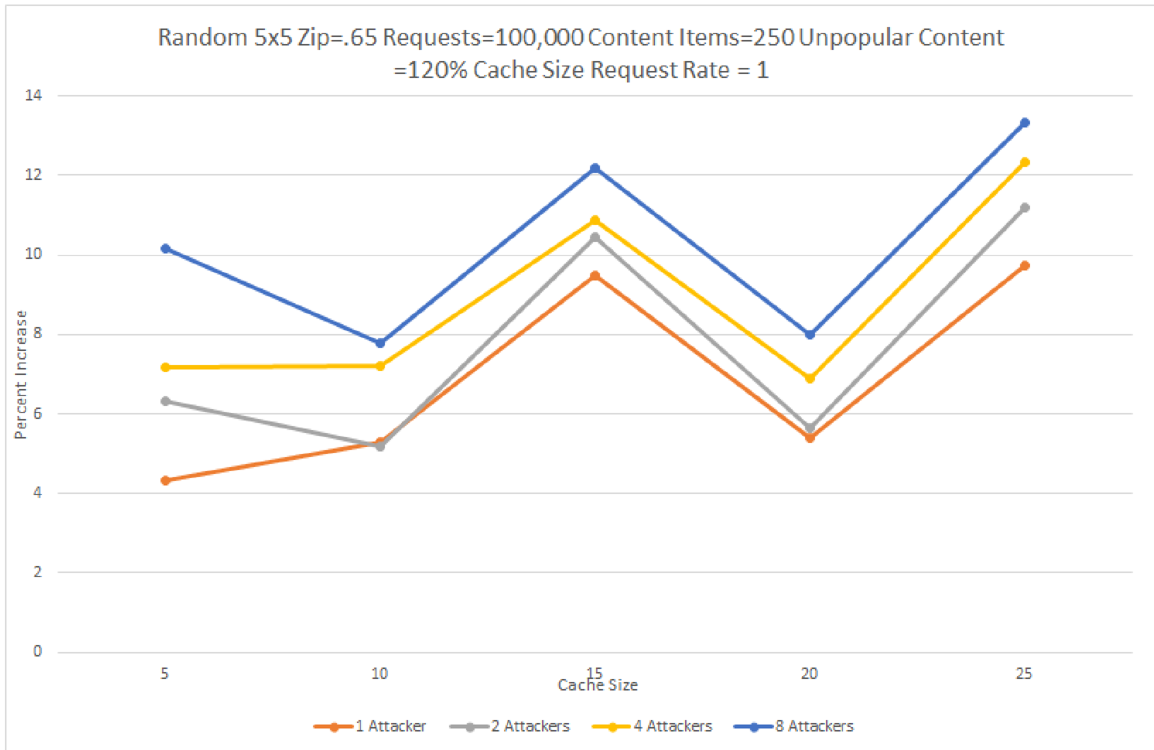
5x5 Square Graph. FIFO. Zipfian = 0.65. Request rate of 2



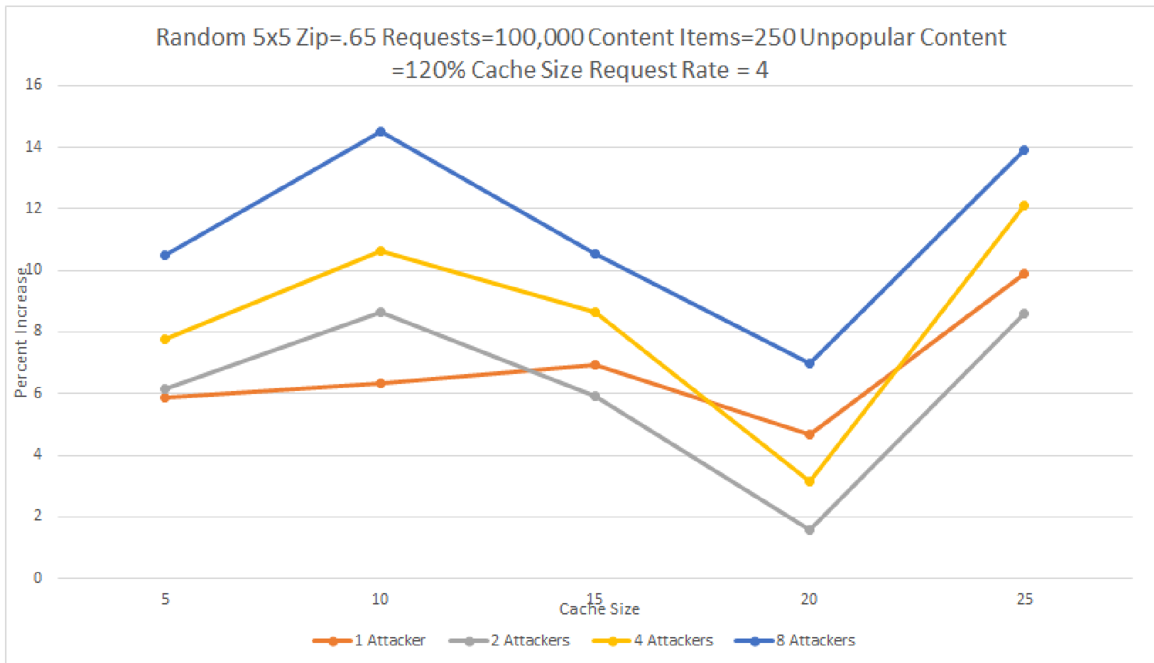
5x5 Square Graph. FIFO. Zipfian = 0.65. Request rate of 4



5x5 Square Graph. Random. Zipfian = 0.65. Request rate of 1



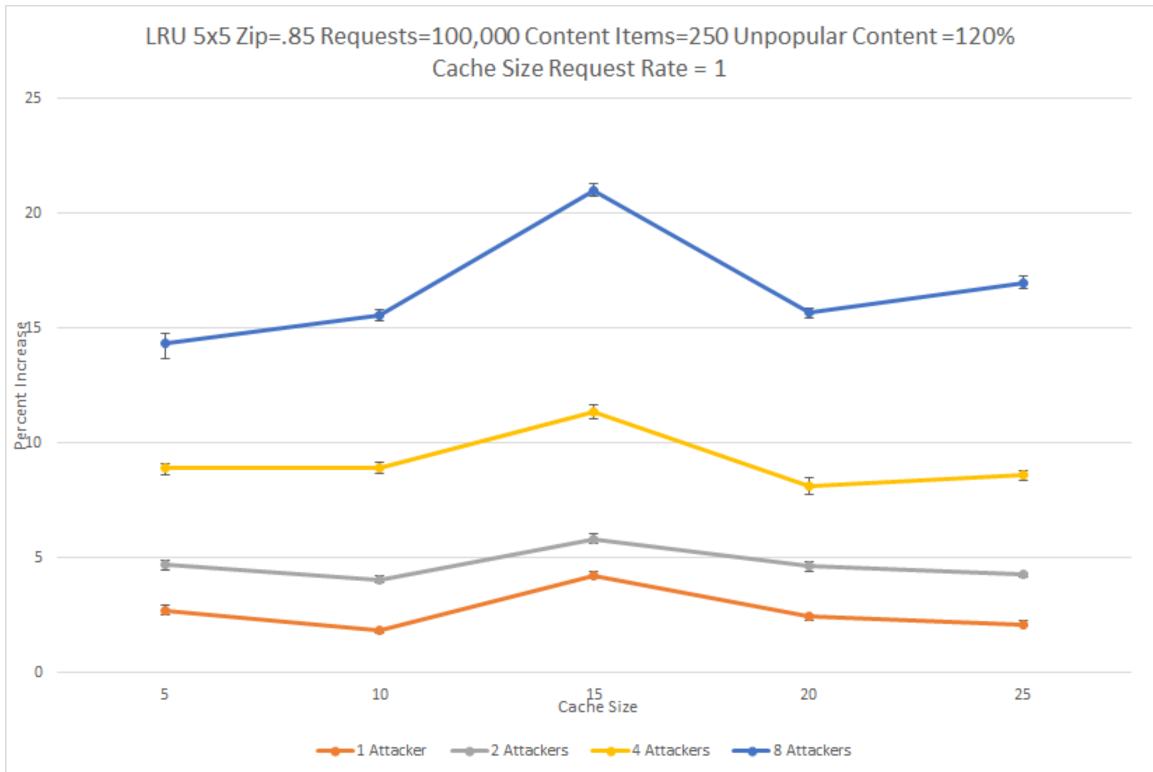
5x5 Square Graph. Random. Zipfian = 0.65. Request rate of 2



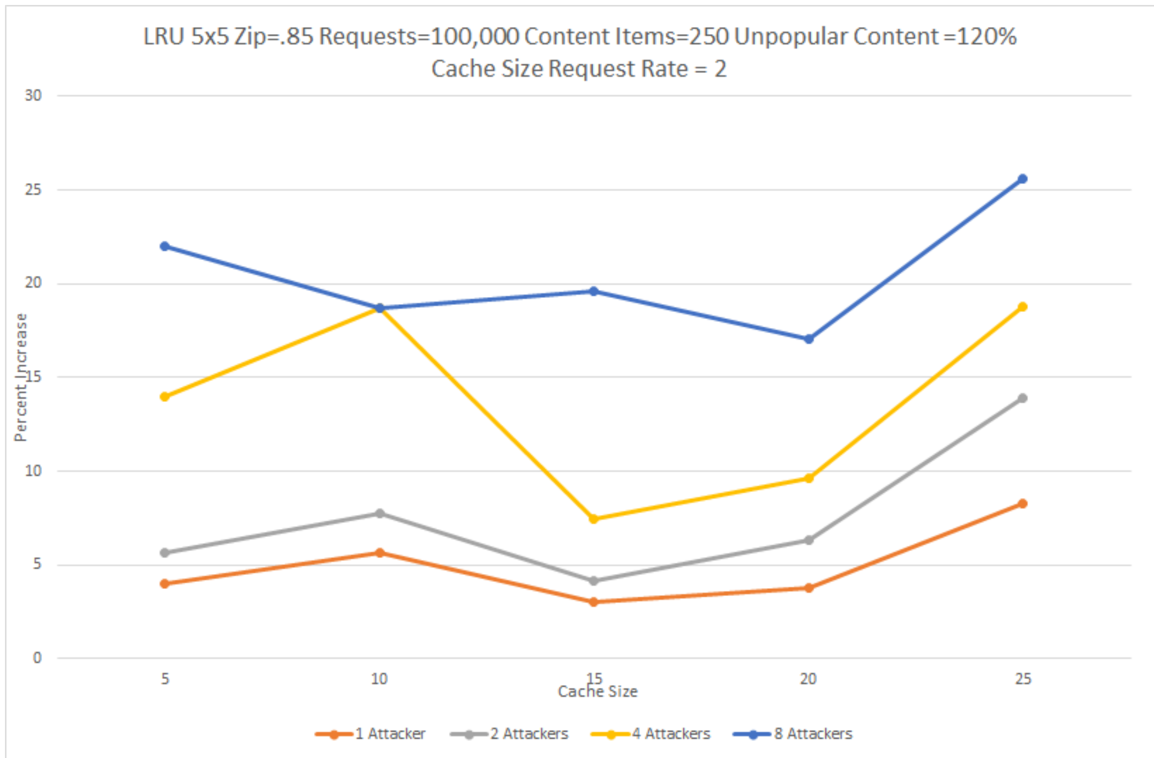
5x5 Square Graph. Random. Zipfian = 0.65. Request rate of 4

Rate	Cache Type	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean 16%: Attackers	Std. Deviation: 16% Attackers	Mean: 32% Attackers	Std. Deviation: 32% Attackers
1.0000	LRU	2.8021	0.8014	5.4001	1.3287	9.4280	1.8237	16.5540	2.1850
	FIFO	4.2650	1.5070	6.7171	1.4512	11.6098	1.4974	17.5756	0.8910
	Random	6.8465	2.2845	7.7570	2.5399	8.9012	2.2551	10.2906	2.2157
1 Total		**4.6379**	**2.3466**	**6.6247**	**2.0907**	**9.9797**	**2.2195**	**14.8067**	**3.7235**
2.0000	LRU	4.6778	1.4546	7.9589	1.6079	13.5661	2.3762	21.9207	3.7898
	FIFO	4.9995	0.9143	8.2810	1.2684	15.2432	1.5159	23.0598	0.5477
	Random	4.6945	1.2789	6.3522	1.2567	8.0969	1.0437	11.8739	1.5567
2 Total		**4.7906**	**1.2454**	**7.5307**	**1.6236**	**12.3021**	**3.5102**	**18.9515**	**5.5639**
4.0000	LRU	6.2265	1.6745	10.4373	1.0293	16.9176	2.3334	25.5914	1.8264
	FIFO	5.5726	2.1682	9.8396	2.8519	16.9229	2.2278	26.6797	2.1167
	Random	6.7405	1.7346	6.1768	2.5650	8.4563	3.0551	11.2954	2.7274
4 Total		**6.1799**	**1.9321**	**8.8179**	**2.9672**	**14.0989**	**4.7434**	**21.1888**	**7.3636**
Grand Total		**5.2028**	**2.0194**	**7.6578**	**2.4658**	**12.1269**	**4.0116**	**18.3157**	**6.3250**

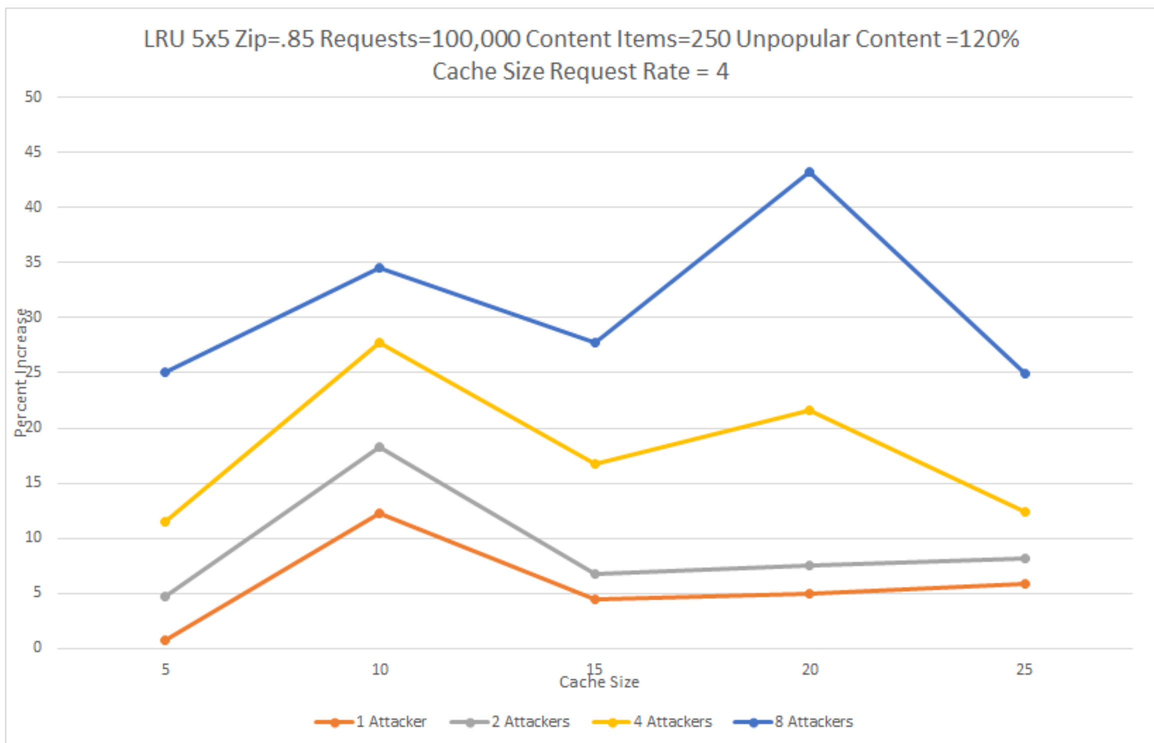
5x5 Square Graph (Zipfian=0.65): Percentage Increase Statistics



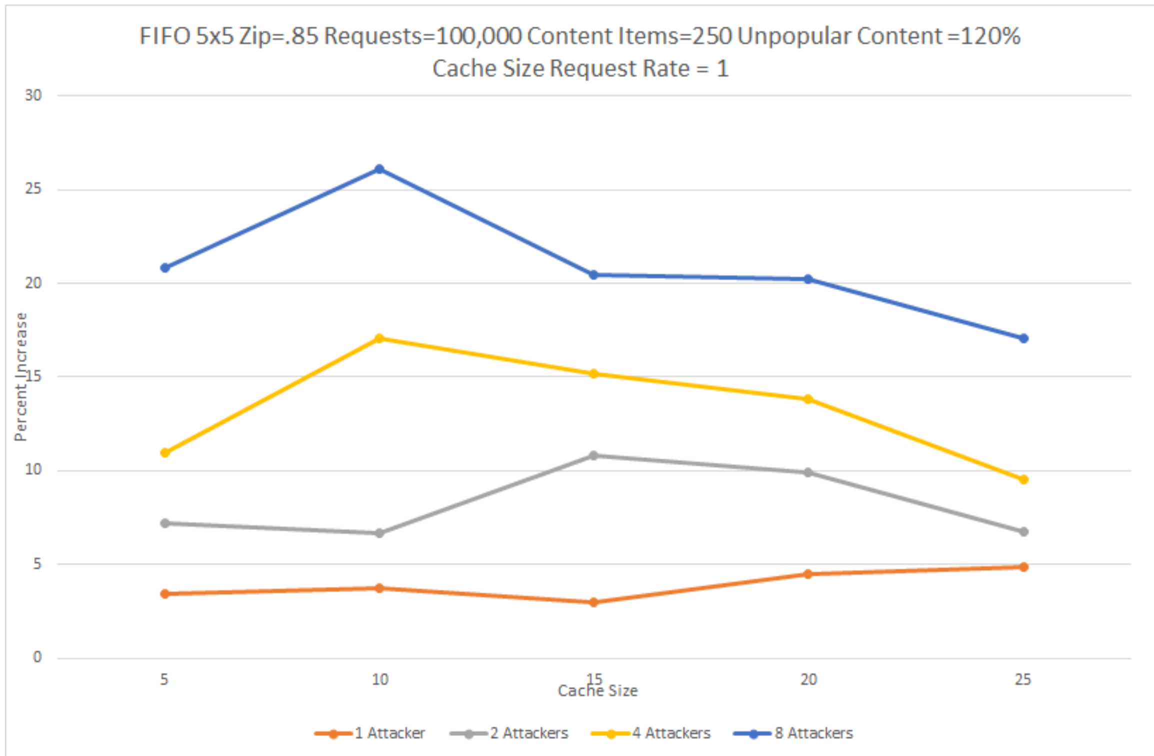
5x5 Square Graph. LRU. Zipfian = 0.85. Request rate of 1



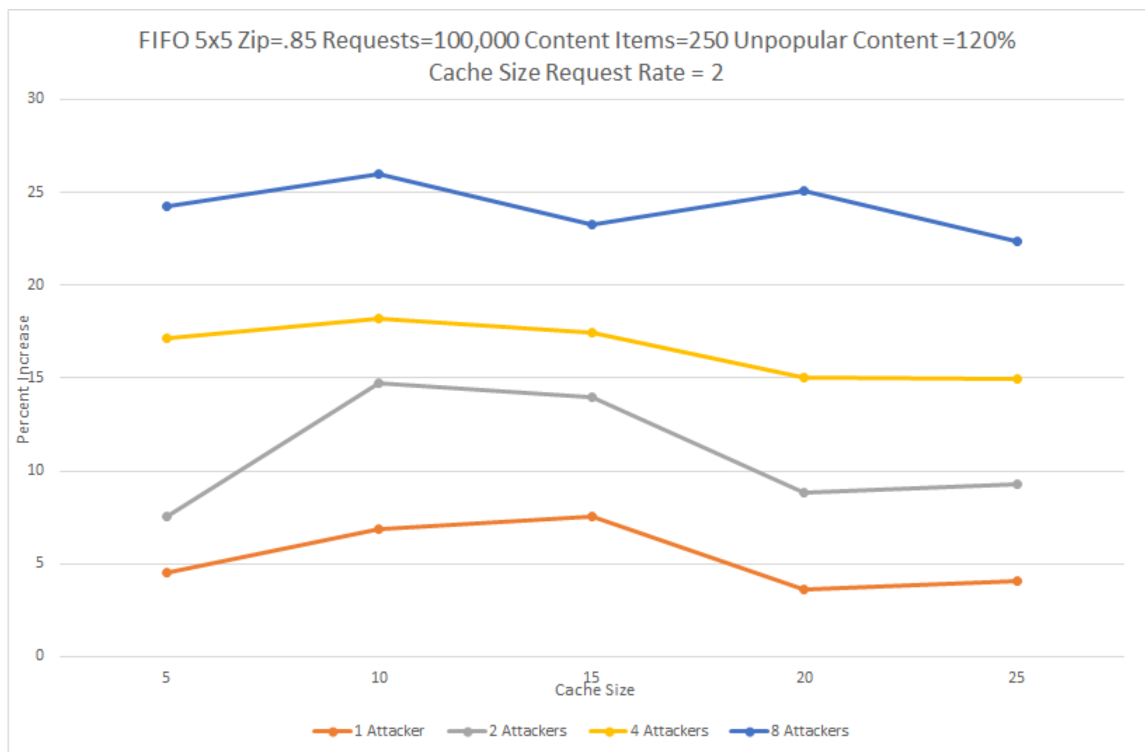
5x5 Square Graph. LRU. Zipfian = 0.85. Request rate of 2



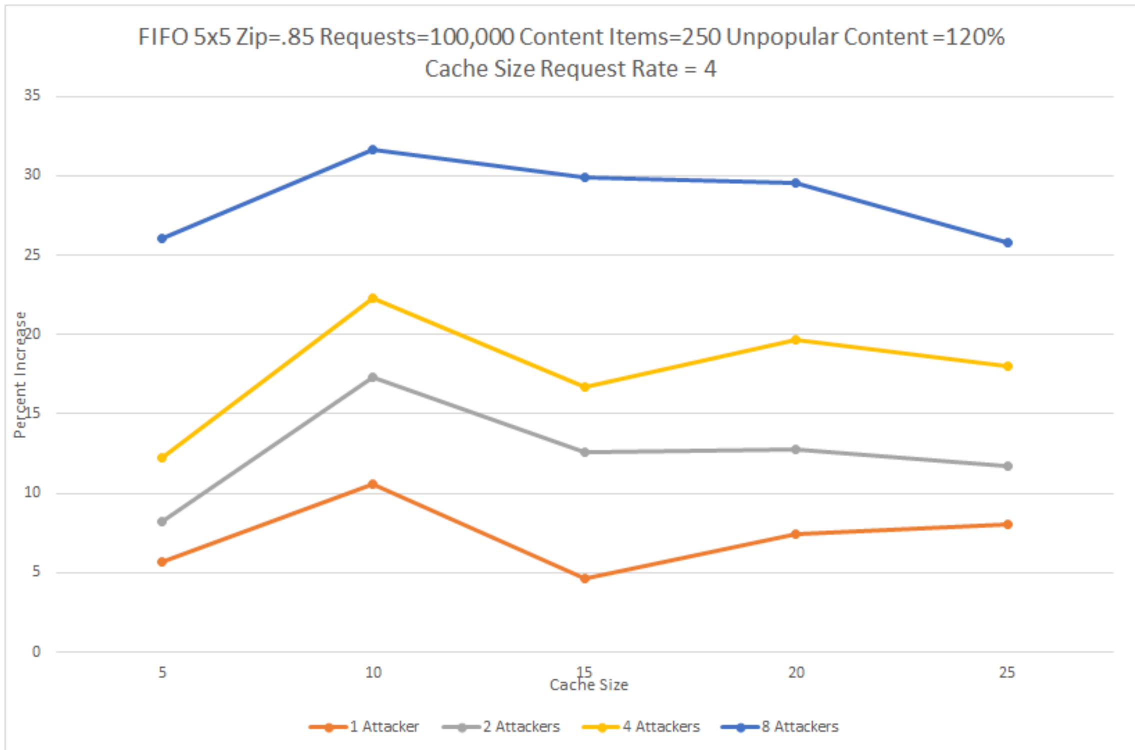
5x5 Square Graph. LRU. Zipfian = 0.85. Request rate of 4



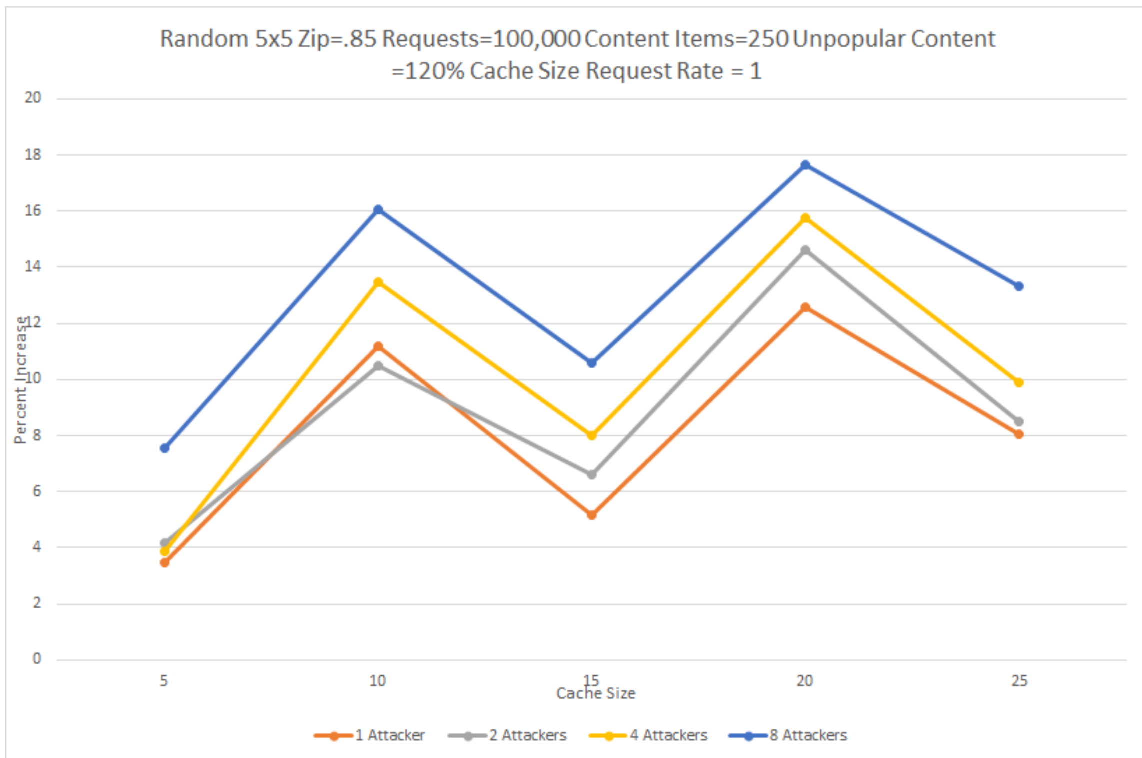
5x5 Square Graph. FIFO. Zipfian = 0.85. Request rate of 1



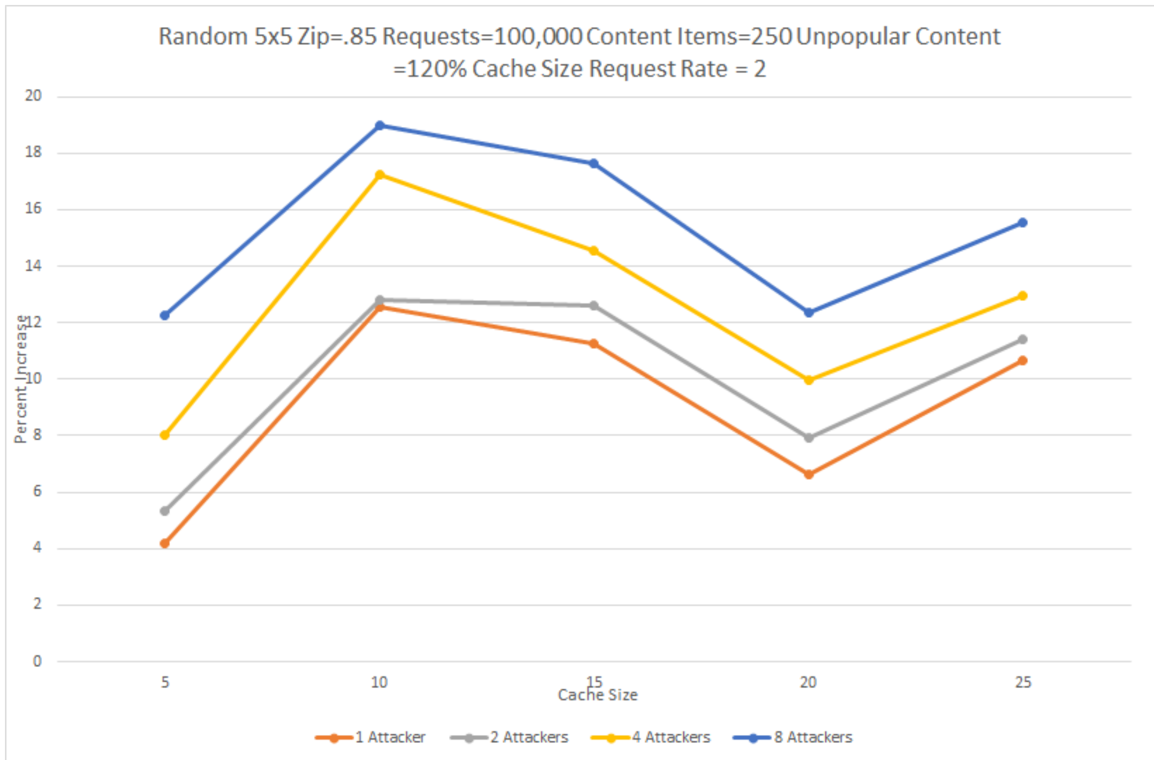
5x5 Square Graph. FIFO. Zipfian = 0.85. Request rate of 2



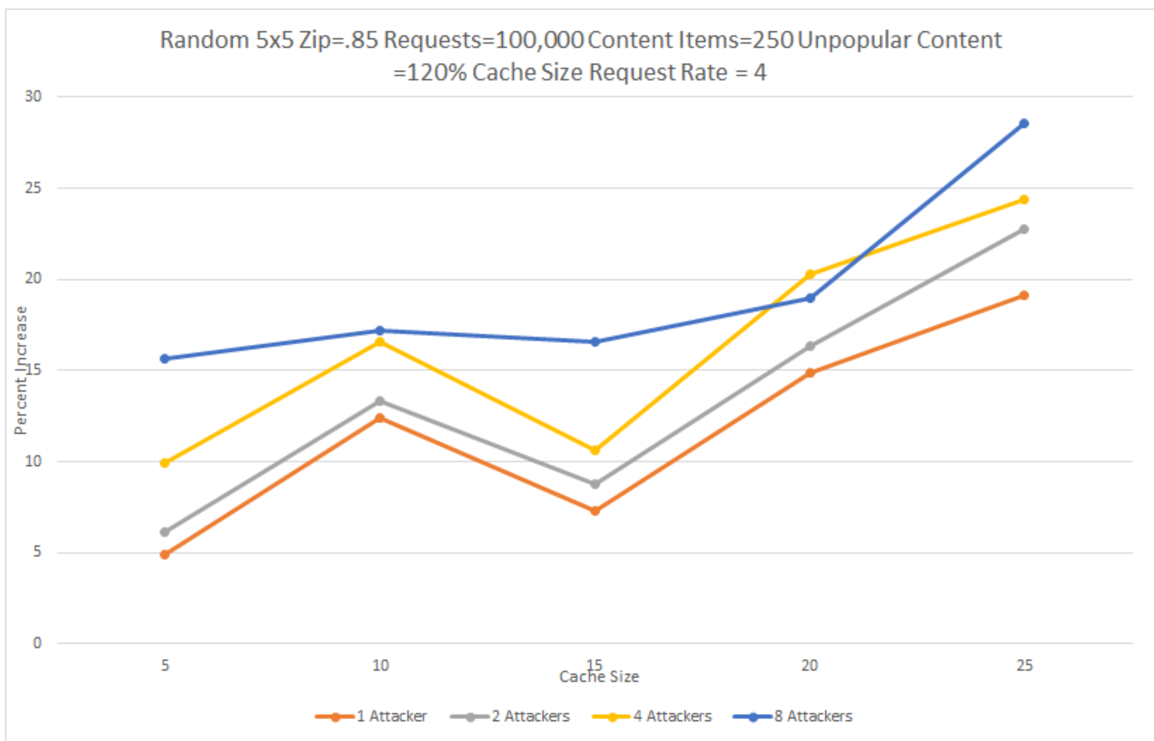
5x5 Square Graph. FIFO. Zipfian = 0.85. Request rate of 4



5x5 Square Graph. Random. Zipfian = 0.85. Request rate of 1



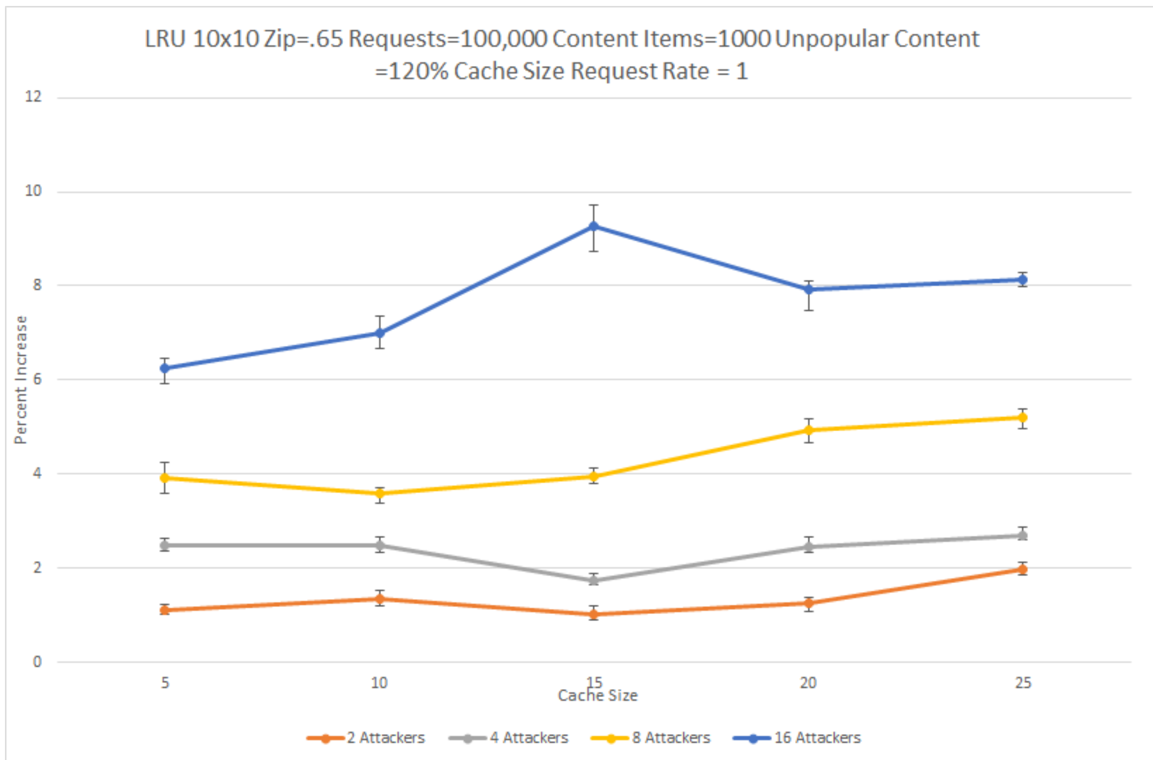
5x5 Square Graph. Random. Zipfian = 0.85. Request rate of 2



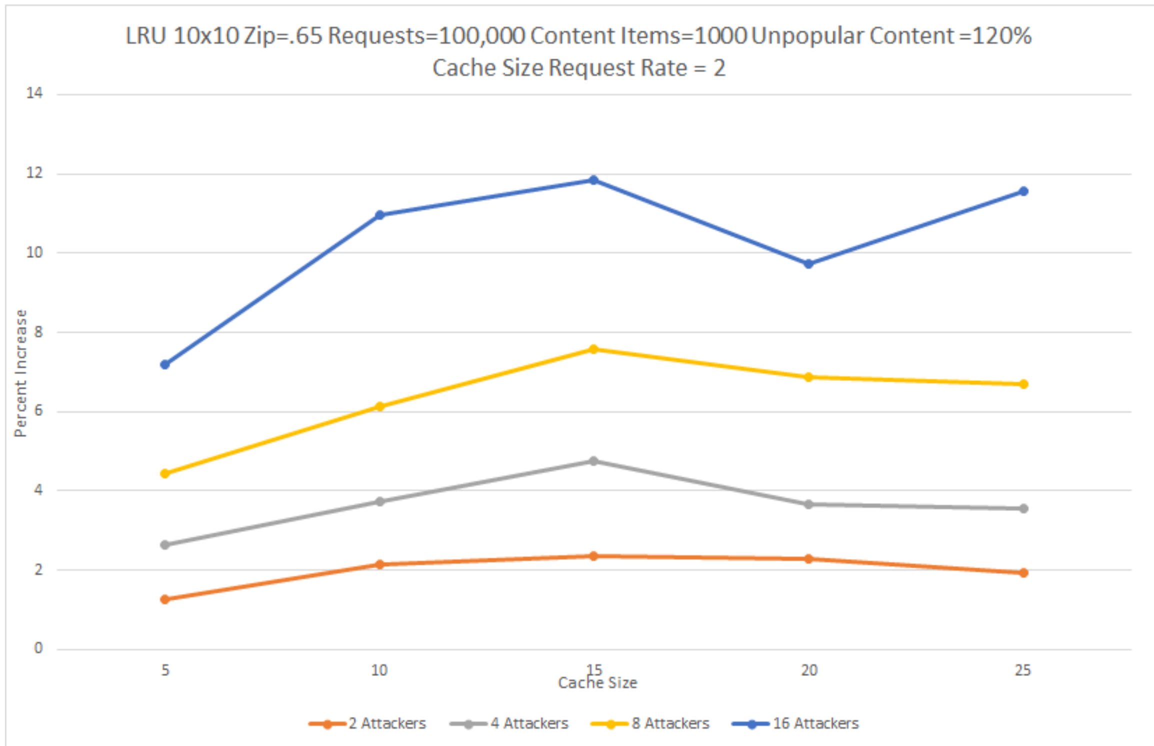
5x5 Square Graph. Random. Zipfian = 0.85. Request rate of 4

Rate	Cache Type	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers	Mean: 32% Attackers	Std. Deviation: 32% Attackers
1	LRU	2.6791	0.8430	4.6961	0.5982	9.1876	1.1279	16.7207	2.2975
	FIFO	3.8945	0.6821	8.2621	1.7348	13.2921	2.7365	20.9475	2.9031
	Random	8.1067	3.4497	8.8739	3.5536	10.1953	4.1504	13.0436	3.6458
1 Total		**4.8935**	**3.1253**	**7.2773**	**2.9539**	**10.8917**	**3.4223**	**16.9039**	**4.4078**
2	LRU	4.9554	1.8696	7.5529	3.3804	13.7105	4.6088	20.5979	2.9804
	FIFO	5.3282	1.5871	10.8814	2.9006	16.5450	1.3192	24.1968	1.2855
	Random	9.0632	3.1369	10.0276	2.9190	12.5534	3.2653	15.3625	2.7185
2 Total		**6.4489**	**2.9538**	**9.4873**	**3.3832**	**14.2696**	**3.7452**	**20.0524**	**4.3739**
4	LRU	5.6615	3.7067	9.1210	4.7414	17.9963	6.0560	31.0906	7.0086
	FIFO	7.2660	2.0473	12.5371	2.9059	17.7808	3.3375	28.5699	2.2961
	Random	11.7210	5.1389	13.4718	5.8235	16.3761	5.5339	19.4002	4.6968
4 Total		**8.2162**	**4.6206**	**11.7099**	**5.0110**	**17.3844**	**5.1635**	**26.3536**	**7.1216**
Grand Total		**6.5195**	**3.8889**	**9.4915**	**4.2858**	**14.1819**	**4.9494**	**21.1033**	**6.7226**

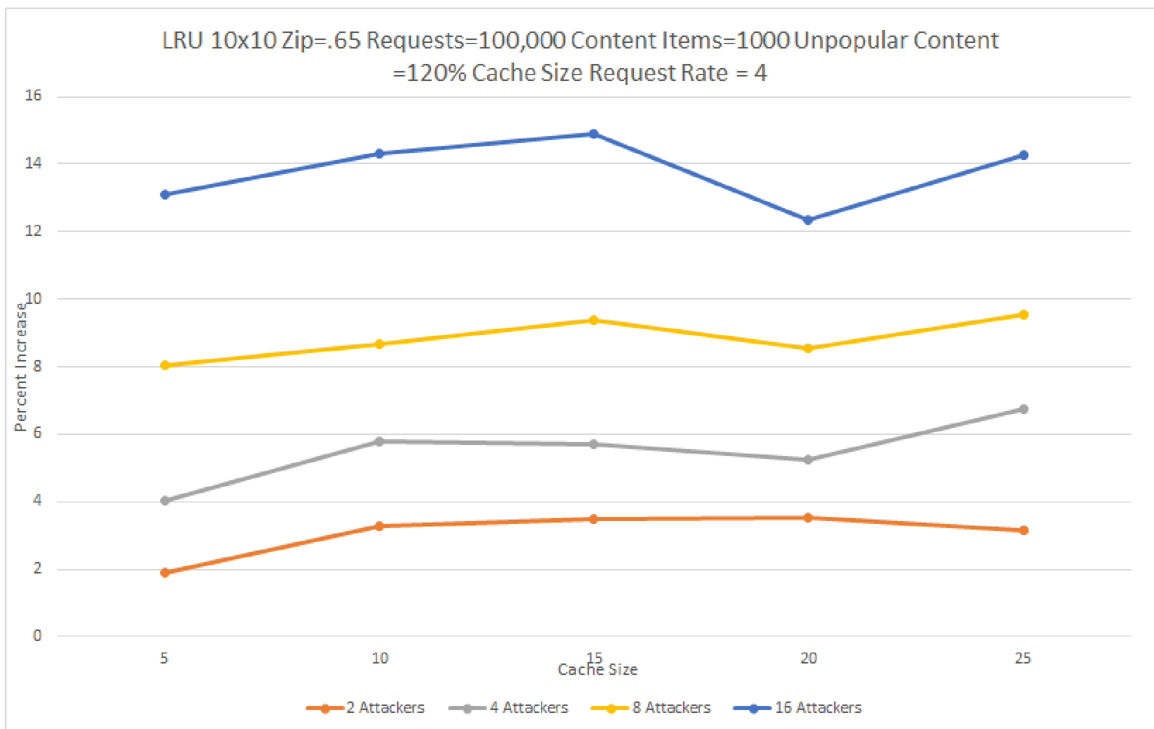
5x5 Square Graph (Zipfian=0.85): Percentage Increase Statistics



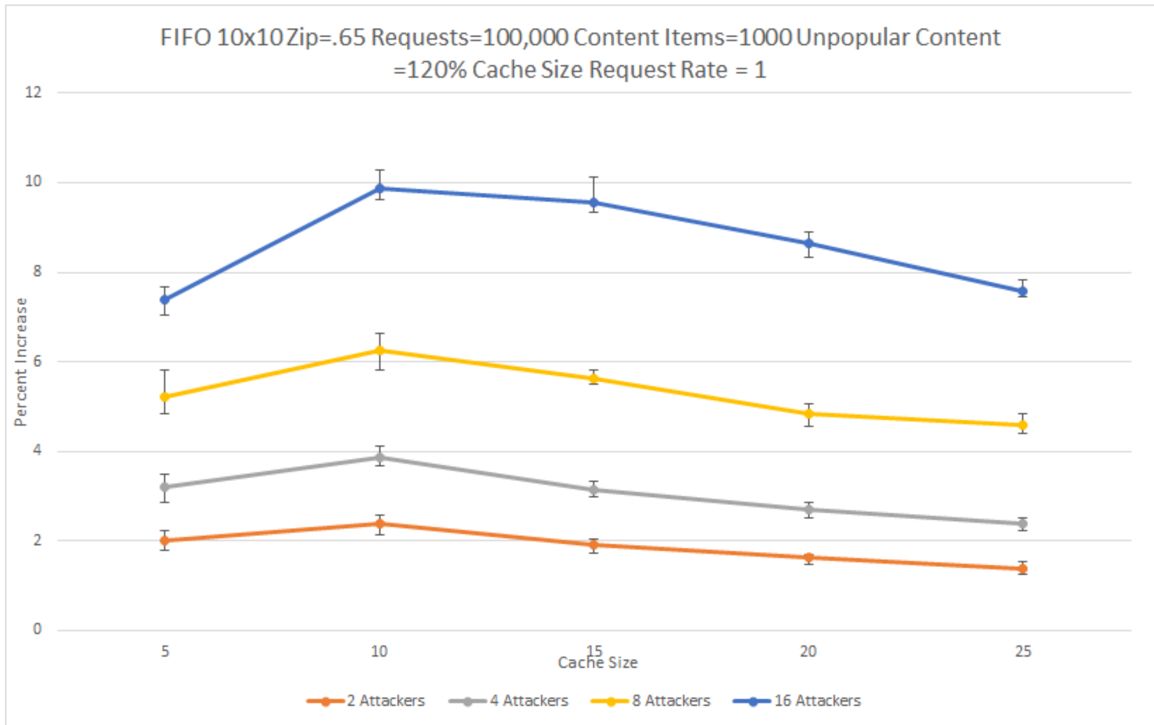
10x10 Square Graph. LRU. Zipfian = 0.65. Request rate of 1



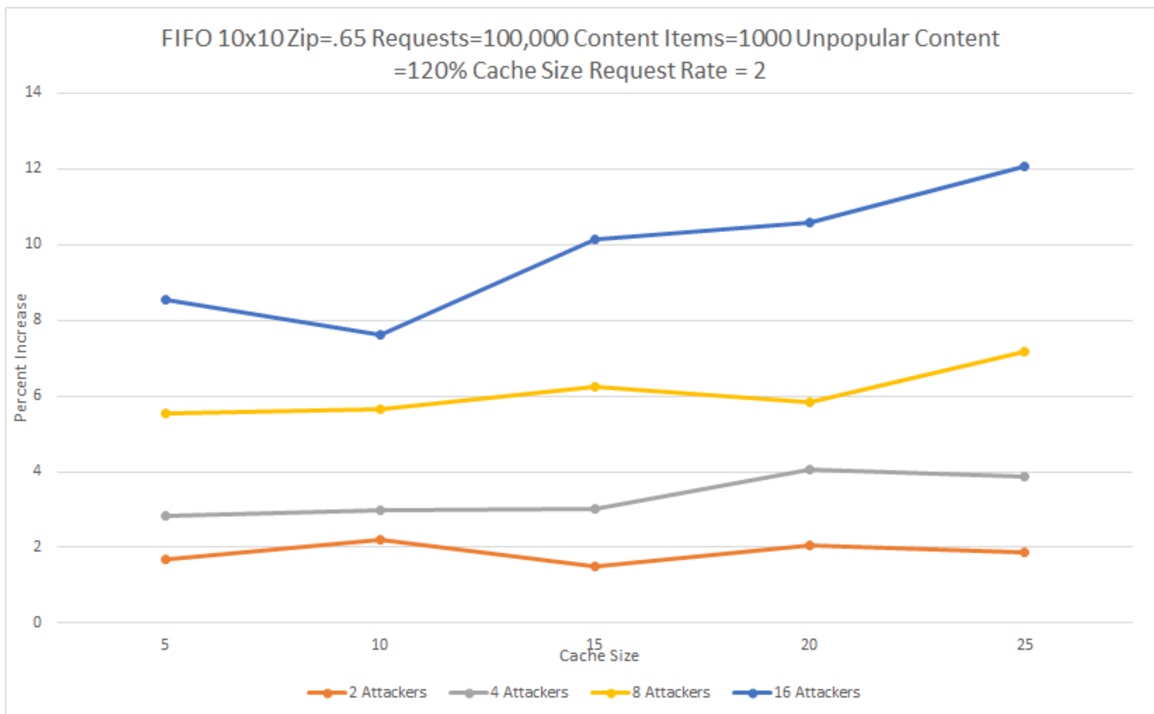
10x10 Square Graph. LRU. Zipfian = 0.65. Request rate of 2



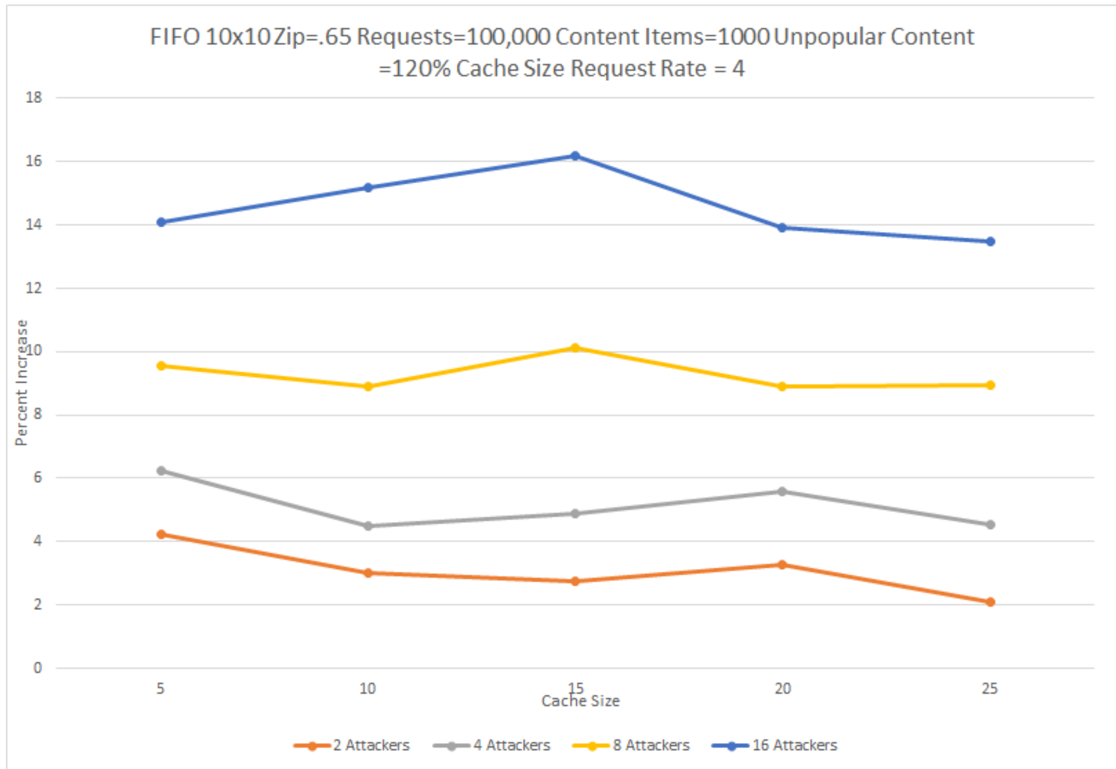
10x10 Square Graph. LRU. Zipfian = 0.65. Request rate of 4



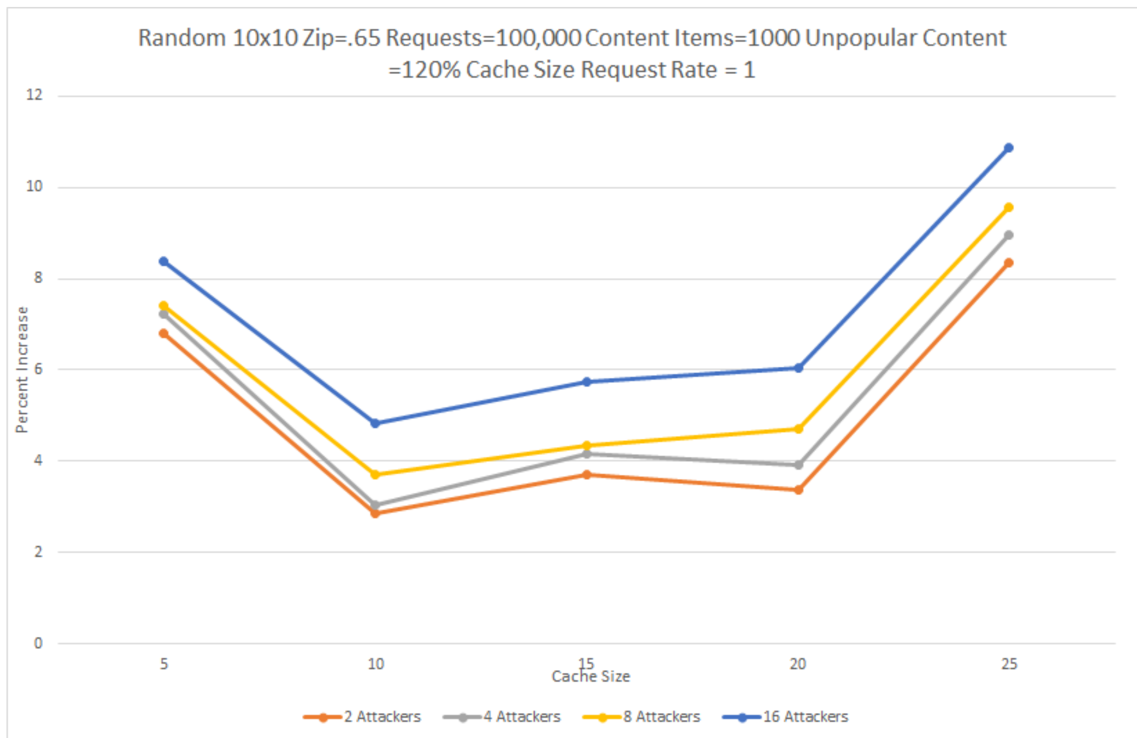
10x10 Square Graph. FIFO. Zipfian = 0.65. Request rate of 1



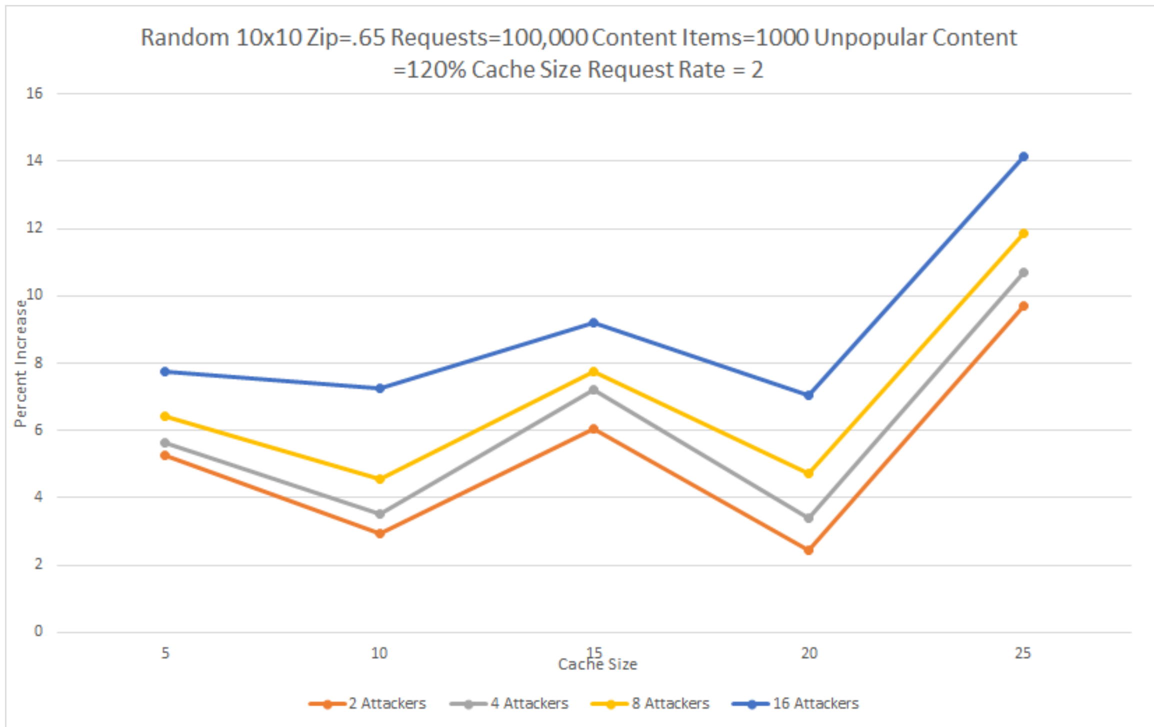
10x10 Square Graph. FIFO. Zipfian = 0.65. Request rate of 2



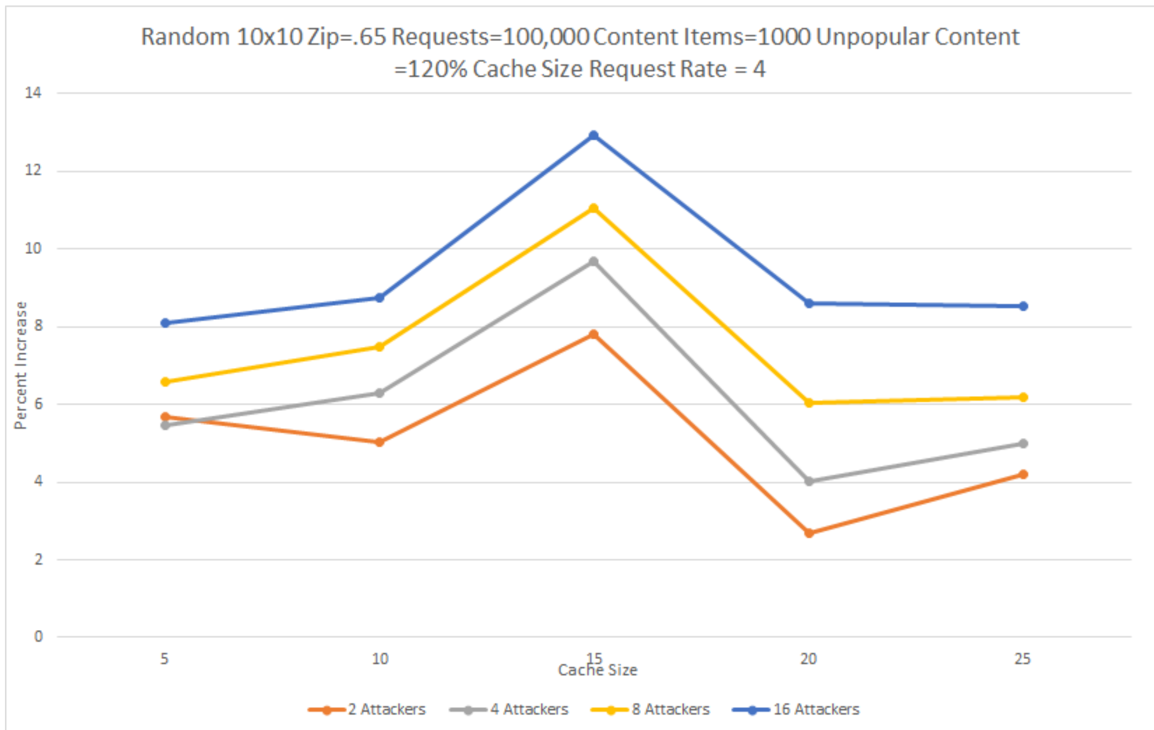
10x10 Square Graph. FIFO. Zipfian = 0.65. Request rate of 4



10x10 Square Graph. Random. Zipfian = 0.65. Request rate of 1



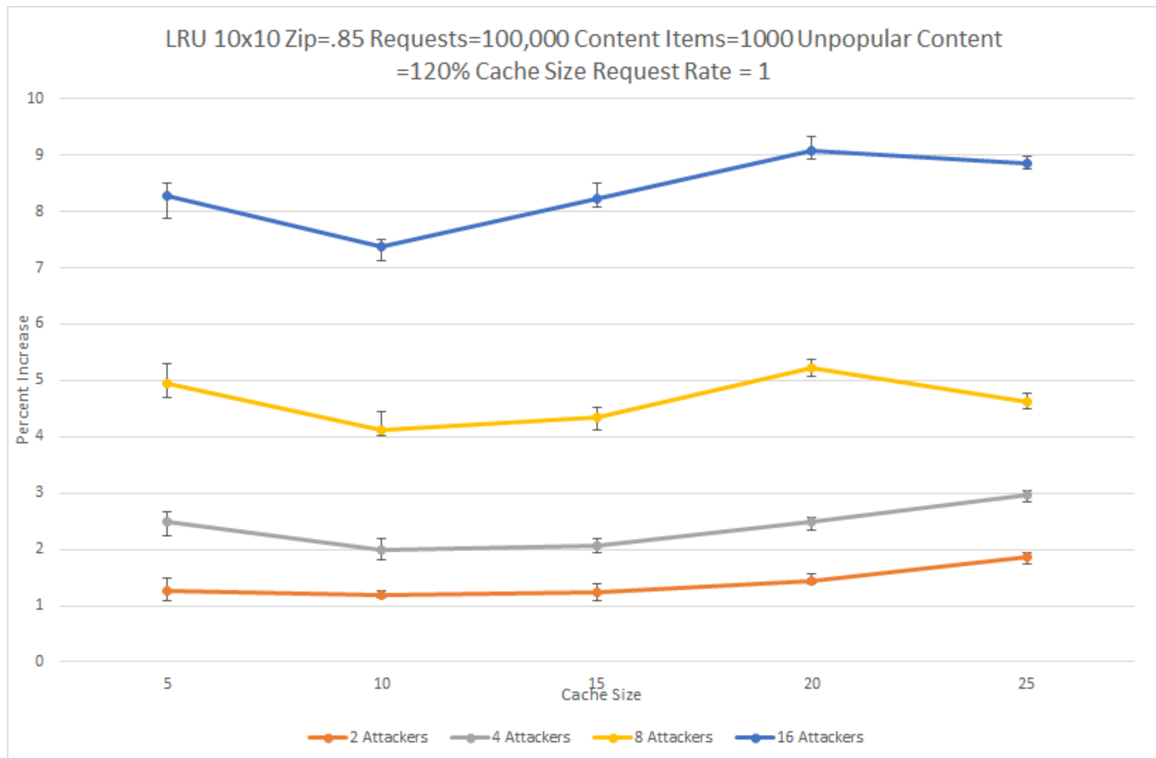
10x10 Square Graph. Random. Zipfian = 0.65. Request rate of 2



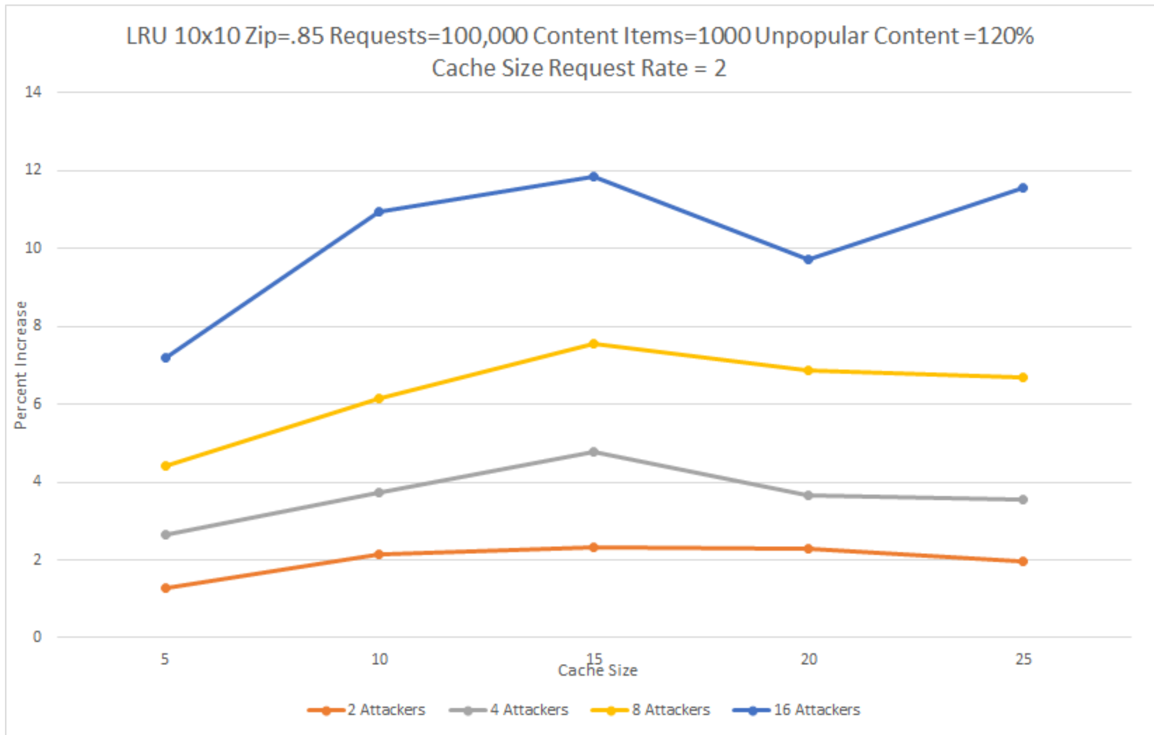
10x10 Square Graph. Random. Zipfian = 0.65. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	1.3405	0.3363	2.3684	0.3245	4.3202	0.6322	7.7166	1.0344
	FIFO	1.8617	0.3391	3.0553	0.5060	5.3064	0.5942	8.6093	1.0126
	Random	5.0120	2.1569	5.4562	2.2420	5.9421	2.2029	7.1760	2.1798
1 Total		**2.7381**	**2.0634**	**3.6266**	**1.8837**	**5.1896**	**1.5211**	**7.8340**	**1.6222**
⊟2	LRU	1.9977	0.3909	3.6754	0.6713	6.3476	1.0592	10.2613	1.6947
	FIFO	1.8622	0.2527	3.3473	0.5085	6.0782	0.5963	9.7940	1.5589
	Random	5.2901	2.5830	6.0852	2.6980	7.0544	2.6648	9.0873	2.6381
2 Total		**3.0500**	**2.1928**	**4.3693**	**2.0379**	**6.4934**	**1.7404**	**9.7142**	**2.0785**
⊟4	LRU	3.0606	0.6066	5.4929	0.8816	8.8349	0.5561	13.7887	0.9263
	FIFO	3.0641	0.7003	5.1621	0.6631	9.2712	0.4866	14.5649	0.9853
	Random	5.0844	1.7007	6.0817	1.9419	7.4762	1.8621	9.3821	1.7938
4 Total		**3.7364**	**1.4693**	**5.5789**	**1.3443**	**8.5274**	**1.3864**	**12.5786**	**2.6252**
Grand Total		**3.1748**	**1.9787**	**4.5249**	**1.9537**	**6.7368**	**2.0756**	**10.0422**	**2.9017**

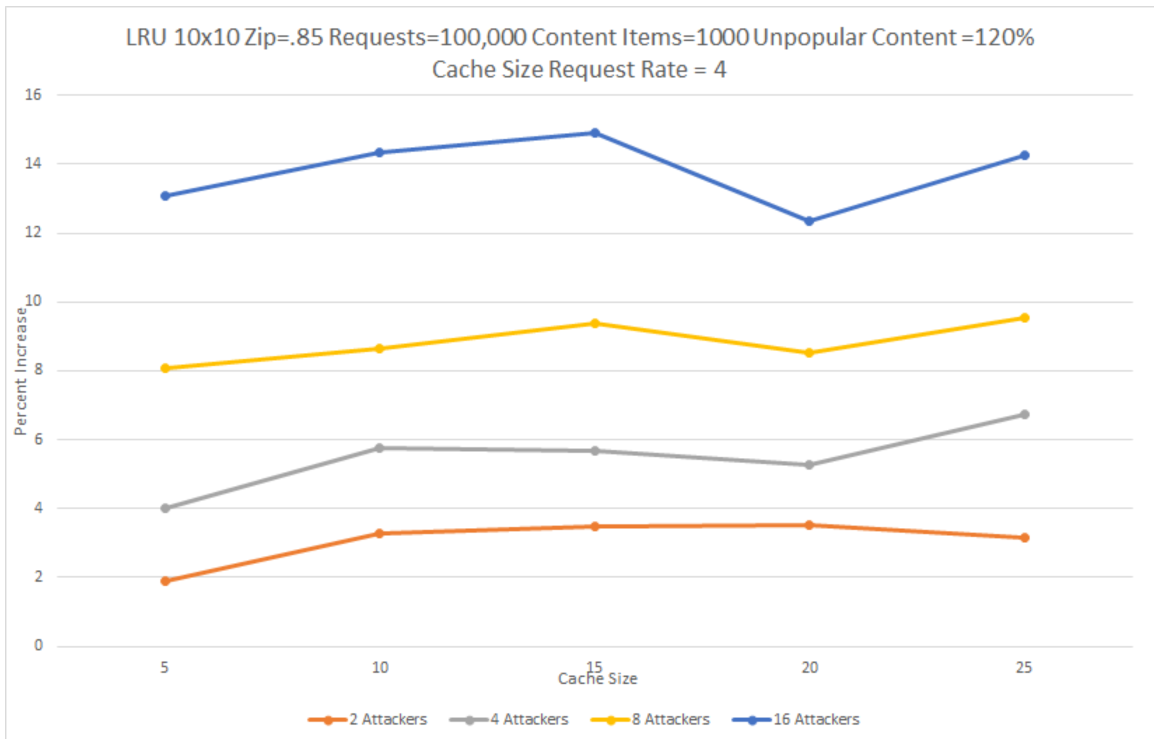
10x10 Square Graph (Zipfian=0.65): Percentage Increase Statistics



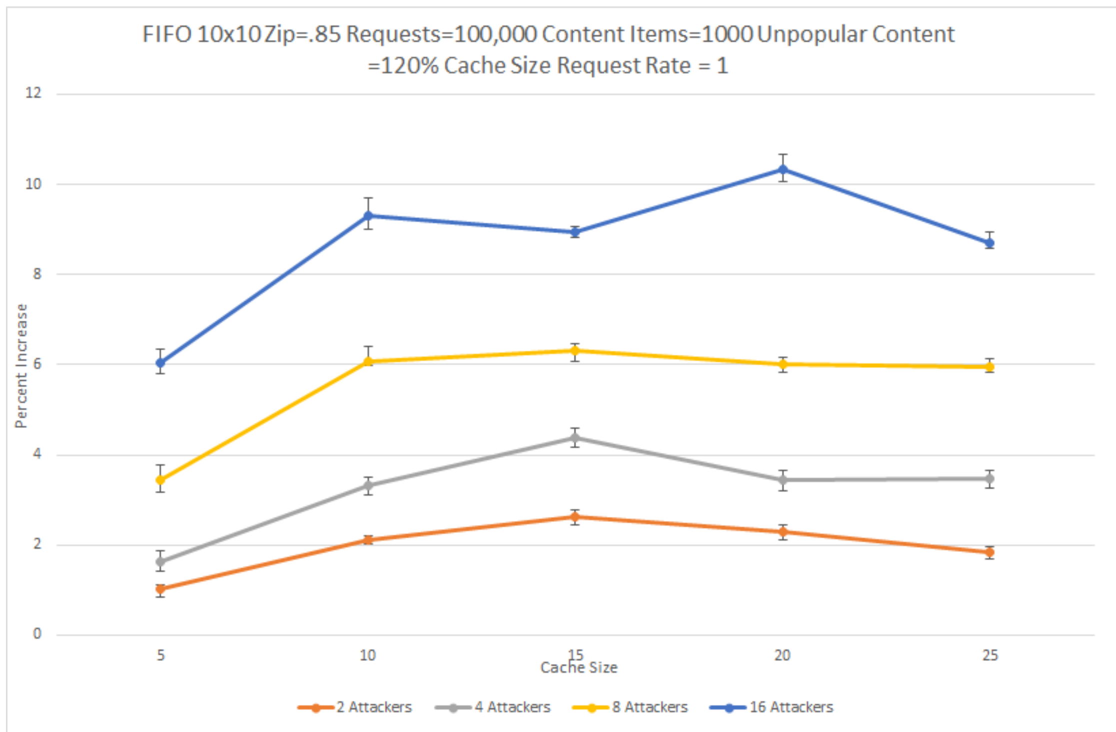
10x10 Square Graph. LRU. Zipfian = 0.85. Request rate of 1



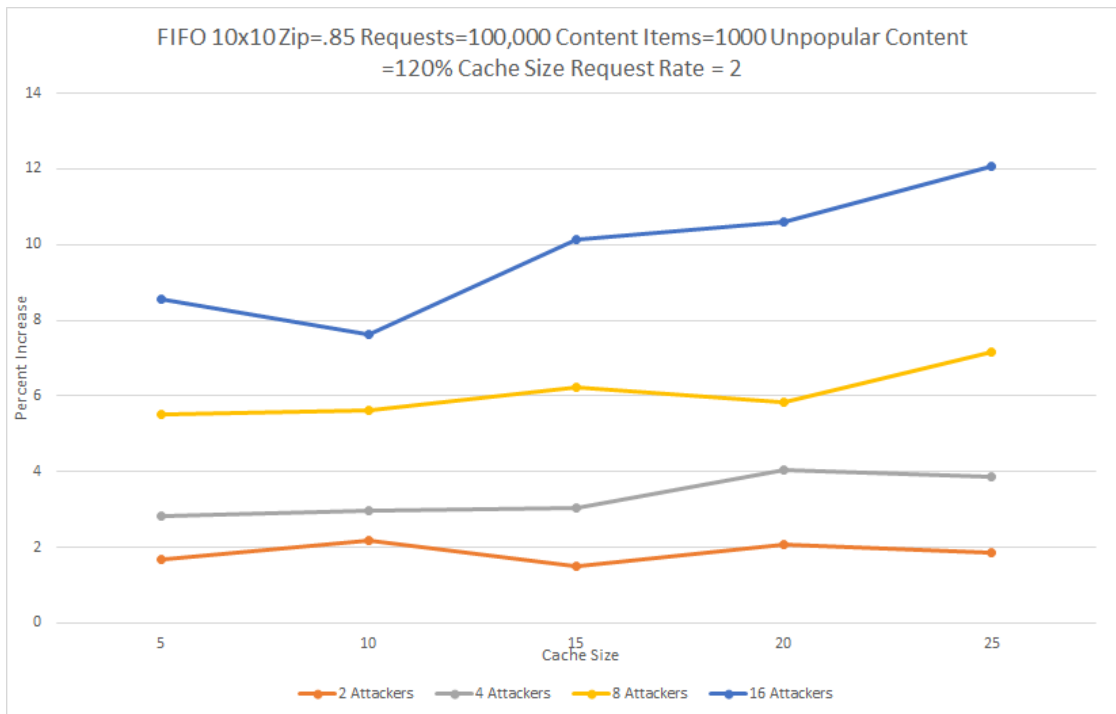
10x10 Square Graph. LRU. Zipfian = 0.85. Request rate of 2



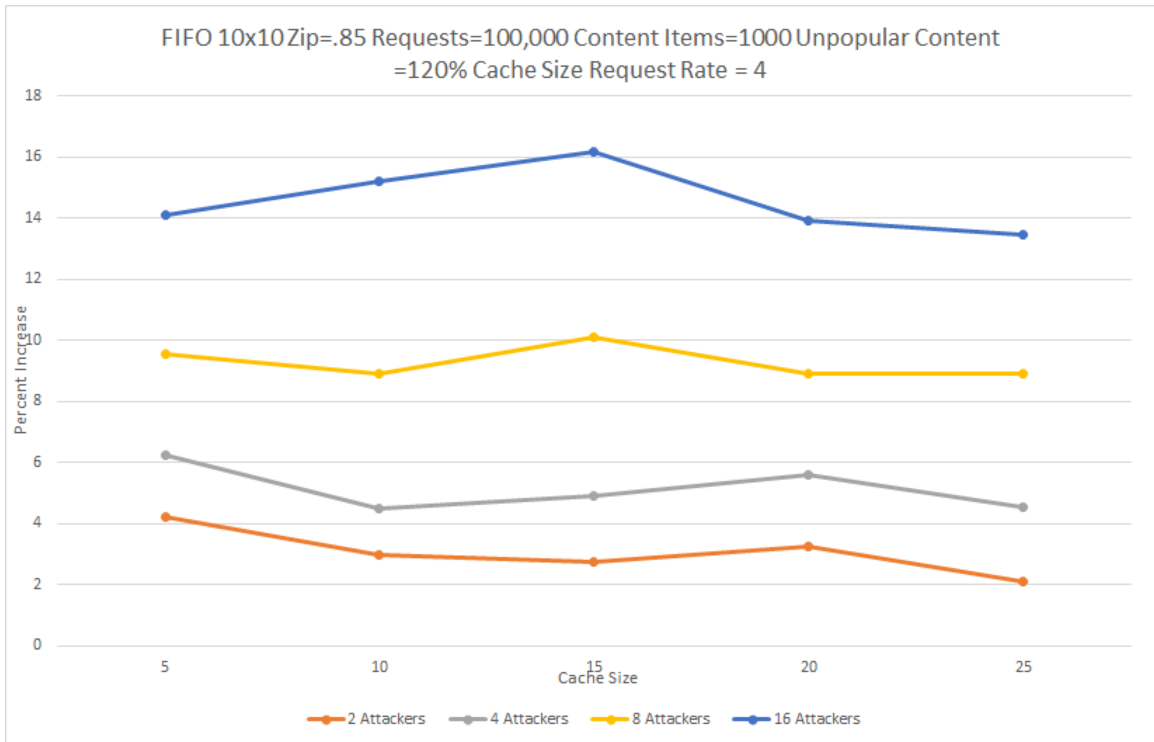
10x10 Square Graph. LRU. Zipfian = 0.85. Request rate of 4



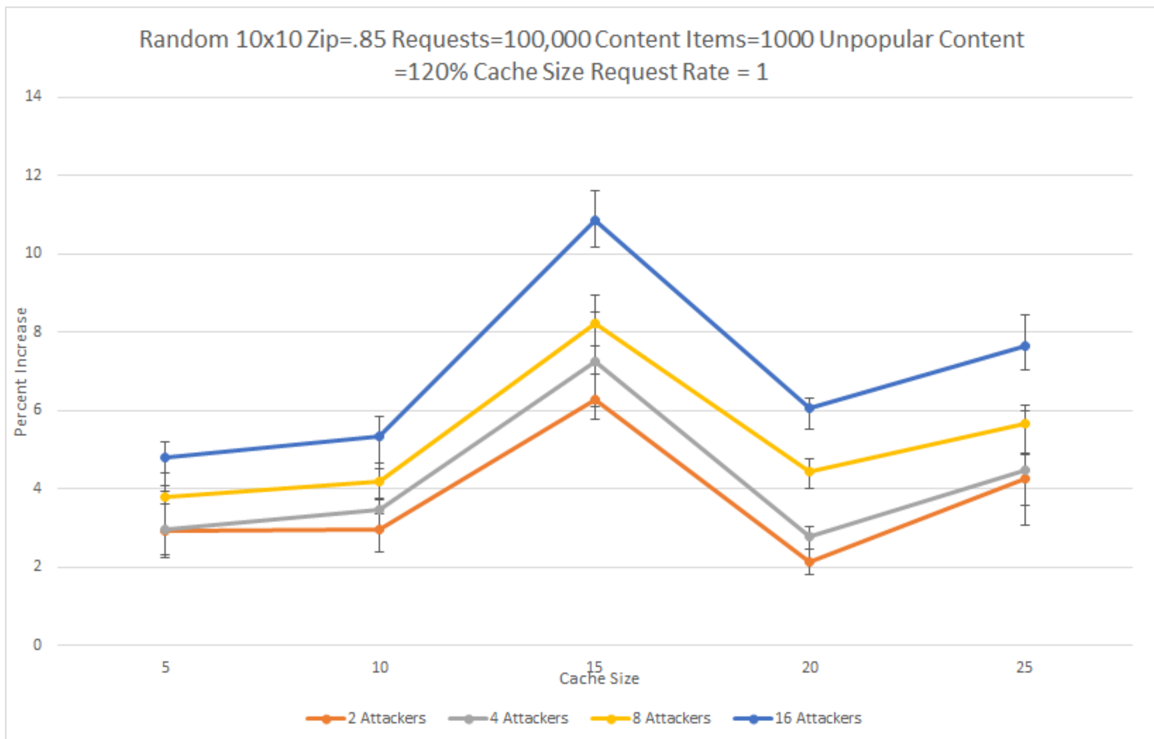
10x10 Square Graph. FIFO. Zipfian = 0.85. Request rate of 1



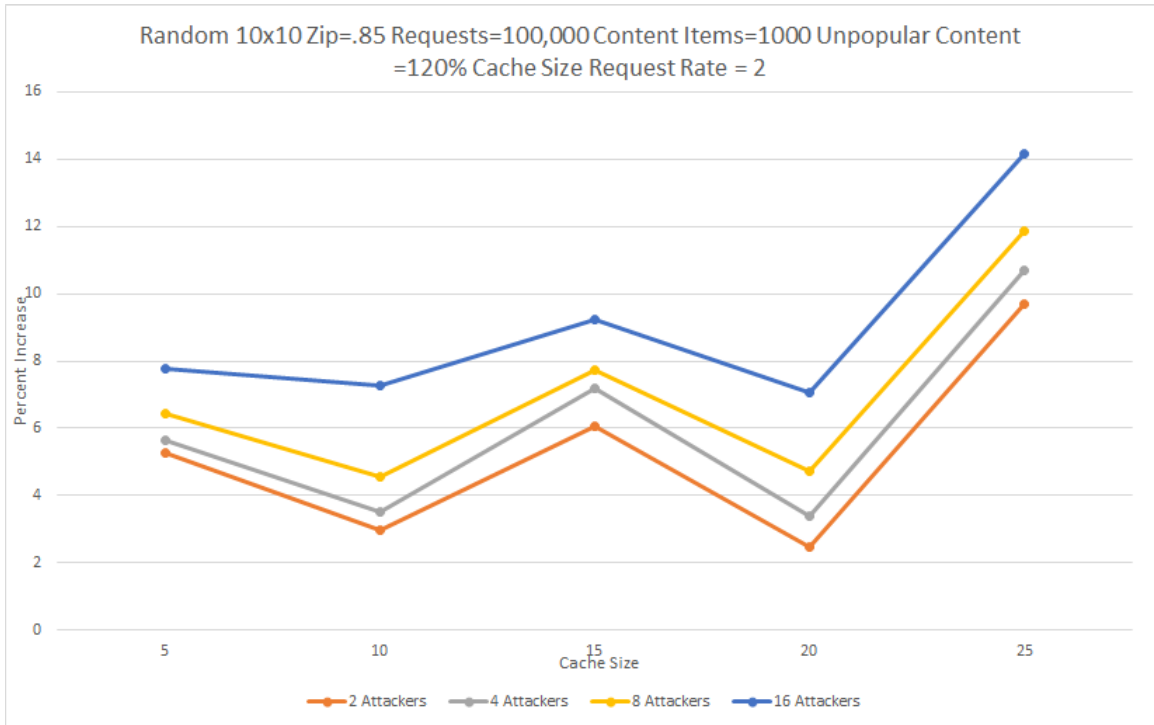
10x10 Square Graph. FIFO. Zipfian = 0.85. Request rate of 2



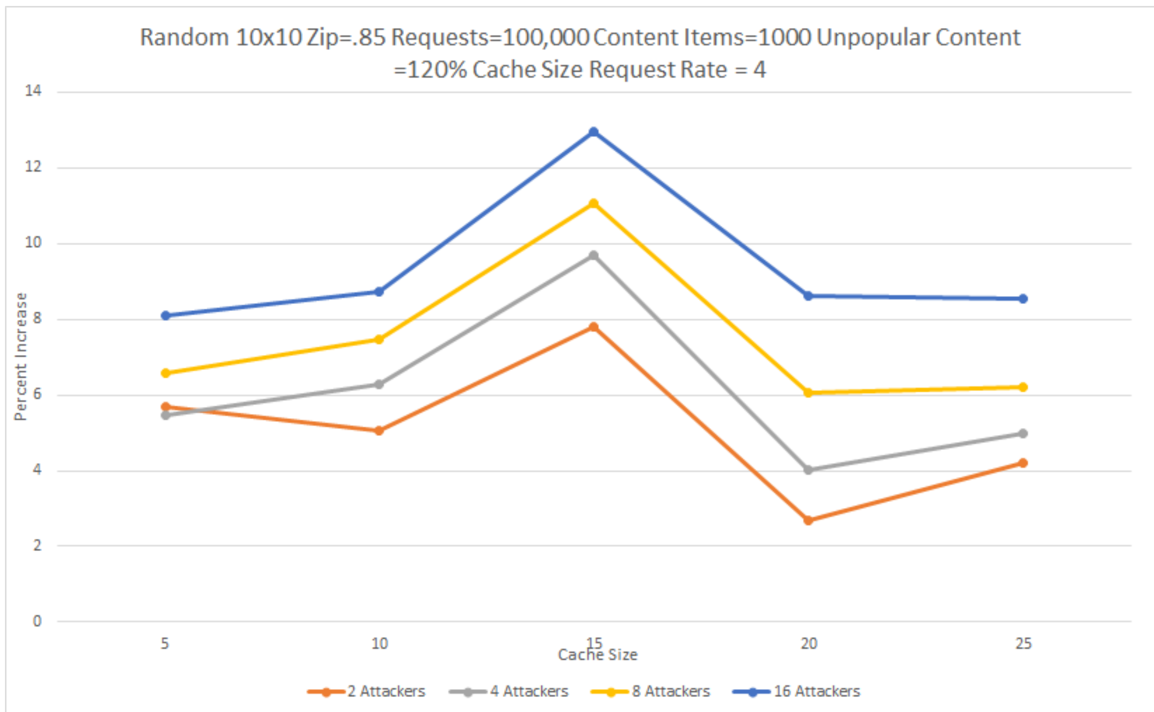
10x10 Square Graph. FIFO. Zipfian = 0.85. Request rate of 4



10x10 Square Graph. Random. Zipfian = 0.85. Request rate of 1



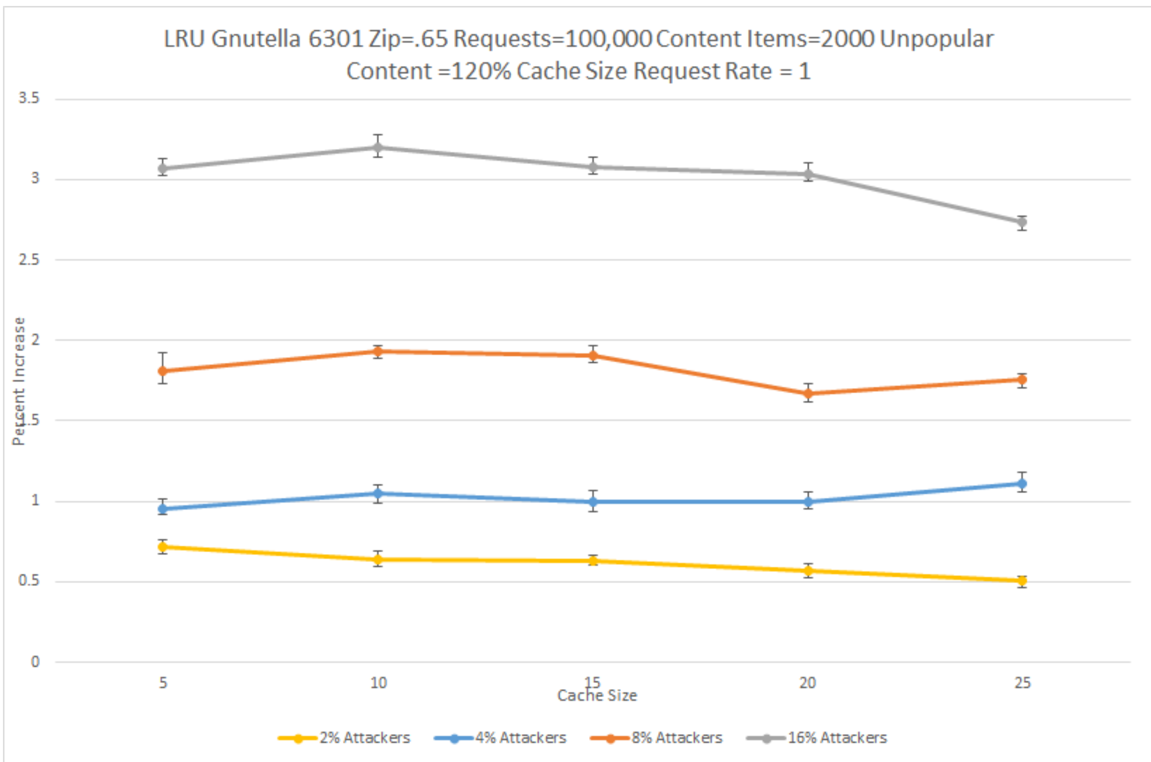
10x10 Square Graph. Random. Zipfian = 0.85. Request rate of 2



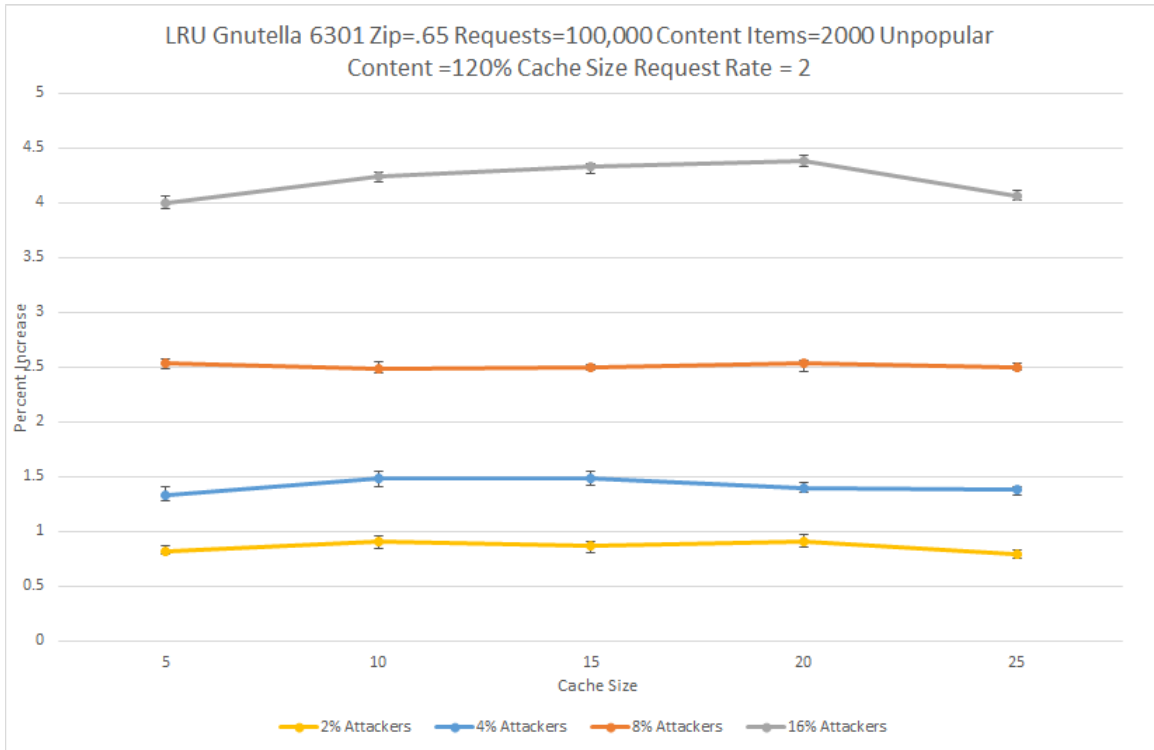
10x10 Square Graph. Random. Zipfian = 0.85. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	1.3995	0.2511	2.3995	0.3532	4.6545	0.4041	8.3624	0.5948
1	FIFO	1.9812	0.5413	3.2452	0.8945	5.5590	1.0617	8.6679	1.4303
1	Random	3.7091	1.4496	4.1899	1.6416	5.2673	1.6040	6.9364	2.1780
1 Total		**2.3633**	**1.3346**	**3.2782**	**1.3196**	**5.1603**	**1.1958**	**7.9889**	**1.7177**
⊟2	LRU	2.2282	0.3315	3.9273	0.4930	7.1111	0.7409	11.9686	0.7611
2	FIFO	3.0883	0.4860	5.1486	0.7346	8.2176	1.3925	13.4664	1.7656
2	Random	4.7116	1.6250	5.5817	1.7453	6.9418	1.6175	9.2007	1.5749
2 Total		**3.3427**	**1.4338**	**4.8859**	**1.3293**	**7.4235**	**1.4218**	**11.5452**	**2.2763**
⊟4	LRU	3.3603	0.7090	5.5590	1.2520	9.7373	1.1048	15.1280	1.6285
4	FIFO	3.7003	0.8937	5.8595	0.8838	9.8577	1.4061	16.1906	2.0791
4	Random	5.7370	2.4612	7.2260	2.7793	8.9926	3.0114	12.2652	3.0205
4 Total		**4.2659**	**1.8853**	**6.2148**	**1.9708**	**9.5292**	**2.0579**	**14.5279**	**2.8486**
Grand Total		**3.3239**	**1.7514**	**4.7930**	**1.9763**	**7.3710**	**2.3968**	**11.3540**	**3.5440**

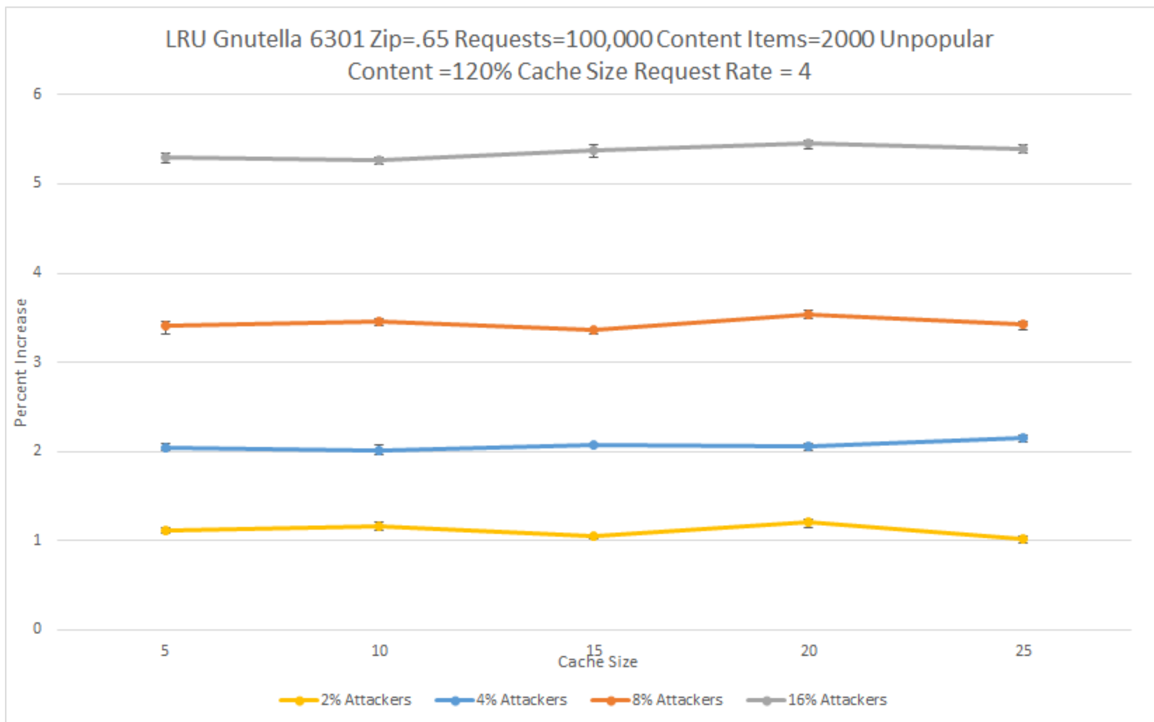
10x10 Square Graph (Zipfian=0.85): Percentage Increase Statistics



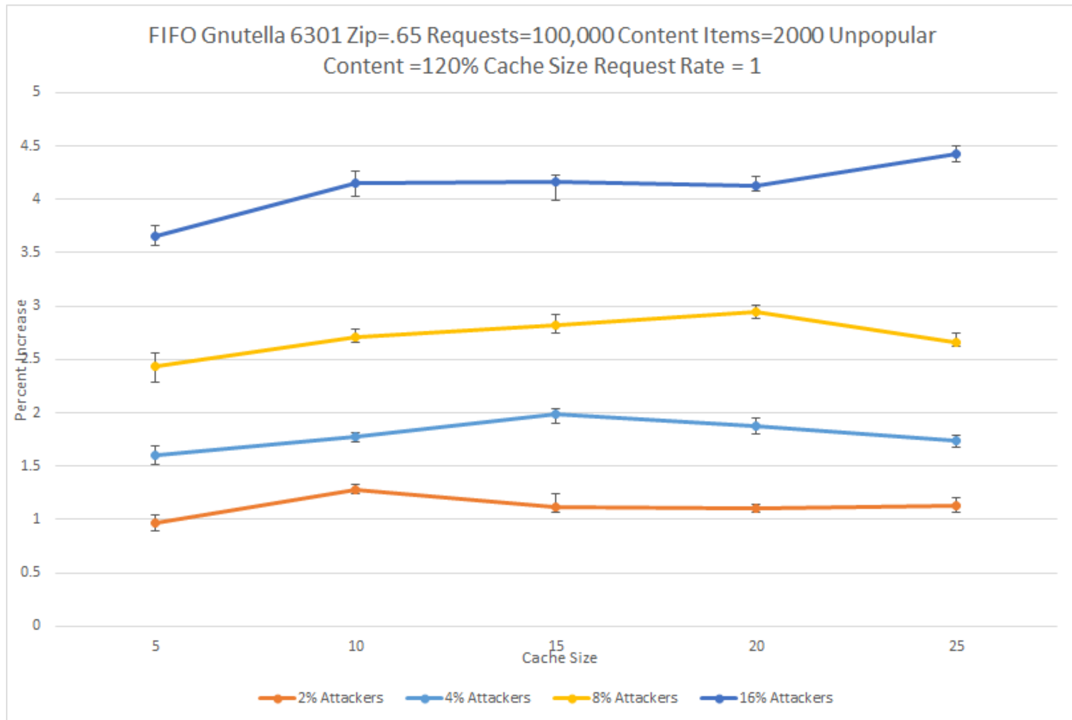
Gnutella 6301 node graph. LRU. Zipfian = 0.65. Request rate of 1



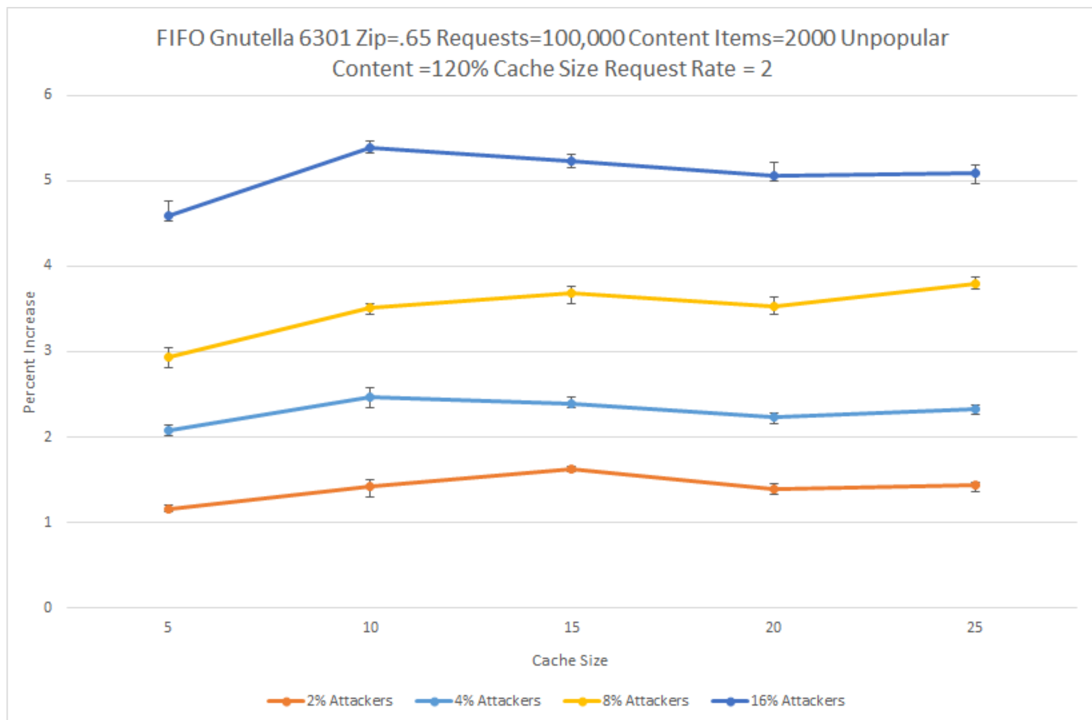
Gnutella 6301 node graph. LRU. Zipfian = 0.65. Request rate of 2



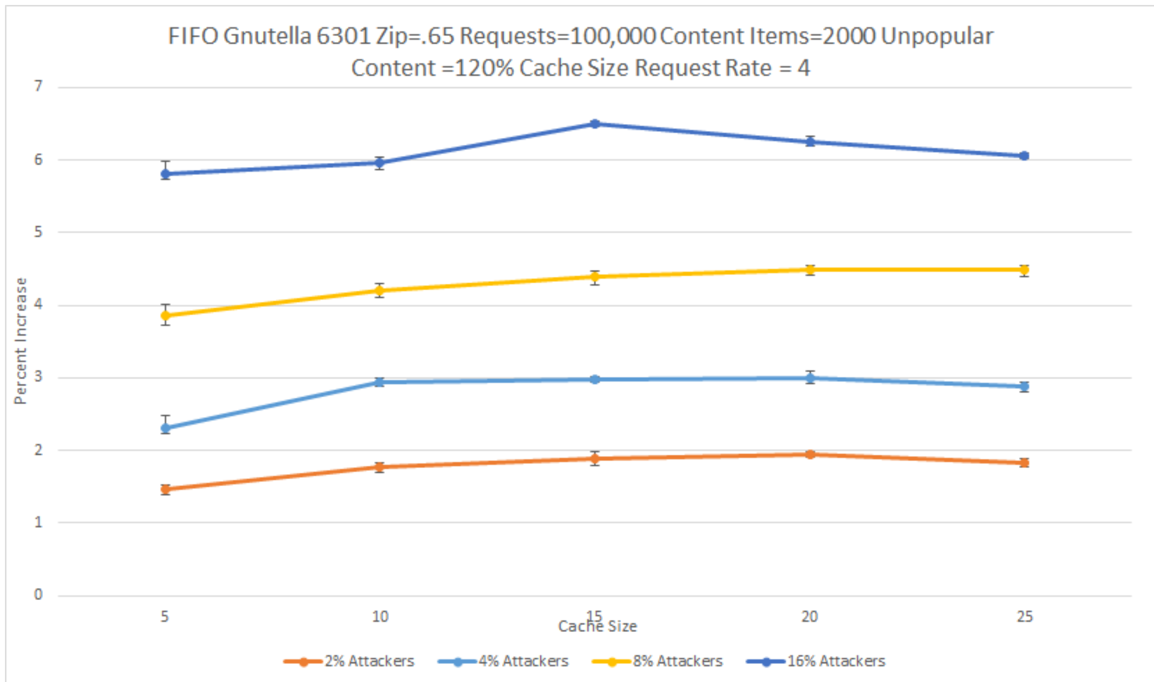
Gnutella 6301 node graph. LRU. Zipfian = 0.65. Request rate of 4



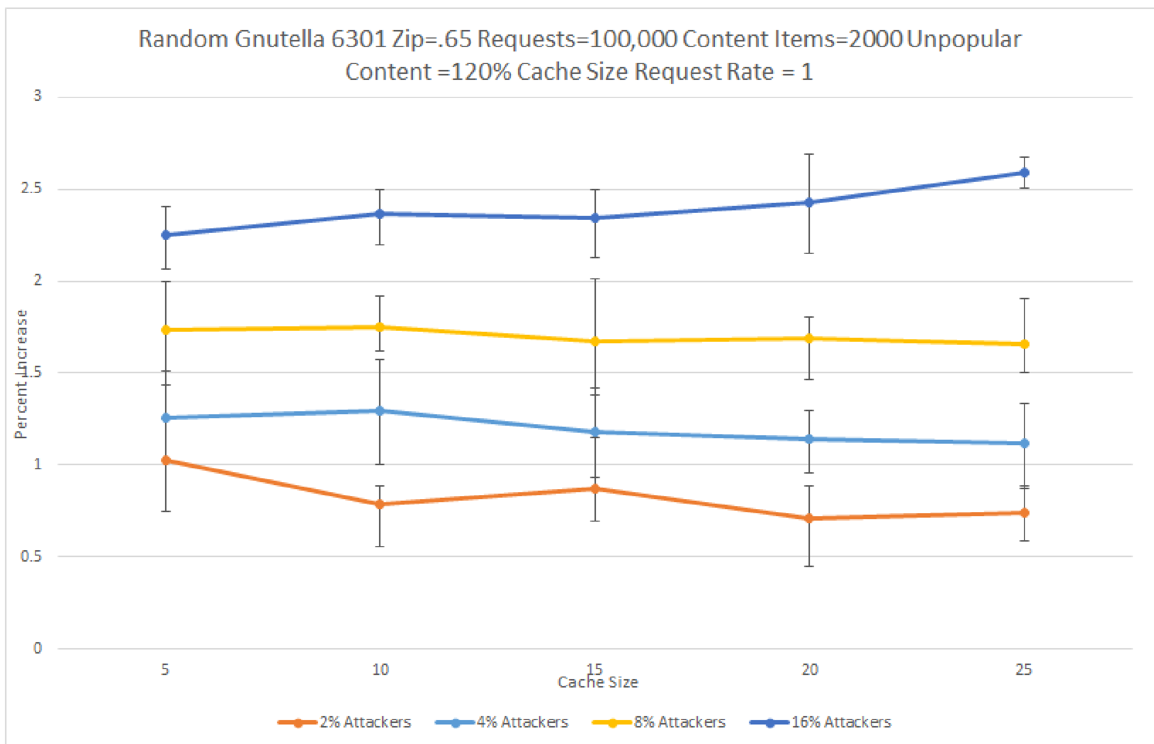
Gnutella 6301 node graph. FIFO. Zipfian = 0.65. Request rate of 1



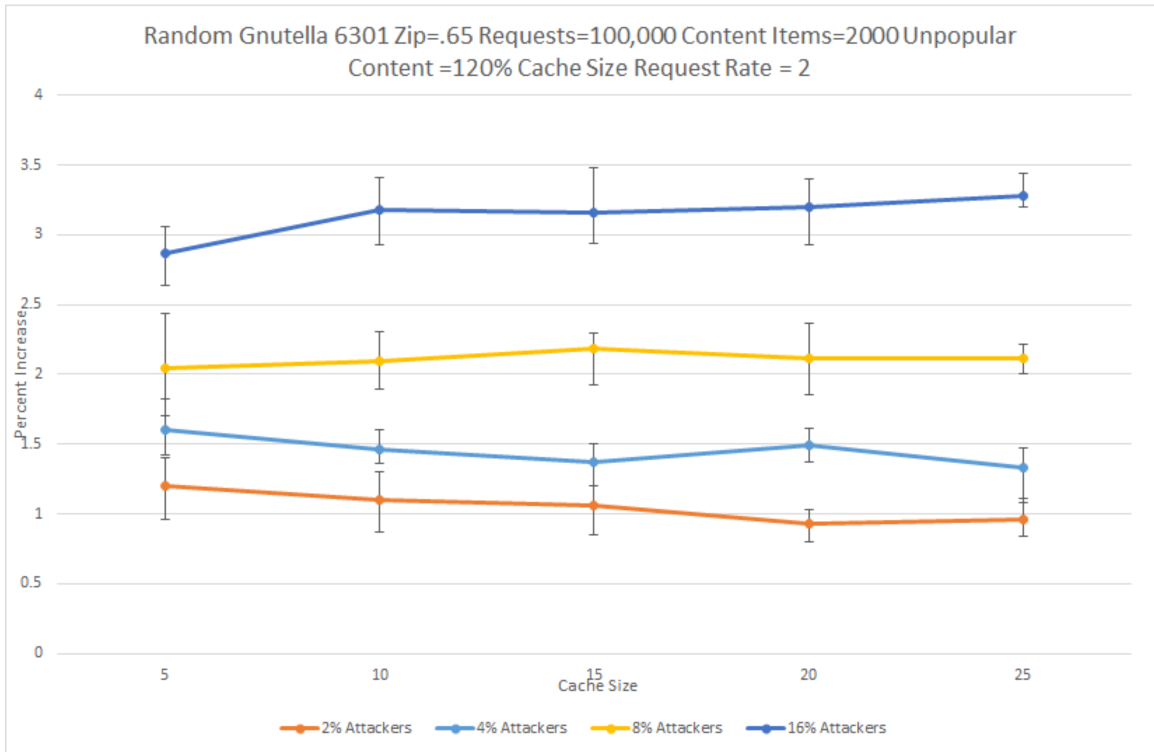
Gnutella 6301 node graph. FIFO. Zipfian = 0.65. Request rate of 2



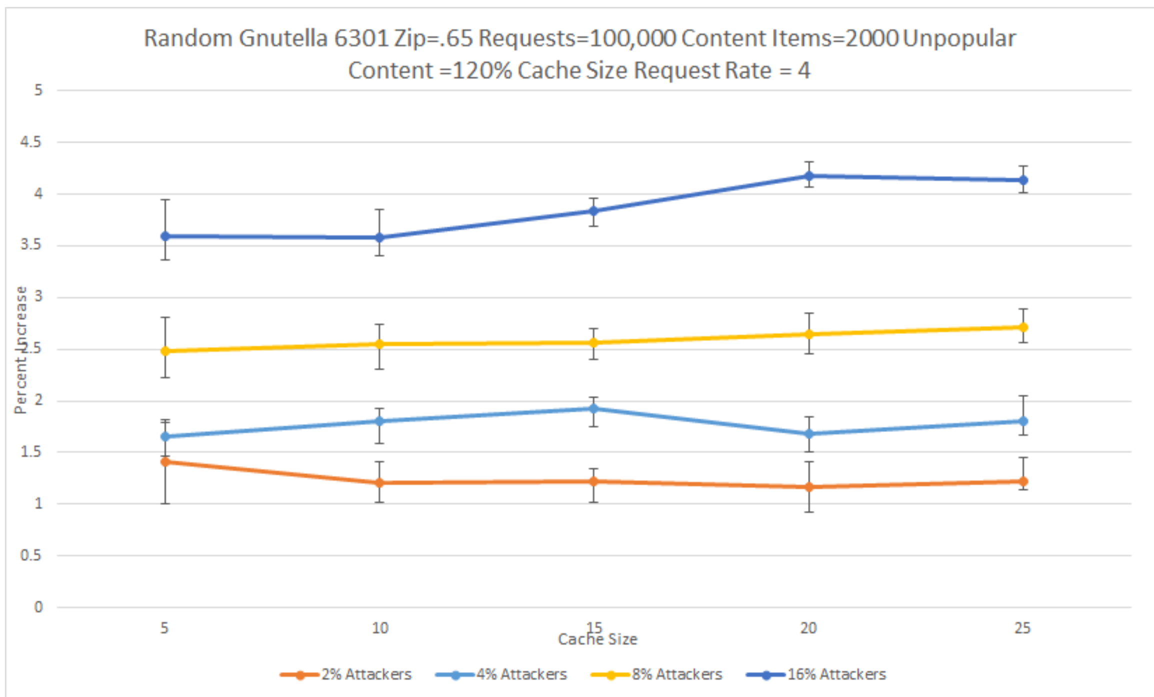
Gnutella 6301 node graph. FIFO. Zipfian = 0.65. Request rate of 4



Gnutella 6301 node graph. Random. Zipfian = 0.65. Request rate of 1



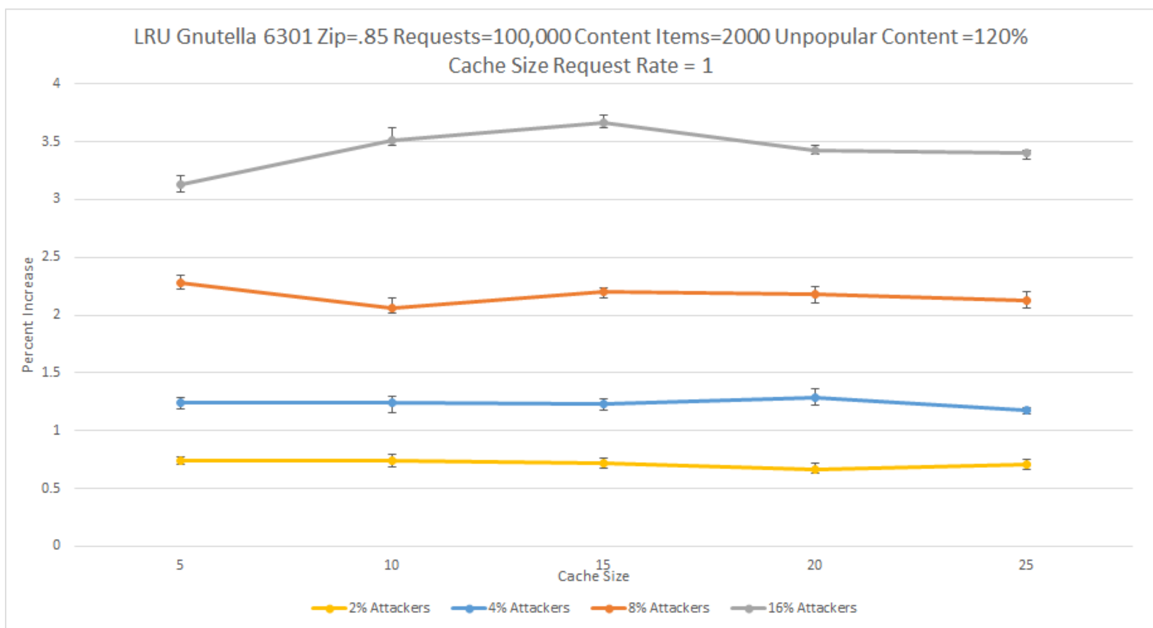
Gnutella 6301 node graph. Random. Zipfian = 0.65. Request rate of 2



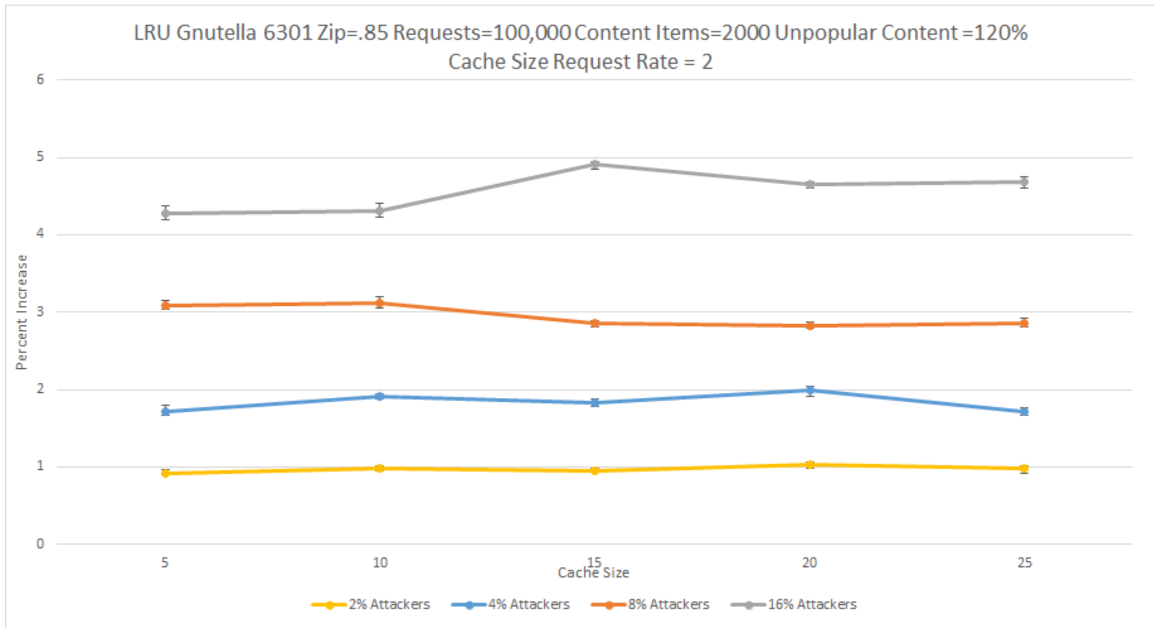
Gnutella 6301 node graph. Random. Zipfian = 0.65. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	0.6122	0.0709	1.0224	0.0515	1.8139	0.0952	3.0220	0.1516
	FIFO	1.1236	0.0972	1.7955	0.1299	2.7140	0.1696	4.1017	0.2510
	Random	0.8270	0.1139	1.1981	0.0688	1.7011	0.0354	2.3947	0.1116
1 Total		**0.8543**	**0.2305**	**1.3387**	**0.3429**	**2.0764**	**0.4674**	**3.1728**	**0.7279**
⊟2	LRU	0.8594	0.0470	1.4177	0.0587	2.5109	0.0202	4.1999	0.1512
	FIFO	1.4053	0.1464	2.2980	0.1358	3.4862	0.2925	5.0724	0.2670
	Random	1.0516	0.0974	1.4507	0.0942	2.1109	0.0456	3.1348	0.1407
2 Total		**1.1054**	**0.2493**	**1.7221**	**0.4198**	**2.7027**	**0.6025**	**4.1357**	**0.8159**
⊟4	LRU	1.1071	0.0698	2.0630	0.0467	3.4351	0.0580	5.3538	0.0681
	FIFO	1.7784	0.1684	2.8220	0.2614	4.2863	0.2365	6.1141	0.2380
	Random	1.2435	0.0855	1.7740	0.0985	2.5912	0.0823	3.8656	0.2549
4 Total		**1.3763**	**0.3122**	**2.2197**	**0.4713**	**3.4375**	**0.7077**	**5.1112**	**0.9561**
Grand Total		**1.1120**	**0.3411**	**1.7602**	**0.5496**	**2.7389**	**0.8187**	**4.1399**	**1.1530**

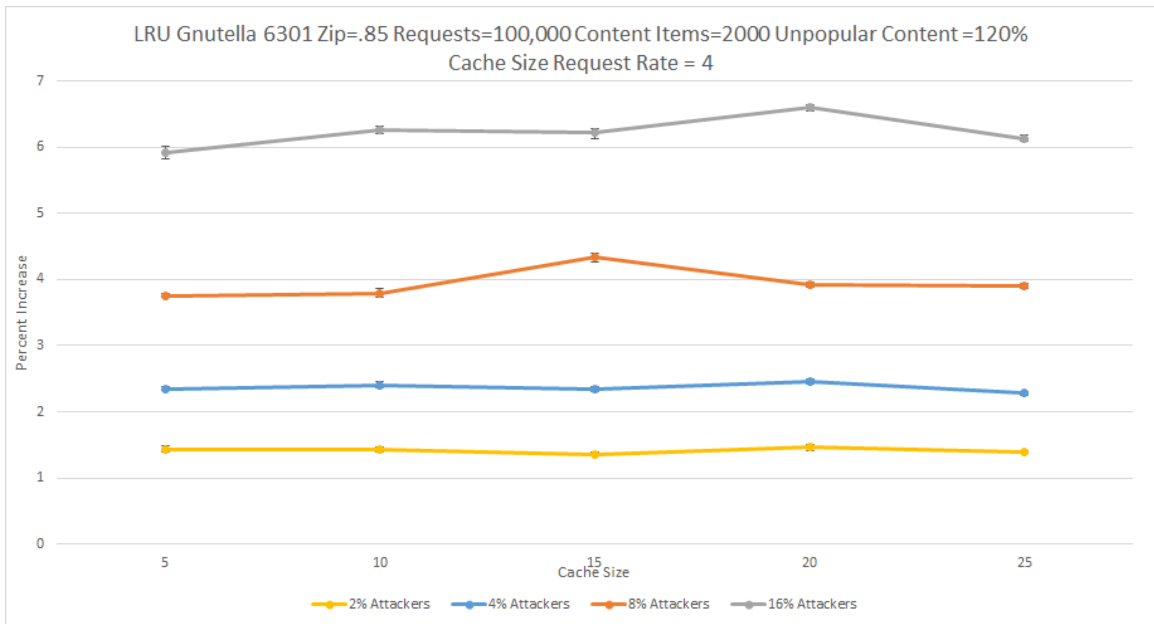
Gnutella 6301 node Graph (Zipfian=0.65): Percentage Increase Statistics



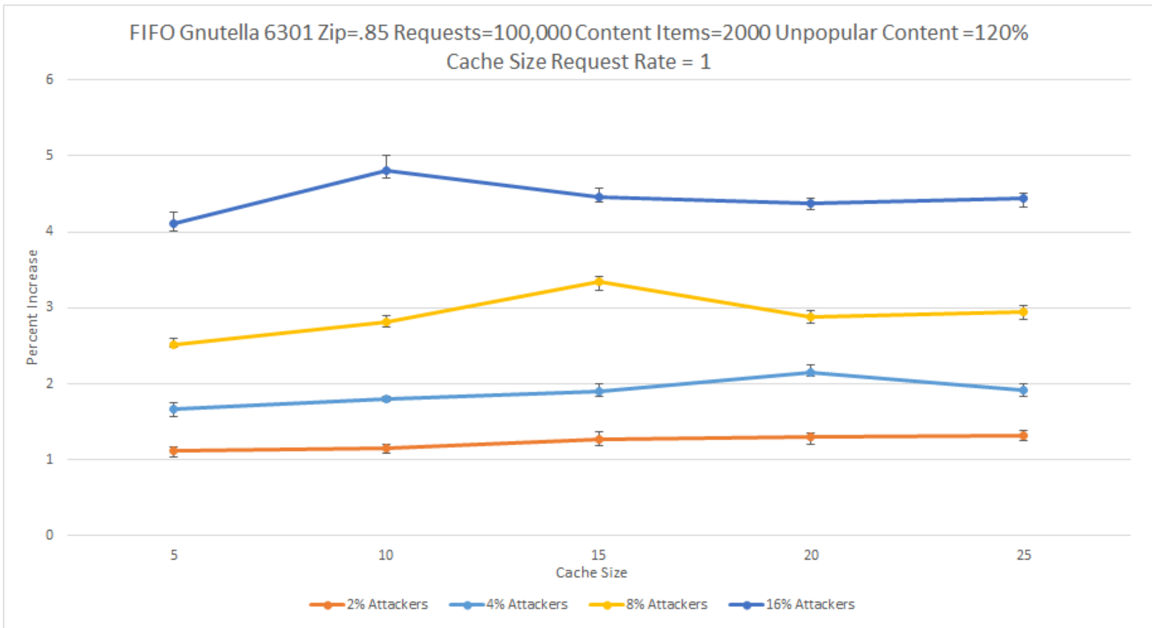
Gnutella 6301 node graph. LRU. Zipfian = 0.85. Request rate of 1



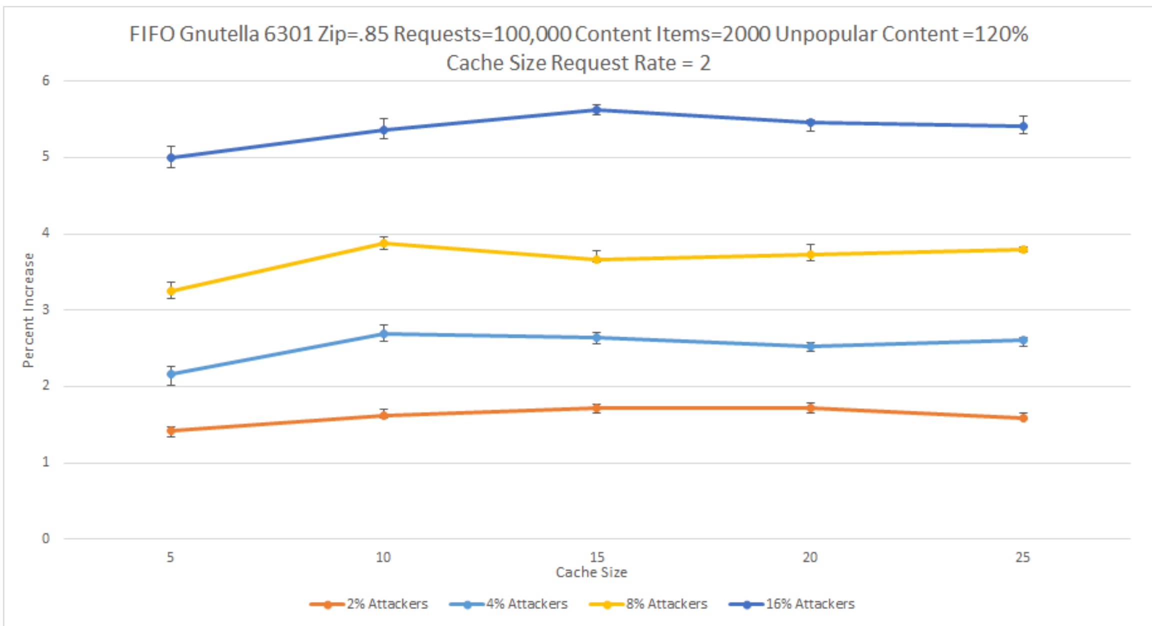
Gnutella 6301 node graph. LRU. Zipfian = 0.85. Request rate of 2



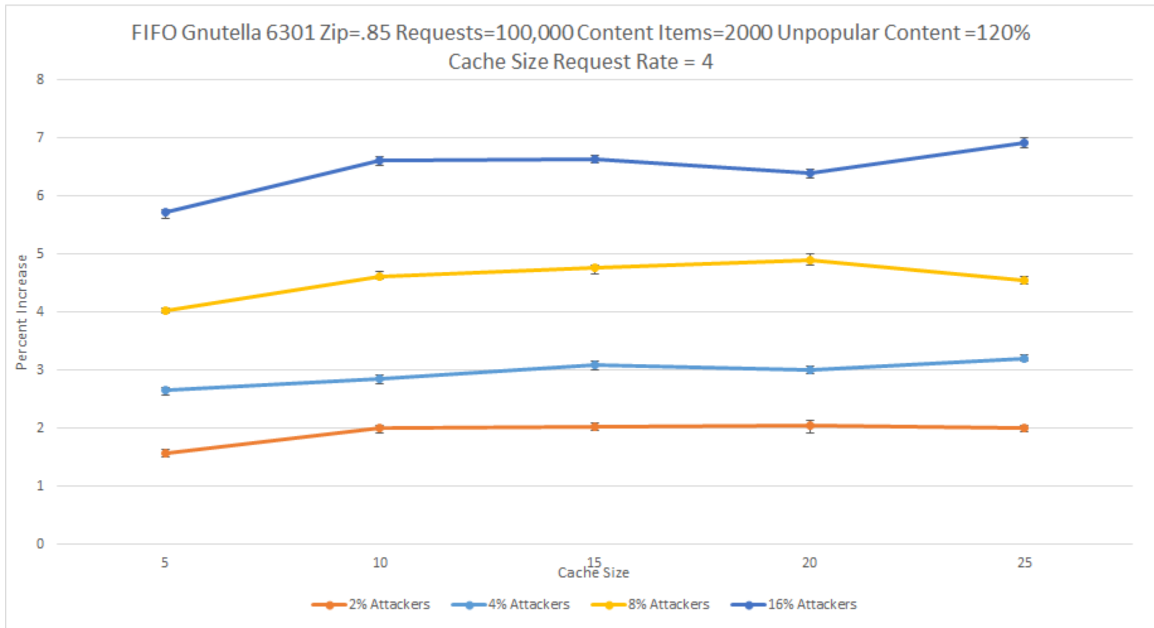
Gnutella 6301 node graph. LRU. Zipfian = 0.85. Request rate of 4



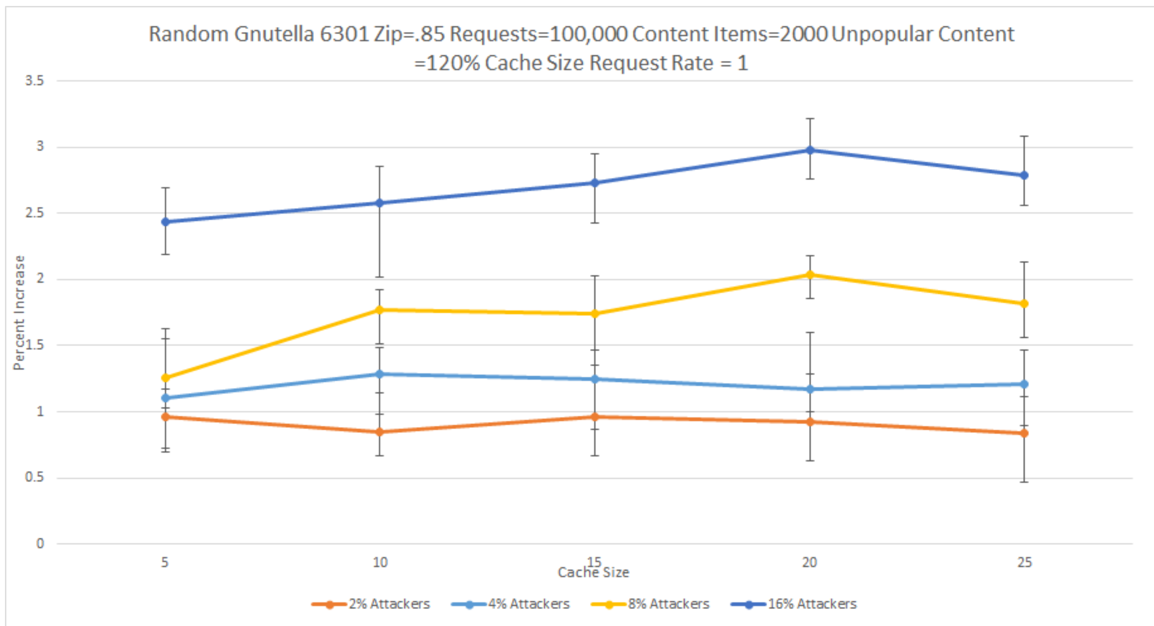
Gnutella 6301 node graph. FIFO. Zipfian = 0.85. Request rate of 1



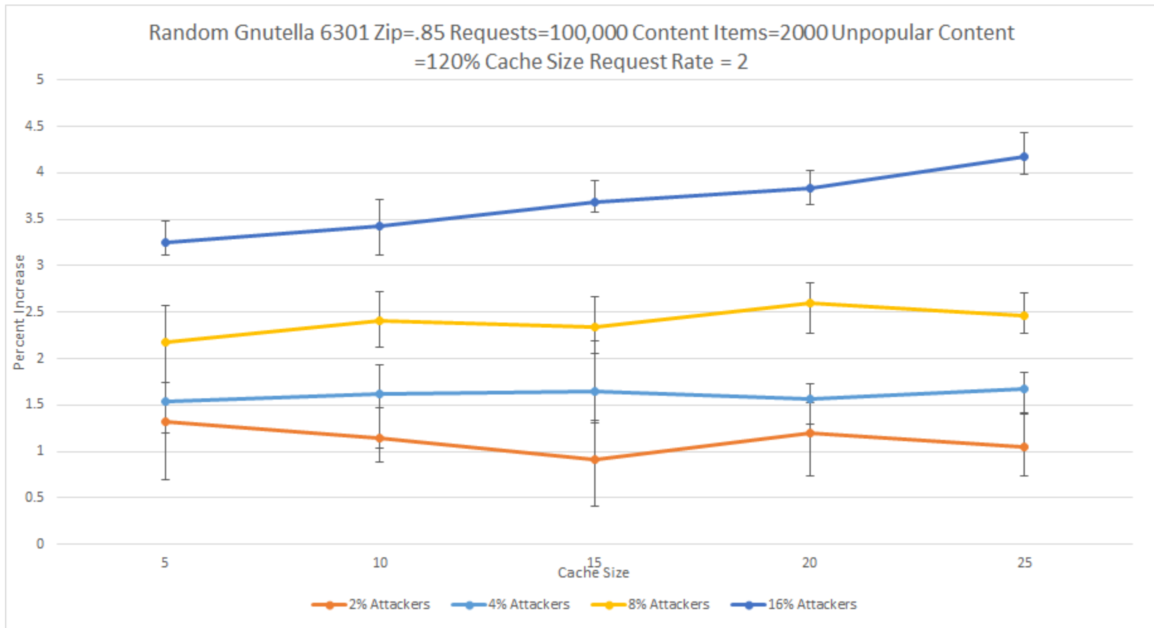
Gnutella 6301 node graph. FIFO. Zipfian = 0.85. Request rate of 2



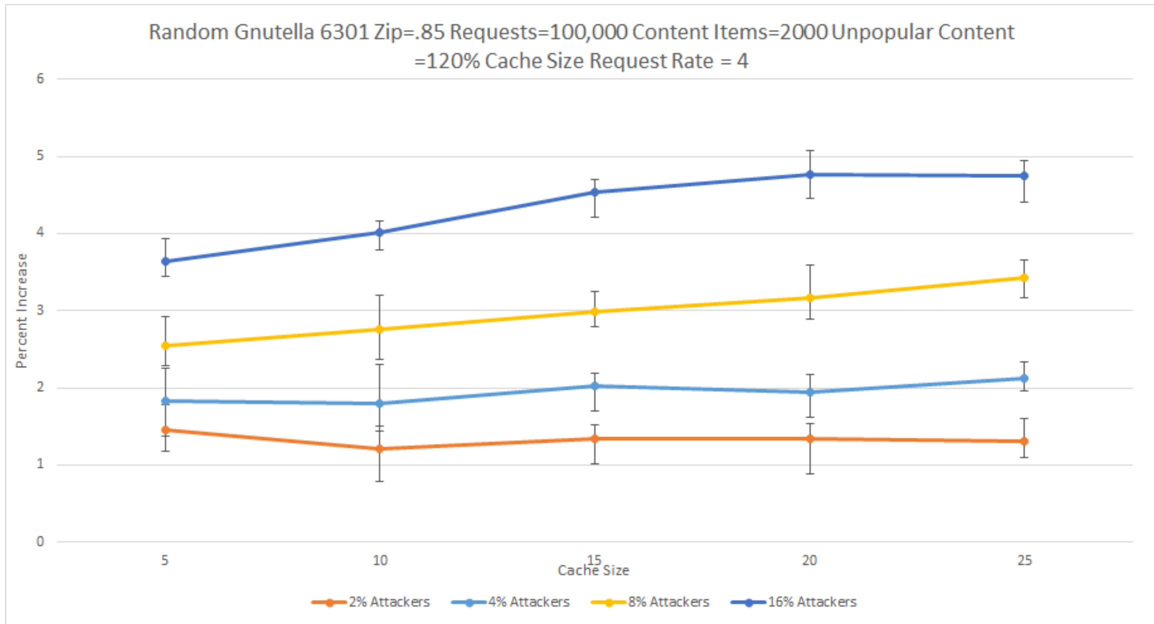
Gnutella 6301 node graph. FIFO. Zipfian = 0.85. Request rate of 4



Gnutella 6301 node graph. Random. Zipfian = 0.85. Request rate of 1



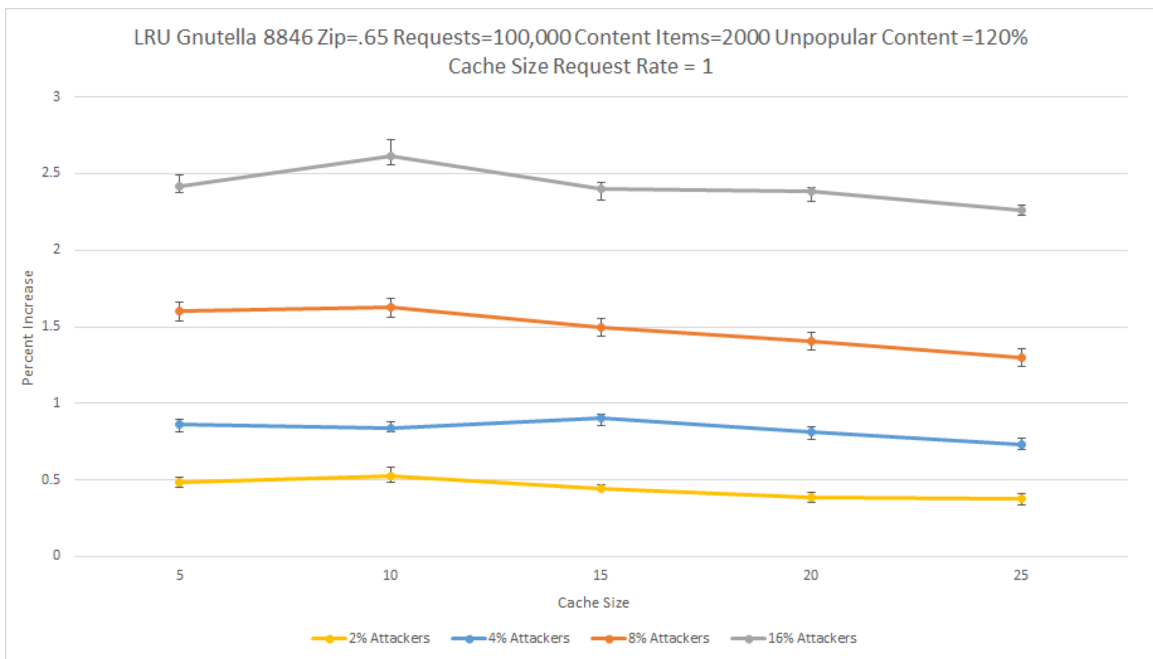
Gnutella 6301 node graph. Random. Zipfian = 0.85. Request rate of 2



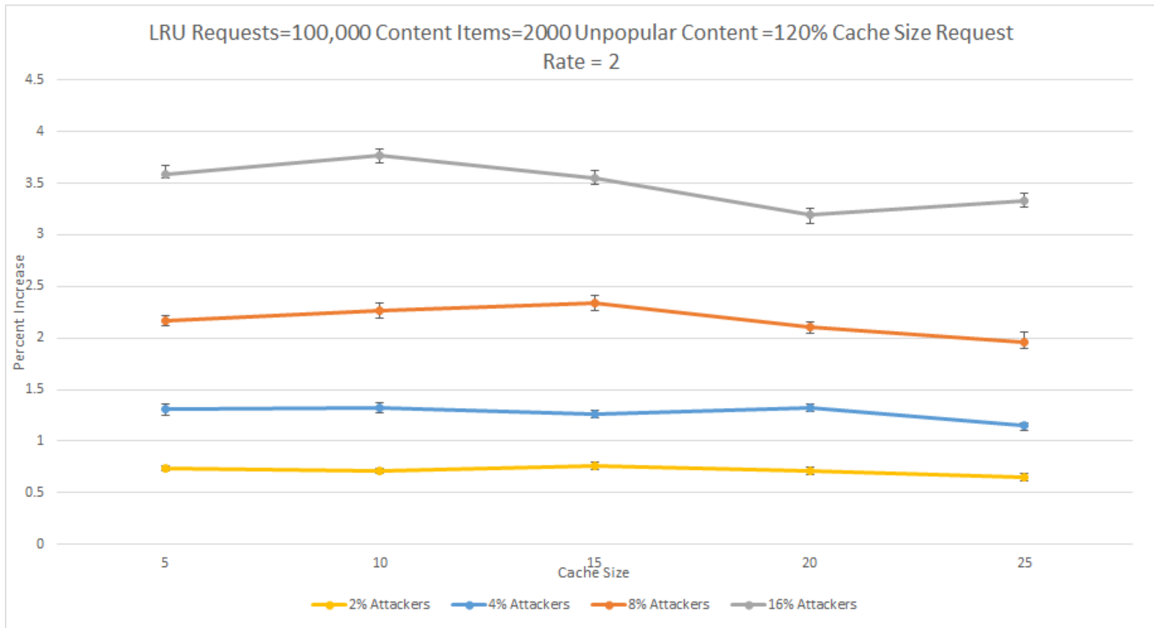
Gnutella 6301 node graph. Random. Zipfian = 0.85. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	0.7147	0.0265	1.2332	0.0351	2.1700	0.0733	3.4262	0.1736
	FIFO	1.2344	0.0836	1.8868	0.1614	2.9033	0.2647	4.4396	0.2211
	Random	0.9055	0.0512	1.2043	0.0636	1.7253	0.2580	2.7031	0.1849
1 Total		**0.9515**	**0.2225**	**1.4415**	**0.3313**	**2.2662**	**0.5322**	**3.5230**	**0.7382**
⊟2	LRU	0.9715	0.0366	1.8333	0.1072	2.9465	0.1297	4.5640	0.2413
	FIFO	1.6142	0.1095	2.5221	0.1891	3.6611	0.2145	5.3675	0.2055
	Random	1.1270	0.1393	1.6085	0.0506	2.3951	0.1359	3.6767	0.3192
2 Total		**1.2376**	**0.2930**	**1.9880**	**0.4095**	**3.0009**	**0.5438**	**4.5361**	**0.7378**
⊟4	LRU	1.4159	0.0370	2.3659	0.0611	3.9437	0.2069	6.2275	0.2199
	FIFO	1.9260	0.1841	2.9640	0.1923	4.5722	0.2959	6.4503	0.4033
	Random	1.3273	0.0766	1.9415	0.1220	2.9783	0.3060	4.3379	0.4386
4 Total		**1.5564**	**0.2887**	**2.4238**	**0.4410**	**3.8314**	**0.7102**	**5.6719**	**1.0161**
Grand Total		**1.2485**	**0.3660**	**1.9511**	**0.5646**	**3.0328**	**0.8774**	**4.5770**	**1.2156**

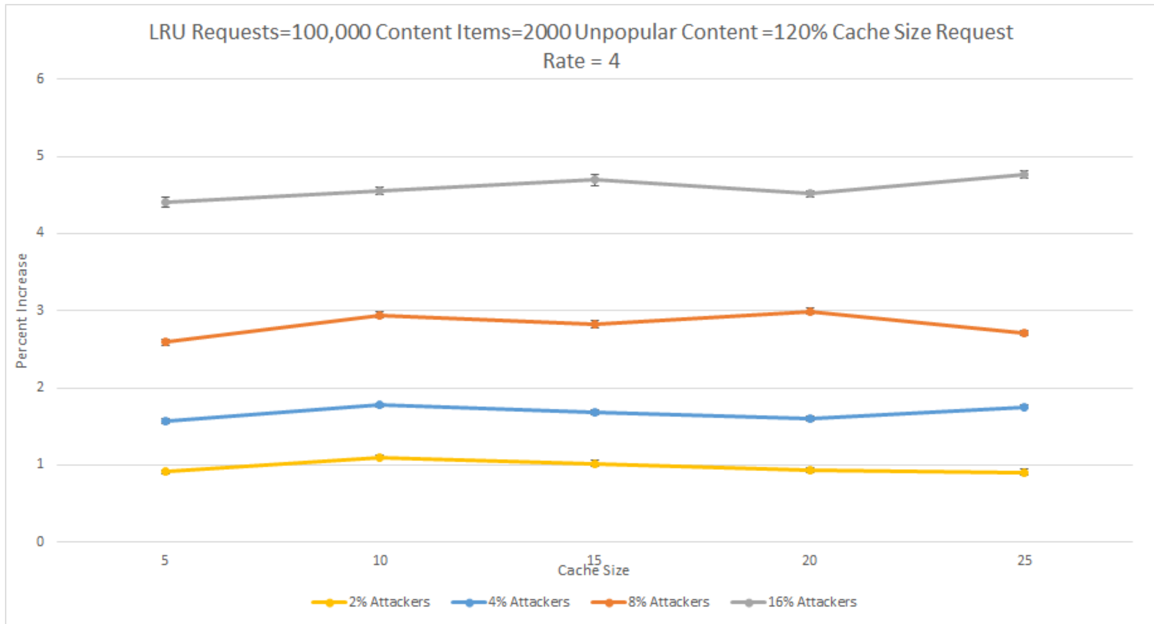
Gnutella 6301 node Graph (Zipfian=0.85): Percentage Increase Statistics



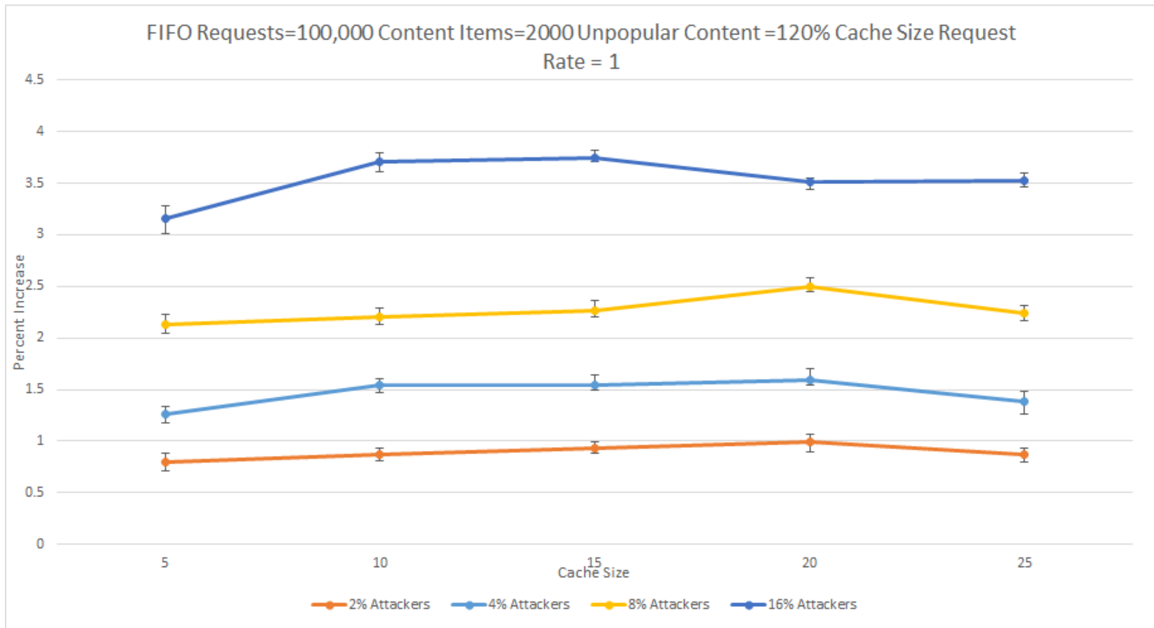
Gnutella 8846 node graph. LRU. Zipfian = 0.65. Request rate of 1



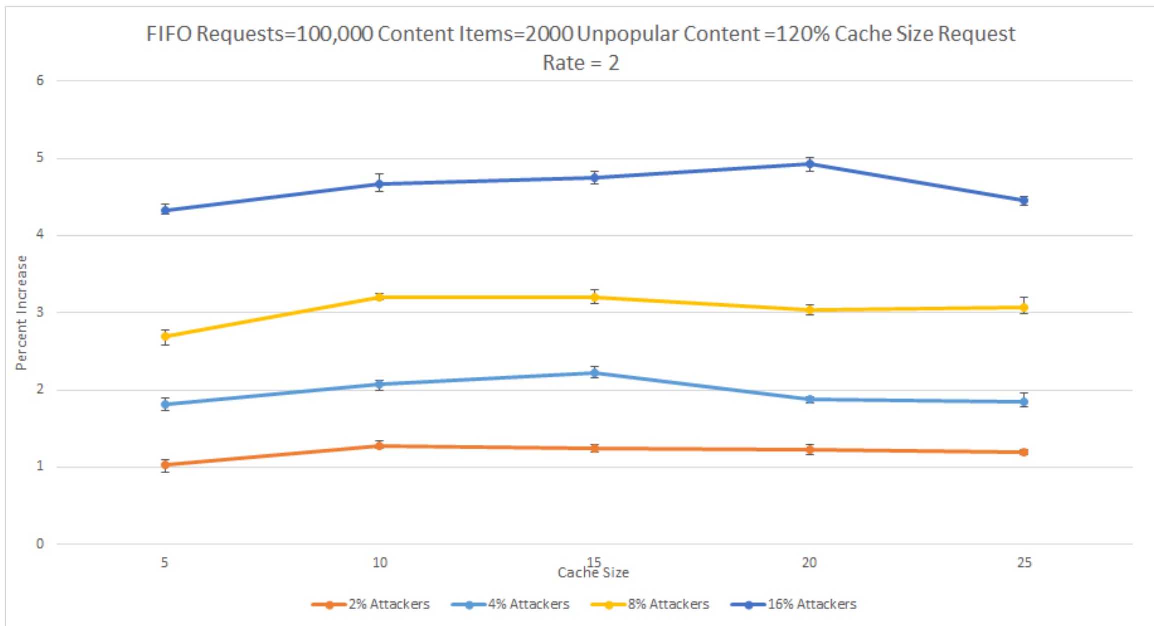
Gnutella 8846 node graph. LRU. Zipfian = 0.65. Request rate of 2



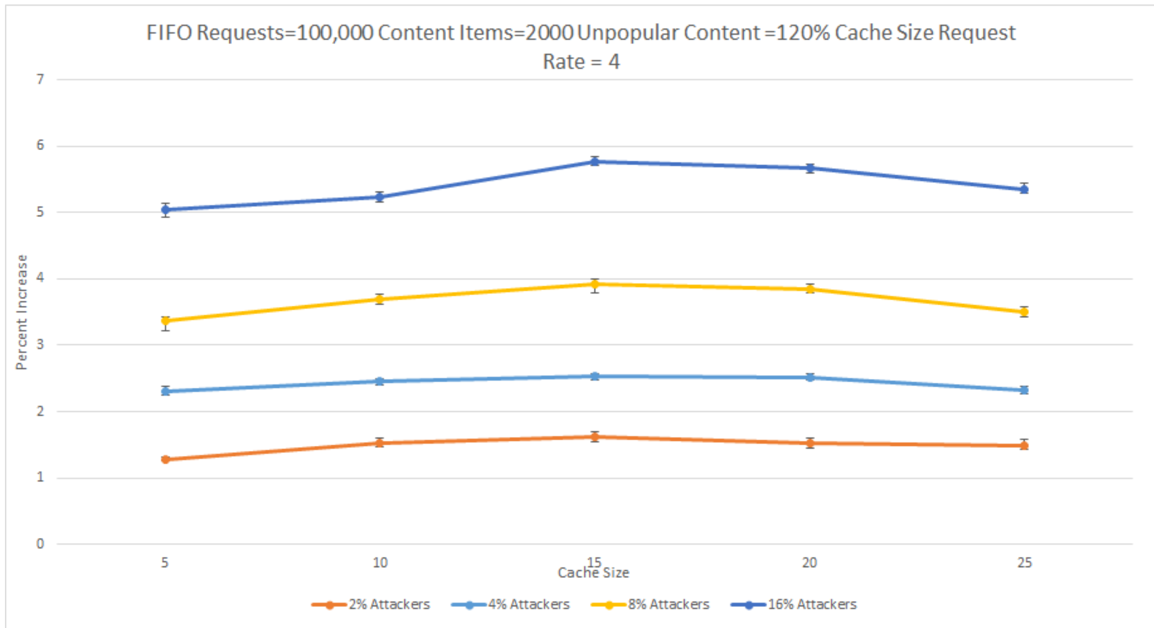
Gnutella 8846 node graph. LRU. Zipfian = 0.65. Request rate of 4



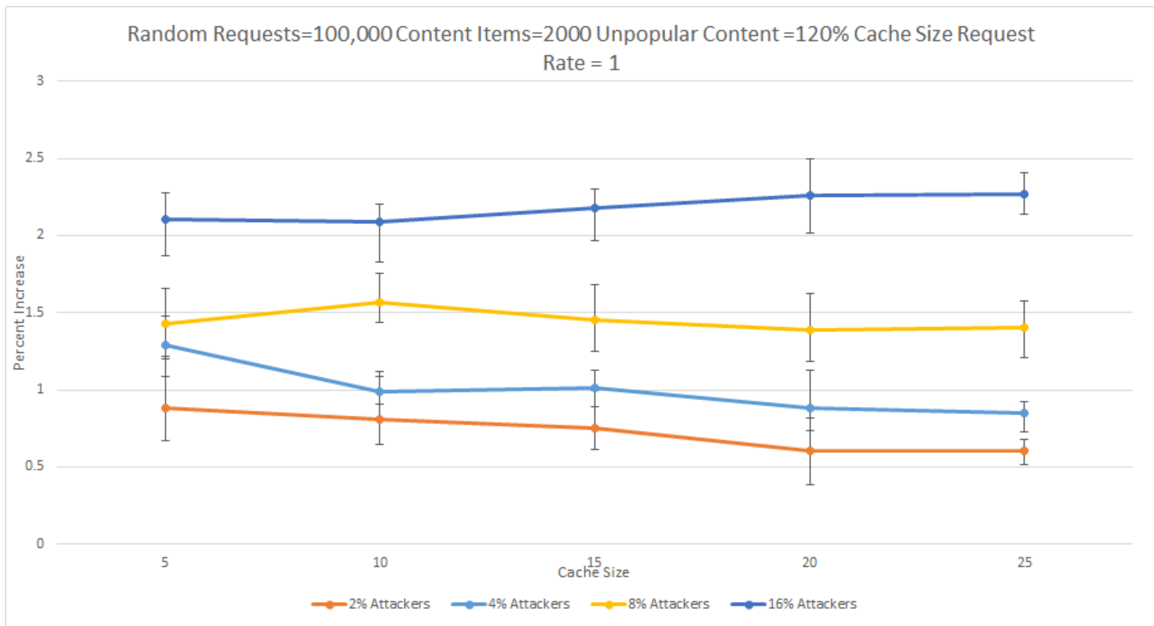
Gnutella 8846 node graph. FIFO. Zipfian = 0.65. Request rate of 1



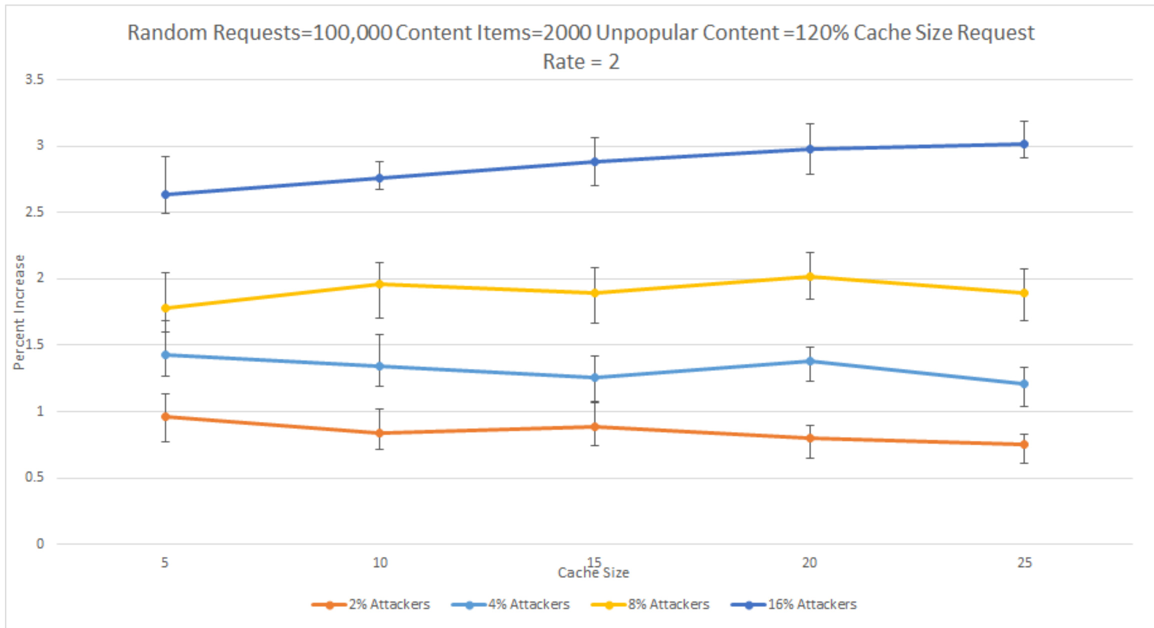
Gnutella 8846 node graph. FIFO. Zipfian = 0.65. Request rate of 2



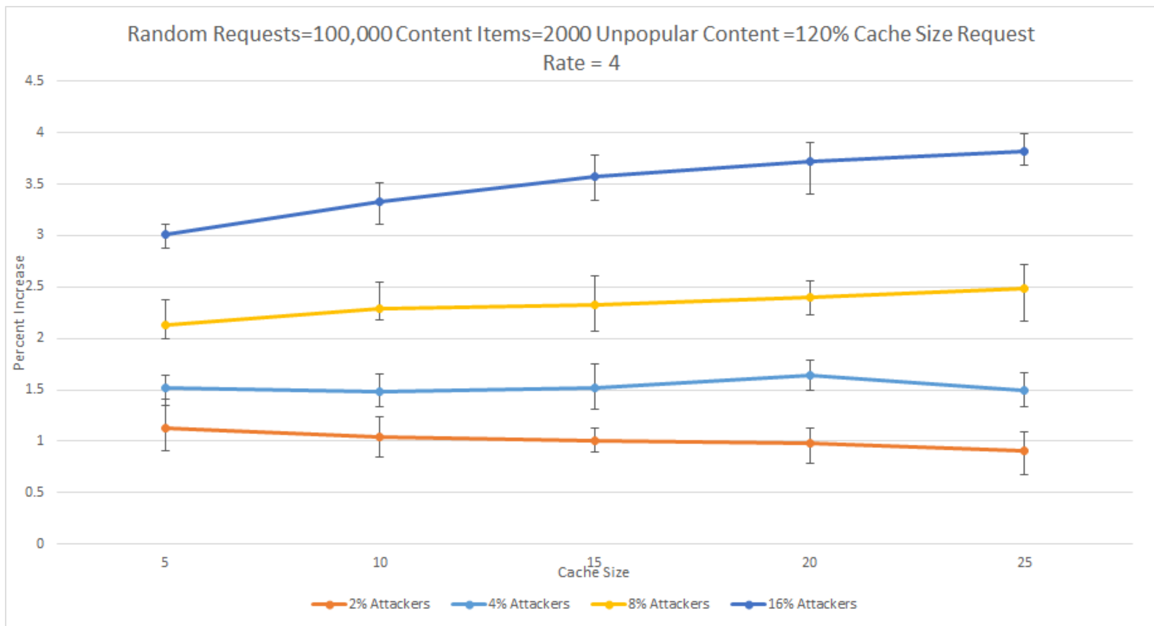
Gnutella 8846 node graph. FIFO. Zipfian = 0.65. Request rate of 4



Gnutella 8846 node graph. Random. Zipfian = 0.65. Request rate of 1



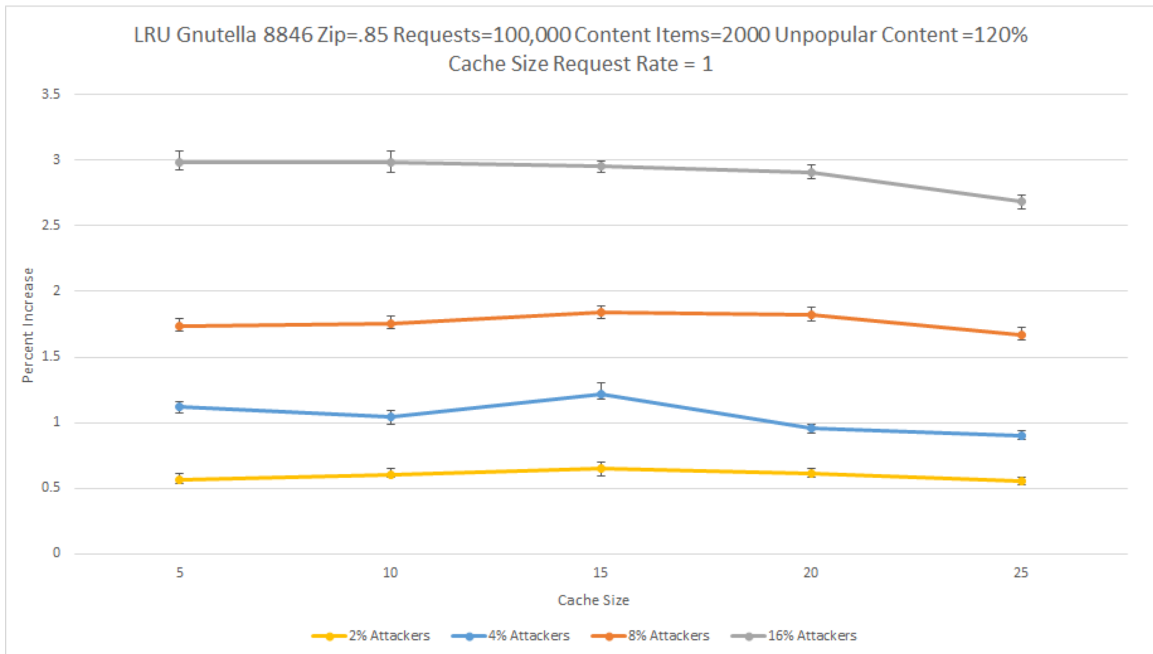
Gnutella 8846 node graph. Random. Zipfian = 0.65. Request rate of 2



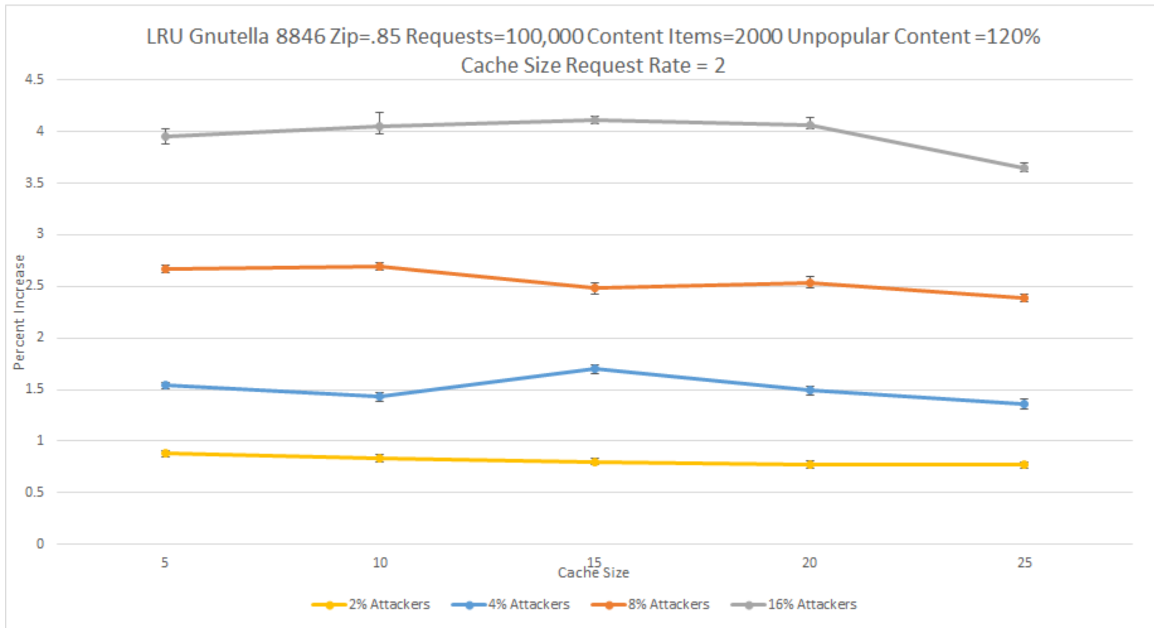
Gnutella 8846 node graph. Random. Zipfian = 0.65. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	0.4441	0.0575	0.8310	0.0572	1.4857	0.1225	2.4147	0.1155
	FIFO	0.8939	0.0685	1.4670	0.1234	2.2687	0.1196	3.5280	0.2095
	Random	0.7301	0.1098	1.0044	0.1551	1.4478	0.0629	2.1777	0.0742
1 Total		**0.6894**	**0.2031**	**1.1008**	**0.2937**	**1.7341**	**0.3928**	**2.7068**	**0.6062**
⊟2	LRU	0.7130	0.0377	1.2734	0.0666	2.1661	0.1312	3.4845	0.1998
	FIFO	1.1960	0.0858	1.9668	0.1548	3.0387	0.1828	4.6194	0.2129
	Random	0.8487	0.0699	1.3241	0.0770	1.9098	0.0798	2.8567	0.1407
2 Total		**0.9193**	**0.2143**	**1.5214**	**0.3332**	**2.3715**	**0.5025**	**3.6535**	**0.7531**
⊟4	LRU	0.9715	0.0785	1.6770	0.0781	2.8111	0.1438	4.5888	0.1307
	FIFO	1.4867	0.1110	2.4278	0.0921	3.6657	0.2104	5.4114	0.2733
	Random	1.0122	0.0696	1.5294	0.0583	2.3276	0.1183	3.4884	0.2909
4 Total		**1.1568**	**0.2500**	**1.8781**	**0.4009**	**2.9348**	**0.5766**	**4.4962**	**0.8243**
Grand Total		**0.9218**	**0.2938**	**1.5001**	**0.4693**	**2.3468**	**0.6979**	**3.6189**	**1.0355**

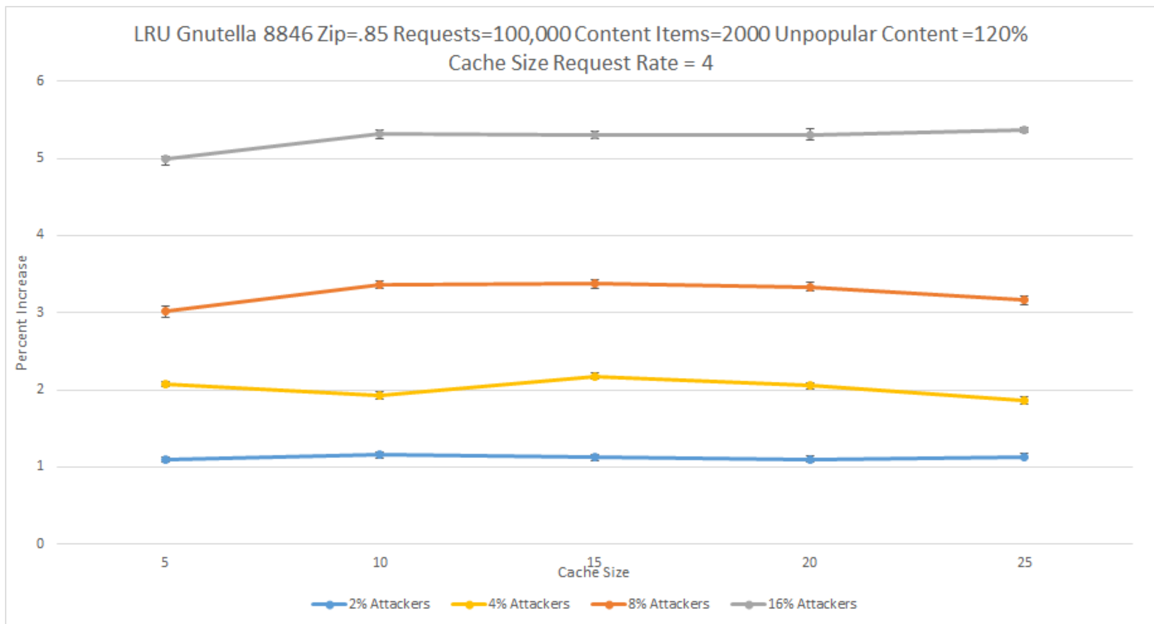
Gnutella 8846 node Graph (Zipfian=0.65): Percentage Increase Statistics



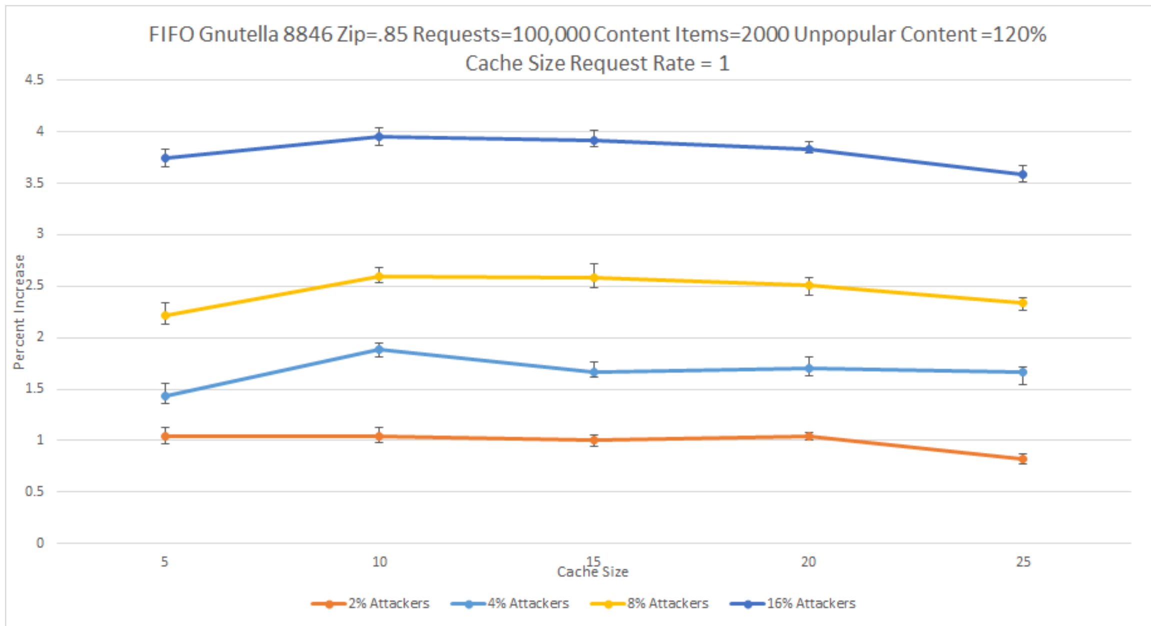
Gnutella 8846 node graph. LRU. Zipfian = 0.85. Request rate of 1



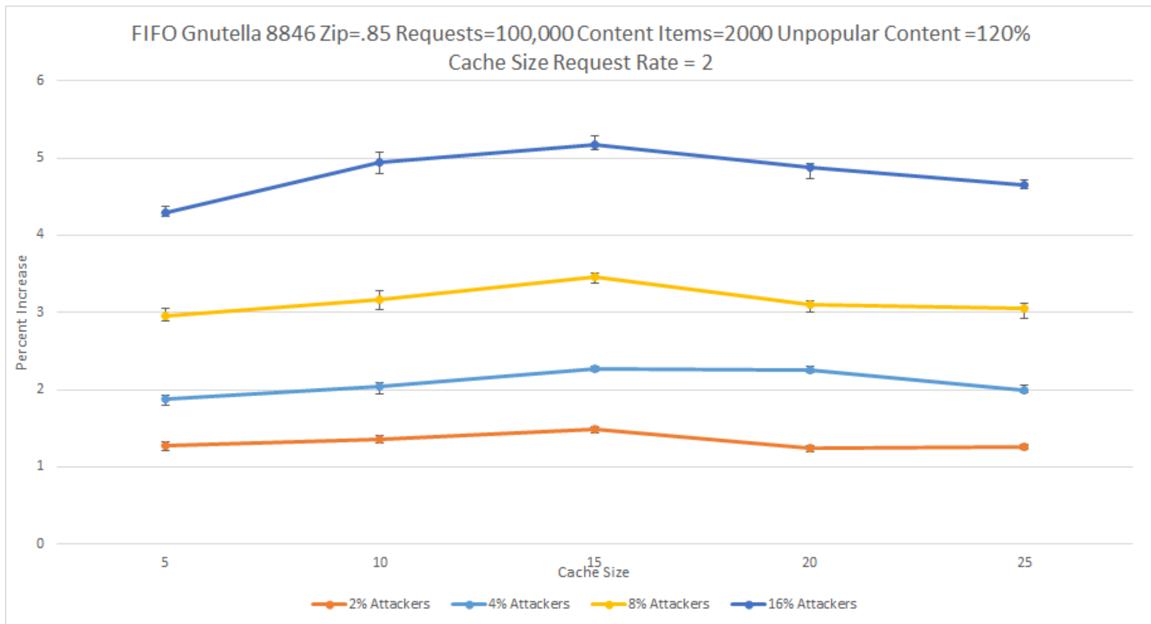
Gnutella 8846 node graph. LRU. Zipfian = 0.85. Request rate of 2



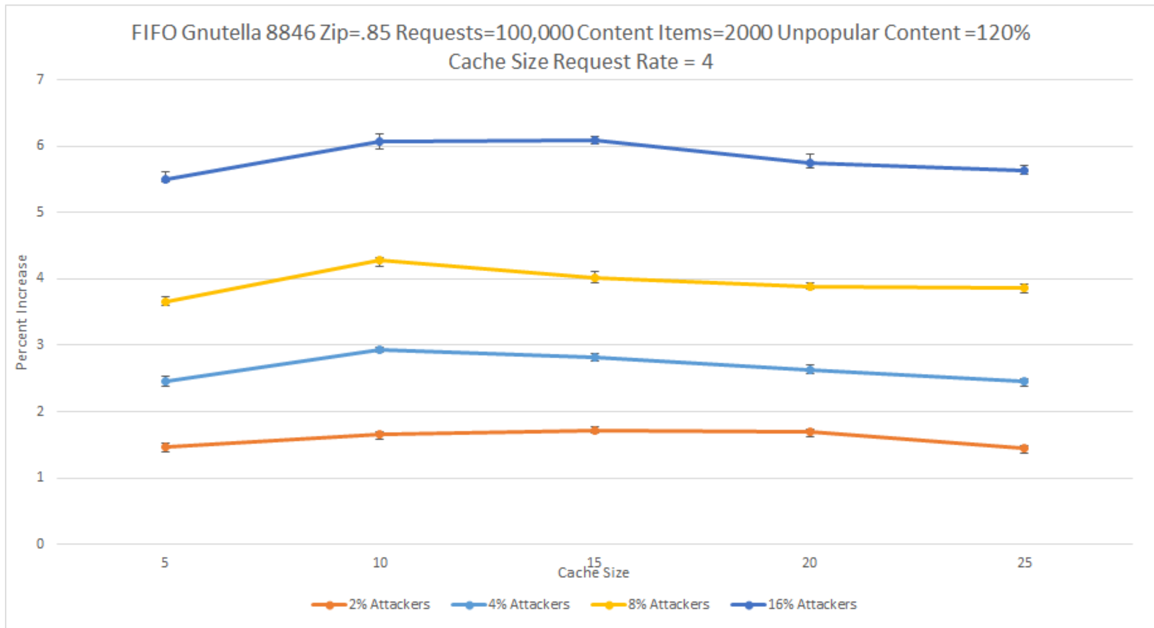
Gnutella 8846 node graph. LRU. Zipfian = 0.85. Request rate of 4



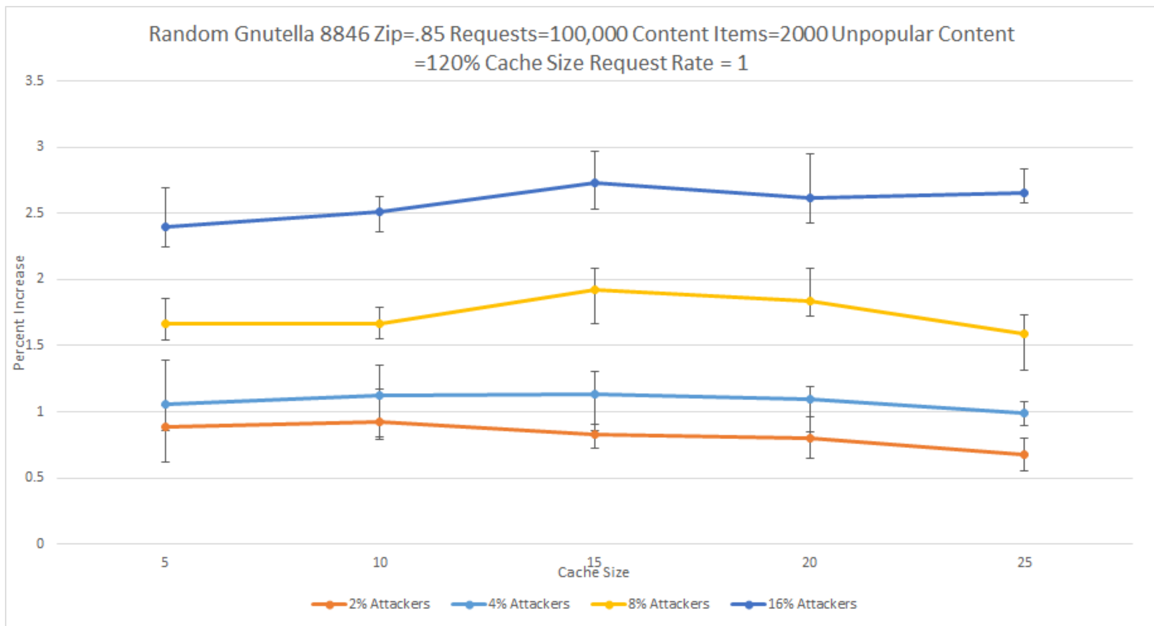
Gnutella 8846 node graph. FIFO. Zipfian = 0.85. Request rate of 1



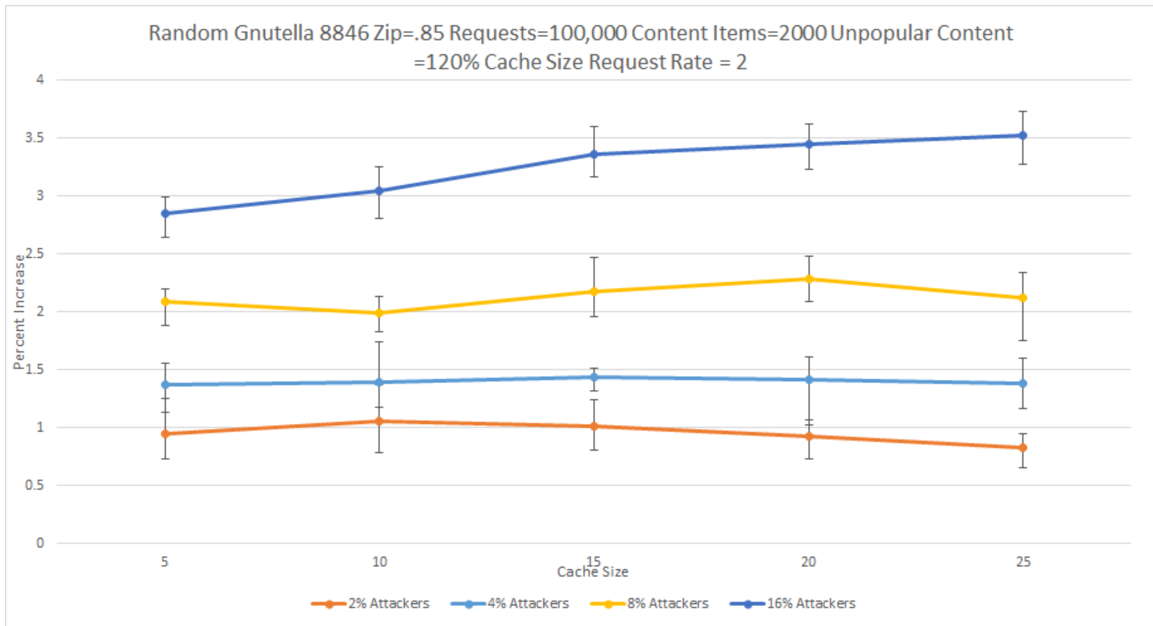
Gnutella 8846 node graph. FIFO. Zipfian = 0.85. Request rate of 2



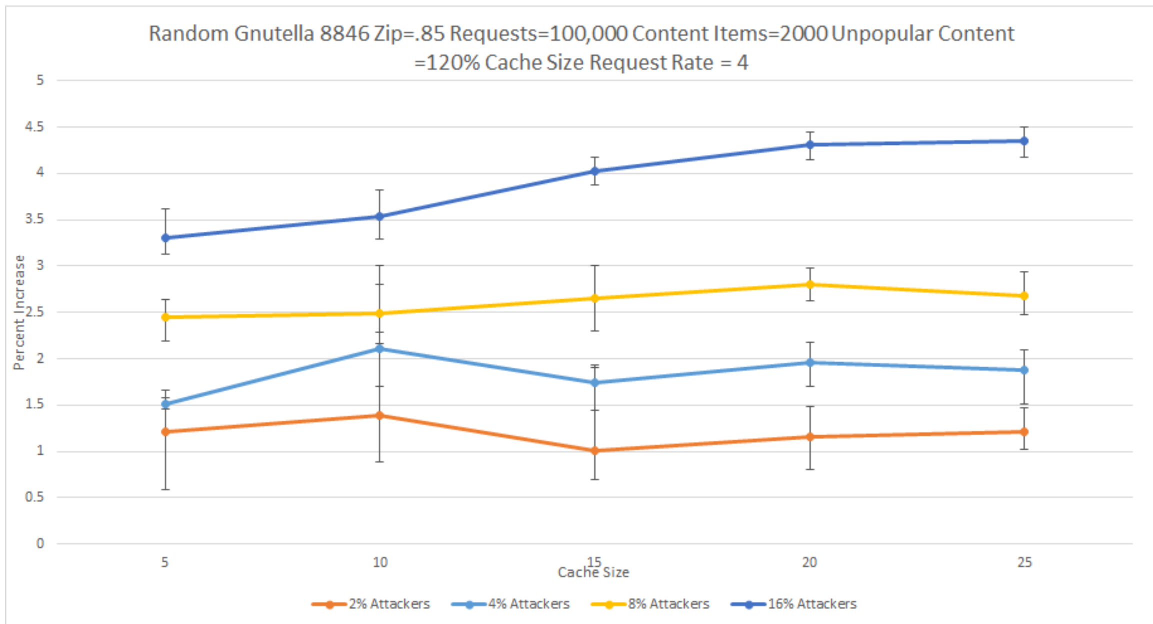
Gnutella 8846 node graph. FIFO. Zipfian = 0.85. Request rate of 4



Gnutella 8846 node graph. Random. Zipfian = 0.85. Request rate of 1



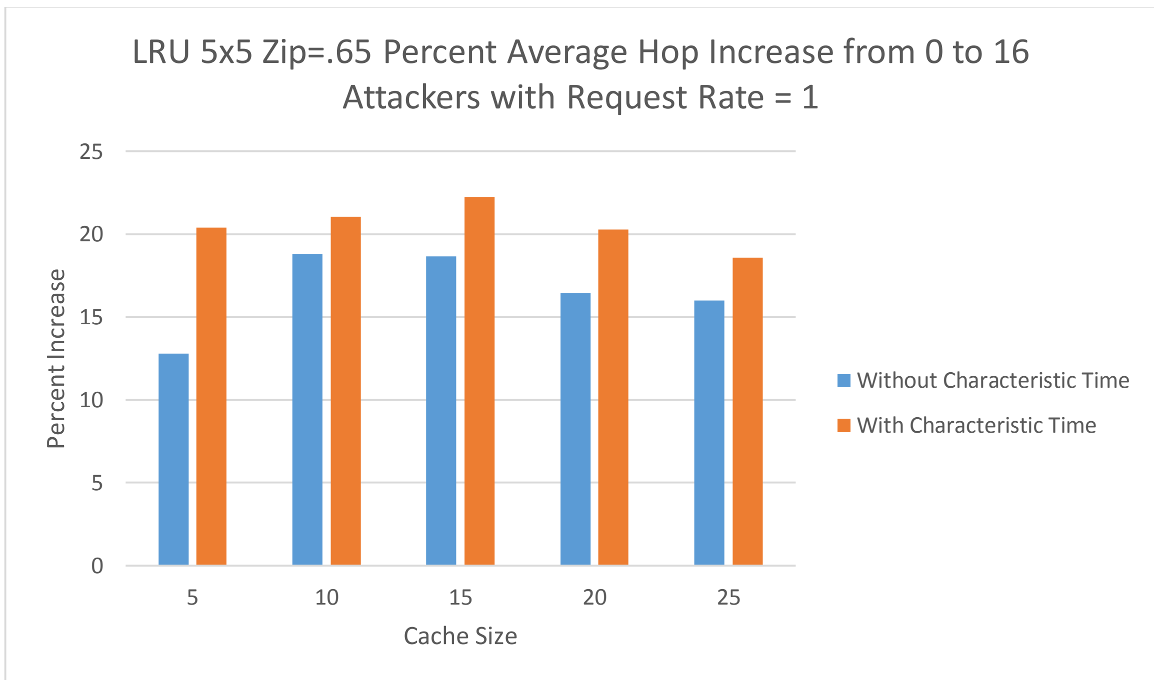
Gnutella 8846 node graph. Random. Zipfian = 0.85. Request rate of 2



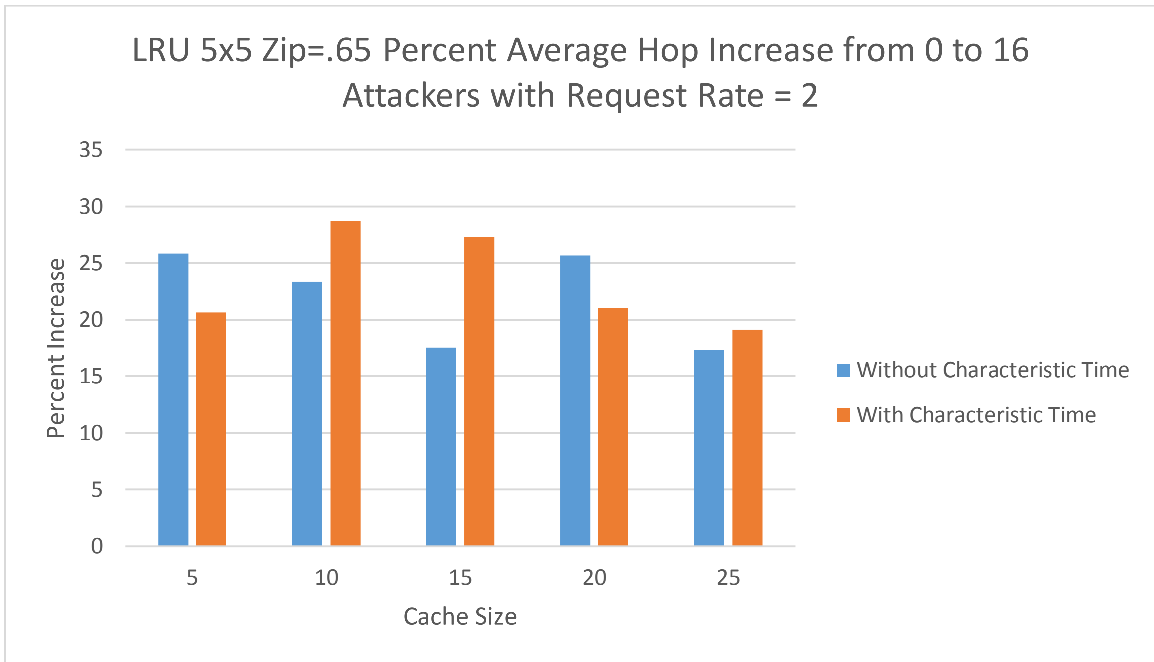
Gnutella 8846 node graph. Random. Zipfian = 0.85. Request rate of 4

Rate	Cache Type	Mean: 2% Attackers	Std. Deviation: 2% Attackers	Mean: 4% Attackers	Std. Deviation: 4% Attackers	Mean: 8% Attackers	Std. Deviation: 8% Attackers	Mean: 16% Attackers	Std. Deviation: 16% Attackers
⊟1	LRU	0.5989	0.0354	1.0474	0.1148	1.7681	0.0616	2.9024	0.1139
	FIFO	0.9921	0.0861	1.6693	0.1430	2.4470	0.1482	3.8083	0.1292
	Random	0.8219	0.0845	1.0798	0.0510	1.7369	0.1227	2.5843	0.1165
1 Total		**0.8043**	**0.1766**	**1.2655**	**0.3062**	**1.9840**	**0.3478**	**3.0983**	**0.5323**
⊟2	LRU	0.8118	0.0421	1.5051	0.1166	2.5528	0.1162	3.9660	0.1673
	FIFO	1.3209	0.0920	2.0861	0.1544	3.1439	0.1690	4.7873	0.2969
	Random	0.9545	0.0800	1.4013	0.0224	2.1331	0.0957	3.2426	0.2557
2 Total		**1.0290**	**0.2270**	**1.6642**	**0.3216**	**2.6100**	**0.4347**	**3.9986**	**0.6773**
⊟4	LRU	1.1214	0.0212	2.0190	0.1105	3.2498	0.1362	5.2524	0.1345
	FIFO	1.5984	0.1144	2.6542	0.1918	3.9424	0.2050	5.8114	0.2325
	Random	1.1941	0.1255	1.8373	0.2001	2.6122	0.1302	3.9064	0.4172
4 Total		**1.3046**	**0.2319**	**2.1702**	**0.3903**	**3.2681**	**0.5665**	**4.9901**	**0.8493**
Grand Total		**1.0460**	**0.2956**	**1.7000**	**0.5036**	**2.6207**	**0.6965**	**4.0290**	**1.0415**

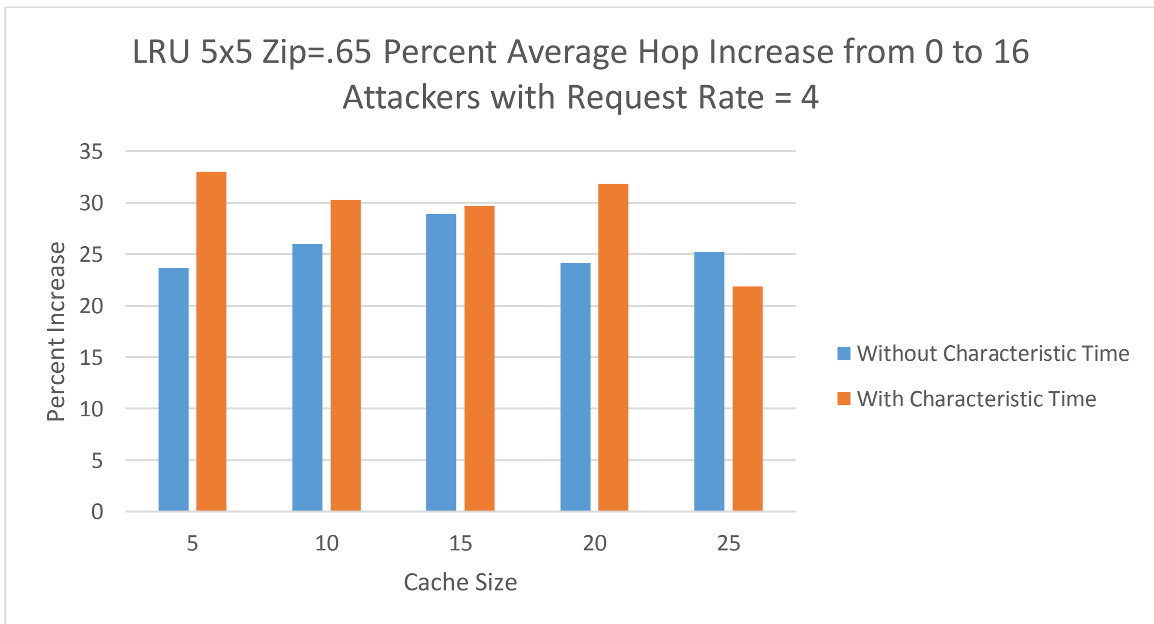
Gnutella 8846 node Graph (Zipfian=0.85): Percentage Increase Statistics



Smart Attack Results. 5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 1



Smart Attack Results. 5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 2



Smart Attack Results. 5x5 Square Graph. LRU. Zipfian = 0.65. Request rate of 4

VITA

Jeffrey B. Gouge is currently working as an IT Security Analyst in the IT Department of the University of North Florida in Jacksonville, FL. He has over 7 years of experience in the IT industry including .NET development, server and application administration, IPS/IDS administration and monitoring, CAS and Shibboleth administration, and other Windows and Linux security administration. He holds a Bachelor of Science in Information Technology with a concentration in General Security Administration. He is a self-motivated worker who works well in team and individual settings. He enjoys learning and using new technologies and tools related to all areas of computing.

and currently lives in Jacksonville, FL.