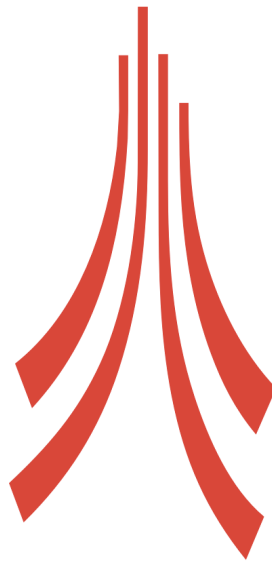


# **Using DHT-based Peer-to-Peer Networks to Orchestrate a Distributed Denial of Service Attack**

## **Project Report**

**Miles Davenport**



**Lancaster University, July 2007.  
Supervisor: Paul Smith**

# Declaration

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

In the case of electronically submitted work, I also consent to this work being stored electronically and copied for assessment purposes, including the department's use of plagiarism detection systems in order to check the integrity of assessed work.

Date: **October 1<sup>st</sup>, 2007**

Signed:

# Acknowledgements

Completing a Masters degree as a part-time student is not for the faint hearted. I would like to personally thank the people who have helped me turn the Masters in Critical Software Engineering into a reality. My dissertation supervisor Paul Smith has been a constant source of ideas and motivation throughout my dissertation. Andrew Brampton has made himself available to answer my questions relating to the peer-to-peer simulator. Chris Edwards and Trish Harrison have taken the time to answer queries and offer advice over the last two years.

Website address for Java simulator, Perl trace log parser instructions, scenario test data, and operating instructions:

**<http://www.milesdavenport.com>**

# Table of Contents

<b>CHAPTER ONE: INTRODUCTION</b> .....	<b>5</b>
1.1 OVERVIEW .....	5
1.2 CONTRIBUTION.....	7
1.3 DISSERTATION OUTLINE .....	8
<b>CHAPTER TWO: BACKGROUND RESEARCH</b> .....	<b>9</b>
2.1 EXISTING PEER-TO-PEER NETWORK SYSTEMS .....	9
2.2 DISTRIBUTED HASH TABLES (DHTs).....	11
2.2.1 <i>Content Addressable Networks (CANs)</i> .....	11
2.2.2 <i>Chord</i> .....	12
2.2.3 <i>Pastry</i> .....	13
2.3 DISTRIBUTED DENIAL OF SERVICE (DDoS) ATTACKS .....	14
2.3.1 <i>Why Launch a DDoS attack?</i> .....	16
2.3.2 <i>Malware and Botnet Creation</i> .....	18
2.3.3 <i>Target Selection and Attack Initiation</i> .....	19
2.3.4 <i>Attacking the Selected Target</i> .....	21
2.4 <i>ATTACK DETECTION AND PEER-TO-PEER SECURITY</i> .....	26
<b>CHAPTER THREE: DESIGN</b> .....	<b>31</b>
3.1 SIMULATOR ARCHITECTURE.....	32
3.2 PARSING PEER-TO-PEER SIMULATOR TRACE LOG.....	33
3.3 BOTNET CREATION .....	36
3.4 DDoS TARGET SELECTION SCENARIOS .....	37
3.5 TESTING STRATEGY .....	42
3.6 INSTIGATING A DDoS ATTACK.....	43
3.7 CONTROLLING A BOTNET .....	43
<b>CHAPTER FOUR: IMPLEMENTATION OF THE DDoS TARGET SELECTION SCENARIOS</b> .....	<b>45</b>
4.1 ASSESSING NORMAL PASTRY PEER-TO-PEER NETWORK BEHAVIOUR .....	45
4.2 IMPLEMENTATION TOOLS AND ENVIRONMENT .....	46
4.2.1 <i>Java Simulator Implementation</i> .....	47
4.2.2 <i>Simulator Trace Log Parser Implementation</i> .....	48
4.2.3 <i>Implementing Scenarios Two and Three</i> .....	50
4.3 REMOVING IRRELEVANT DATA FROM THE SIMULATOR TRACE LOG.....	51
<b>CHAPTER FIVE: EVALUATION</b> .....	<b>52</b>
5.1 TARGET SELECTION SCENARIOS .....	52
5.1.1 <i>Scenario One</i> .....	52
5.1.2 <i>Scenario Two</i> .....	54
5.1.3 <i>Scenario Three</i> .....	55
5.2 TARGET SELECTION SCENARIOS - LARGER NUMBERS OF REQUEST MESSAGES .....	57
5.2.1 <i>Scenario One</i> .....	57
5.2.2 <i>Scenario Three</i> .....	58
5.3 BOTNET CREATION .....	59
<b>CONCLUSION</b> .....	<b>61</b>
<b>BIBLIOGRAPHY</b> .....	<b>62</b>
<b>GLOSSARY</b> .....	<b>65</b>

# Chapter One: Introduction

## 1.1 Overview

Peer-to-peer networks provide the means for a widely dispersed audience to share information of common interest such as files or music by allowing a peer to accurately request items of interest which are not locally available. A request can be forwarded across the network until it arrives at a peer who has local access to the object. This functionality differs from a client server environment because peers implement client and server behaviour (such as routing and request forwarding) instead of routing behaviour being the sole responsibility of the server. As traditional working environments have been replaced with working away from the office, mobile environments have flourished. The disadvantage of a widely dispersed user base is that the once readily available fast network and access to resources are suddenly **not** available. More flexible approaches to this dispersed sharing of files have allowed peer-to-peer networks to be the foundation for mobile applications, which are an alternative to the traditional client-server model.

A peer-to-peer network has the same traits as any other interconnected computer system. For the system to function there has to be available network and computational resources to fulfil the designated role of the application which uses the peer-to-peer network. A Distributed Hash Table (DHT) is a peer-to-peer network that provides a deterministic lookup service which associates ownership of a file to a network peer. Peers on the network route requests for files by determining how “close” they are to the requested asset.

A distributed denial of service attack (DDoS) is a term given to an attack which is launched from multiple locations whose goal is to compromise the availability of the target system. DDoS attacks have the traits of flooding a target with a high volume of traffic, impacting the services and network which they rely on. This type of attack will be easy to detect because of the sudden change to the operating environment but not easy to contain. Because the attack is not limited to a single location it cannot be easily “filtered”, the attack will likely alternate its source and characteristics of system availability do not allow you to “just” pull the network connection, the attack must be managed rather than eradicated. The characteristics of a peer-to-peer network allow them to be the basis to launch a Distributed Denial of Service attack where the target can be an internal node or an external target. Examining the peer-to-peer network environment and carefully selecting a suitable target gives the attack the greatest chance of succeeding.

Why has more effort not been spent in preventing Distributed Denial of Service Attacks? The anti-virus industry has been accused cashing in on virus infections [5] with anti-virus “patches” being developed shortly after the virus is released “into the wild”. This reactive approach to the defence of computer systems is in contrast to the pro-active view offered by autonomic computing [6].

A DDoS attack which has the traits of flooding a network with data and exhausting available resources will change the normal behaviour profile of a peer-to-peer network. By observing these changes the attack can be managed with the objective of successfully repelling it. An alternative approach would be to attack the peer-to-peer network in a stealthy way, such that it does not change the observable normal network behaviour. Before launching an attack a suitable target should be selected. By selecting a node which is popular and subsequently launching a DDoS attack, the attack has a greater chance of success and remaining undetected if the normal characteristics of the network are not changed. By closely examining the normal behaviour of a peer-to-peer network, the criteria for stealthy target selection will be identified. This includes acceptable margins for normal network behaviour. Measuring acceptable behaviour will allow a number of target selection scenarios to be designed and implemented. These will be the basis for a DDoS attack using selected nodes in a DHT peer-to-peer network as the botnet.

For the target selection scenarios to be implemented, a DHT Pastry simulator will be extended to implement the scenarios. The scenarios will form part of a malware component which will extend the functionality offered by the simulator. By varying the number of peer-to-peer nodes with malware, the objective will be to assess if the malware implementing the target selection scenario will remain stealthy. As essential part of evaluating the success of the stealthy Target selection scenarios will be to answer the question, "Is this target selection scenario stealthy?" By plotting and comparing the normal behaviour of the simulator and the scenario the question will be answered visually.

This dissertation will look at novel target selection approaches that can be used to determine the most suitable target on a peer-to-peer network, such that by attacking it maximum disruption is caused to the network. A core aim is to achieve this in a **stealthy** manner. Stealthy target detection is not an established part of a Distributed Denial of Service (DDoS) attacks. Current ways of selecting a target are based on financial, political, brand success, and publicity characteristics. Existing attacks are rarely stealthy, and do not pay attention to observing a targets operating environment and normal behaviour. They more commonly focus on flooding a target with traffic (which will overload the network or services which the target relies on. Some attacks (cyberprotests) are advertised in a deliberate ploy to get the attention of the media (or a relevant authority), and cannot be classed as stealthy.

The disadvantage of none-stealthy attacks is the investment that the attacker must make in acquiring and controlling a botnet (a network of compromised computers). This investment will be lost as soon as the botnet control mechanism is discovered and disabled. If as much resource was spent in DDoS target selection when compared with protecting the botnet, attacks would be far stealthier. Moving away from high-profile attacks which have a short duration, to stealthy attacks which assess the target environment and are classified as normal traffic will have a stronger chance of success.

## 1.2 Contribution

The **primary** aim of the project is to research stealthy target selection within a Distributed Hash Table (DHT) peer-to-peer network. A Pastry [7] (implementation of a DHT network) simulation will be used to implement the target selection scenarios.

By analysing the normal behaviour of a peer-to-peer network, characteristics such as the total number of requests or average round trip time (RTT) for a certain key can be used as measurements for target selection. By creating a number of scenarios which creates a network of malicious peers whose size can be changed, the threshold for consistently selecting the correct target node can be gauged. The findings will illustrate what number (percentage) of malicious nodes and volume of traffic is required to successfully target a suitable node.

To meet the primary aims, the focus of this dissertation is to research, design, implement, and evaluate the steps for selecting a target node within a botnet, prior to an attack being launched. The steps are:

1. **Botnet creation** – ensure that zombie creation does not disrupt the normal operation of the peer-to-peer network. As much as possible, the administration of the zombie, and botnet should not be distinguishable from normal traffic.
2. **Target Selection** – selecting the correct DDoS target will involve asking specific questions of potential target nodes. Questions may include “how busy are you?” or “how many messages have you forwarded (processed) in the last N minutes?” Answering these questions will allow a busy node to be attacked by a realistically sized botnet. Attacking a well-provisioned node with a botnet that is too small (that generates too little traffic) will not interrupt the service.
3. **Launching the attack** – The attack will have to be coordinated, ensuring each zombie in the attacking botnet is correctly synchronised.

The **secondary** aim of the project is to research and implement stealthy control centre functionality. The role of the control centre is to manage the botnet, selecting an attack target and instructing and synchronising zombies when to carry out an attack.

Control centre functionality is becoming more complex with network monitoring, attack statistics for the botnet, and mechanisms for the defence of the control centre being implemented. These characteristics are implemented in the SpamThru [3] controller, which has the ability to move the control centre, and regain control of the botnet after a control centre has been disabled.

Researching the possibilities of statically locating the control centre outside the botnet, dynamic relocation of the botnet from one node to another are important in ensuring that the botnet remains stealthy.

### **1.3 Dissertation Outline**

The background research chapter of the dissertation details how peer-to-peer networks have developed with an explanation on the reasons for launching a DDoS attack on an often high-profile target. Particulars about peer-to-peer network implementations and distributed hash tables (DHTs) will be included. The individual steps of creating a malicious network, selecting a target, the variety of attacks which can be used against the victim and how attacks can be detected and the risk mitigated will be discussed.

The design chapter details the components of a peer-to-peer simulator that have been utilised to implement scenarios for stealthy peer-to-peer target selection. Design particulars are presented on how the simulator trace log is parsed so that accurate information on normal peer-to-peer network behaviour can be obtained and analysed.

Following the design chapter, the implementation chapter will present target selection scenarios which will measure popularity and round trip time (RTT) for a specific number (initially one thousand) key requests messages which will provide traffic for the peer-to-peer simulator.

The evaluation chapter will analyse peer-to-peer simulator parser logs for each scenario and will plot graphs relating to popularity and round trip time. Size of botnet and volume of request messages will be varied to assess the number of malware peers to reliably select the most vulnerable target node. The findings of the evaluation will show that the greater the volume of request messages, the smaller the size of botnet threshold needed to consistently select the same target. Botnets which are larger than this threshold will select the same target but stealthy characteristics will be reduced because the size of botnet will increase the chance of discovery.

## Chapter Two: Background Research

The chapter presents an introduction to peer-to-peer networks illustrating some well-known networks which have been involved in file sharing applications, focussing on the different design of each network.

A peer-to-peer network differs from a client server network with peers taking an equal view on the provision of a service. A network of interconnected peers distribute tasks including responding to requests for files or data, routing requests to neighbouring peers who can service a request, and maintaining the integrity of the network by assisting new peers in joining the network, and modifying routing information for peers who have left the network. These characteristics differ from a client-server model where a central server services client requests. Work is not distributed across the network but is kept within the confines of the server which will have superior computational power to clients.

### 2.1 Existing Peer-to-Peer Network Systems

A peer-to-peer network is made up of a varying number of “peers” (nodes). The number of peers and size of the network depends on the application or service that the network supports. A common application of peer-to-peer networks is file sharing [12].

Napster [36][37] was one of the first peer-to-peer networks to be used widely. The peer-to-peer technology used a central server which indexed every file which was available for sharing. To locate a file, Napster clients would search a central server. The search application would associate search terms with locations of known files which were shared.

Napster relied on the resiliency of the central server for the file sharing network to function. Having sole control of the server allowed the company behind Napster to provide an efficient service by staying in control of the network. A legal challenge ended with the server being shut down, leading to the instant demise of Napster. The Napster brand was subsequently purchased and is a popular legal music sharing business.

Gnutella [37] is another popular file sharing network. A decentralised network which does not rely on a centralised server prevents the network being shut down (like Napster). The Gnutella peer-to-peer network functions by connecting users directly to others, with no server intervention. This connectivity resulted in one huge network. Whenever a Gnutella node wants to search for a file, a broadcast request is sent to every node connected to it. Neighbouring peers check to see if they had the requested file, and pass the request on to all peers they are connected to. The way in which requests are broadcast to adjacent peers is slow and inefficient. However the distributed way in which the network operates, means there is no single point of failure.

The FastTrack [37] peer-to-peer network introduced SuperNodes.

These are nodes which are promoted, allowing them to take on “centralised server responsibilities” including search request resolution and requests for neighbouring peers. This allows search requests to be efficiently resolved, but removes peer equality from the network.

The equivalent to SuperNodes in BitTorrent [38] (another peer-to-peer files sharing protocol) are Trackers. Peers connect to Trackers allowing connected peers to find each other. A Torrent is a small file which contains the initial detail which is required for downloading a file. Multiple Torrents are associated with a Tracker. The most popular way of searching for a Torrent is to use a Torrent index website. These sites list Torrents, providing an effective way of searching for files. Some BitTorrent peers support Distributed Hash Table (DHT) functionality (see section 2.2), which allows them to work without a Tracker. A peer is then able to send a search request, which will be resolved using the deterministic routing offered by the DHT protocol.

Regardless of the specific peer-to-peer network implementation, the purpose of the peer-to-peer network protocol will be to specify how each peer in the network is to service requests for whatever object (file, or asset) is being requested. This request will involve a decision on whether to forward the request to the node that is (potentially) closer to the correct location, or to make a decision that the asset is stored locally. Additionally, network routing and maintenance must also be accommodated by the network. If a node joins the network, a mechanism must exist for allowing the node to efficiently join the network, update its own routing table and neighbouring nodes. If a node leaves the network, neighbouring peers should update their routing tables to allow for this situation.

## 2.2 Distributed Hash Tables (DHTs)

Distributed Hash Tables (DHTs) are an approach to building peer-to-peer networks. They provide a deterministic lookup service that takes a key value representing a resource, and returns an address associated with the given key. The key is typically a value that is based upon a hash of the resource's identifier, for example its file name. As with other approaches to peer-to-peer networking, there is no central server in a DHT network. Every client either routes a request to a node that is closer to the resource, or returns its address if the resource is located locally. A number of protocols have been designed to implement DHTs, some prominent examples will be discussed in the following sections.

### 2.2.1 Content Addressable Networks (CANs)

A Content Addressable Network (CAN) [13] provides distributed hash table functionality that can be deployed on the scale of the Internet. CANs utilise a Cartesian coordinate space, which is partitioned amongst the peers in the network. Each partition is a zone which contains a number of hash (key, value) pairs and details of neighbouring zones. Search requests for a specific key are routed via intermediate nodes towards the node which has the desired key in their zone. A CAN node holds a routing table with the IP addresses and coordinates of its neighbours. Figure 2.1 illustrates how two nodes are considered neighbours if their coordinate spans overlap along  $d - 1$  dimensions and abut along one dimension.

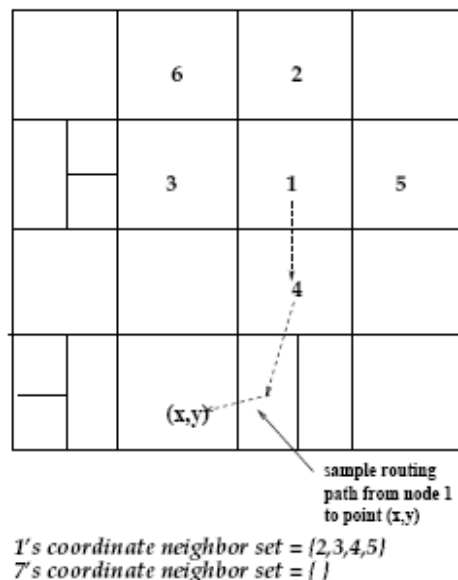


Figure 2.1: CAN Routing table of Immediate Neighbours [13]

Figure 2.1 illustrates that node five is a neighbour of node one because its coordinate zone overlaps with 1's along the Y axis and abuts along the X axis. Node 6 is not a neighbour of 1 because their coordinate zones abut along both the X and Y axis. A CAN node routes a message by forwarding it to the neighbour whose coordinates are closest in the coordinate space to the target node.

## 2.2.2 Chord

Chord [14] provides a peer-to-peer framework for mapping hashed keys to a node that holds the object associated with the key. A consistent hash function uses a node's IP address to obtain its identifier and a key identifier by hashing the key. A standard base hash function (SHA-1) is used to hash (key, value) pairs. A circle of nodes is constructed by using consistent hashing. A successor node is identified as the first node whose identifier is equal to or follows the key value in the identifier space.

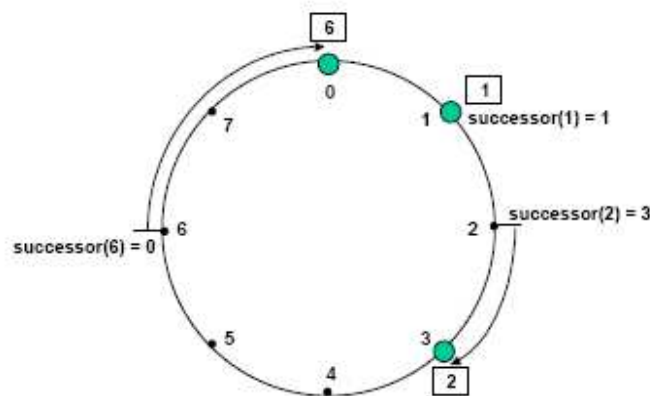


Figure 2.2: Identifier Circle consisting of three nodes 1, 2, and 3[14]

Figure 2.2 illustrates an identifier circle which shows that successor nodes are located by finding the first node clockwise from 0 to  $2^m - 1$ . The circle shown has three nodes 0, 1, and 3. The successor of identifier 1 is node 1. Key 1 would be located at node 1, key 2 located at node 3, and key 6 at node 0.

Consistent hashing allows nodes to leave and join the network with minimal changes to routing information. When a specific node leaves the network, all keys that were assigned to the node are moved to the node's successor. If a new node were to join the network with an identity of seven, it would assume responsibility for the identifier six key from identifier zero. Queries are passed round the circle until they encounter a node which the query maps to. This type of query resolution may be inefficient if too many successor nodes are visited.

To increase efficiency, additional routing detail is stored in an optional finger table. The finger table entry includes the IP address (port), and Chord identifier of the relevant node.

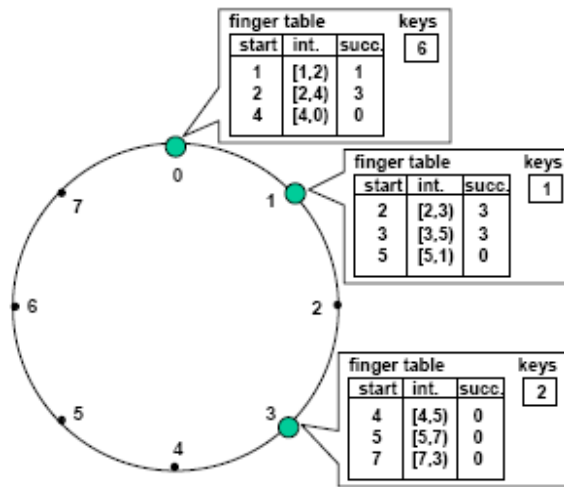


Figure 2.3: Identifier Circle with finger table key locations [14]

Figure 2.3 illustrates the finger table of node one which contains details of successor nodes with identifiers  $(1 + 2^0) \bmod 2^3 = 3$ , and  $(1 + 2^2) \bmod 2^3 = 5$ . Each node stores the details of a small number of adjacent nodes, and does not have sufficient information for nodes which are further away. If a node  $n$  has no detail on a successor for the key  $k$  the node  $n$  will attempt to find a node  $x$  which has an identity which is closer to key  $k$ . By repeatedly asking node  $x$  which node in its finger table is closest to preceding key  $k$  node  $n$  learns about nodes which are closer to  $k$ .

### 2.2.3 Pastry

Pastry is designed to efficiently locate objects in very large peer-to-peer networks that use the internet for connectivity. Pastry can support a wide range of applications including file sharing and publish-subscribe systems. A node in a Pastry network has a unique node identifier (nodeId). When a Pastry node receives a message with a target key, it will route the message to a Pastry node with a nodeId that is closest to the target key. This will involve a node comparing its node identity with the identity of the message. The message will be forwarded to a node which when compared with the requested key has a nodeId which is at least one digit longer than the present nodeId. To support this routing mechanism each node maintains a routing table.

NodeId 10233102			
Leaf set		SMALLER	LARGER
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	
Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

$\lfloor L/2 \rfloor$  numerically closest smaller nodeIds  
 $\lfloor L/2 \rfloor$  numerically closest larger nodeIds  
 ceiling( $\log_2 bN$ ) rows with  $2^b - 1$  entries  
 each entry in row  $n$  shares a prefix of length  $n$  with the present node  
 nodes chosen according to proximity metric  
 $\lfloor M \rfloor$  closest nodes (according to proximity metric)

Figure 2.4: Pastry Routing Table

A routing table for pastry node with nodeId of 10233102 is illustrated by Figure 2.4. The top row of the routing table is row zero. Shaded cells in each row of the routing table highlight the corresponding digit of the present nodeId (10233102). Nodes are selected by applying a proximity metric with nodes in row  $n$  of the routing table which share a prefix of length  $n$  with the node 10233102.

Peer-to-peer networks which utilise Distributed Hash Tables (DHT) improve on the design of Napster and Gnutella by employing a distributed environment which is more resilient than a network which utilises a central server. Request resolution which broadcasts a query to neighbouring peers is an inefficient (excessive traffic and peer load) way of routing and resolving a query. The deterministic way of routing a request via nodes with a nodeId numerically closest to a key is more efficient. Pastry has been tested on simulated networks of one hundred thousand nodes [7]. Results were positive proving that DHT based networks scale efficiently.

### 2.3 Distributed Denial of Service (DDoS) Attacks

The Internet consists of an interconnected network of computers. Attitudes to security and system administration vary widely. This situation leads to interconnected systems with limited resources, bandwidth and processing power. These characteristics are specifically targeted by Denial of Service (DoS) attacks.

A Distributed Denial of Service (DDoS) [28] attack is a concerted effort to make a computer resource or service unavailable. It has the same purpose as a Denial of Service (DOS) attack, but relies on a network of compromised computers (a botnet) to attack the service.

This will use up system resources, disrupting Web servers, email, or any service that uses the network (or Internet) to function.

A botnet is made up of so-called zombie machines – these are computers that have been compromised and are remotely managed. An attack usually involves the whole botnet being instructed to attack a specific target. Although individual zombies run autonomously – attacks that utilise the whole network are likely to be co-coordinated from a central point (often referred to as a command / control centre). This will usually control the botnet via a common protocol such as via Web server (using HTTP) or an Internet Relay Chat (IRC) server.

Botnets are distributed throughout the Internet and can be present in dial-up, home broadband, academic and business environments. They work most effectively on high-specification machines with a fast network connection and are valuable resources that take time to setup. Individual zombies who are of a high specification, with fast connectivity, which are also vulnerable are not readily available. Computer security awareness, up-to-date anti-virus software and firewalls are measures that can reduce the likelihood of a server becoming compromised.

Responding to a DDoS attack is not as simple as changing one part of a system. Limiting the impact of an attack may involve installing packet filtering, restricting protocols or changing system parameters in response to environmental changes [15]. Managing a DDoS attack so system availability is retained can involve numerous small real-time changes to an operational environment. Where a production system utilises a peer-to-peer network retaining resiliency in the face of unexpected events is critical. On August 16, 2007 the Skype peer-to-peer network became unstable, leading to service disruption. A large volume of patched Skype servers were rebooted at this time resulting in a flood of login requests which overwhelmed the peer-to-peer network. This service disruption had the traits of a DDoS, but was instead the victim of a number of simultaneous events which overstretched the Skype infrastructure [40].

The importance placed on system availability means that a strategy for responding to DDoS attacks should be in place. Some high profile websites have looked to third party [39] DDoS defence services because of the variety and volume of attacks.

Regardless of the number of attack points, a popular attack is to send a large volume of packets to a target. This will cause excessive bandwidth to be used in transit and in the target's locality. These types of attacks are defined as packet flooding attacks.

TCP packets that have the SYN flag set indicate a request to open a new connection – this is standard TCP behaviour for any networked application to establish a new connection. In an attack, the request will likely have a random spoofed source IP address. The target will respond and then waits for confirmation which will never arrive. The timeout duration for the connection is minutes. This situation will be repeated numerous times from multiple attack locations. The targets resource of available network

connection will be quickly exhausted, with new connections (legitimate or otherwise) being ignored.

The TCP/IP protocol does not readily provide mechanisms to ensure integrity and authenticity. A false source IP address can be used to launch an attack specifically using intermediate sites. Sending a large number of false emails “supposedly” from a victim’s email address, “to a non-existent” email address will result in large volumes of “Cannot Deliver email” bounced email messages being sent to the victim. The legitimate intermediaries used in this attack make detection of the source difficult.

### **2.3.1 Why Launch a DDoS attack?**

The rationale for launching a DDoS attack is varied. They include making a statement and gaining notoriety, brand damage, extortion, and phishing.

#### **Making a statement and gain notoriety**

Virus writers are motivated to climb the hierarchy from script kiddies, to warez distributor, to cracker. This is not only driven by technical expertise, but focuses on making a statement and gaining notoriety. One popular way to go about this is to attack computer systems in a blatant, non-stealthy way. This would get you recognised for technical acumen.

Political and economic motivation has led to opponent sites being the target of DDoS attacks [16][17]. Danish and related blogs, websites and email infrastructure were the victims of DDoS attacks after details of the inflammatory Mohammed cartoons were published. These attacks have originated from a large number of countries, have used a variety of attack strategies and have been noticeable by the long duration (more than twenty four hours continuous) of attacks.

Making any sort of statement gets you noticed. This is not limited to the stereotypical teenage hacker, but it also the model for security consultants, ethical hackers and vulnerability analysis and discovery businesses. A “statement” includes finding a new vulnerability which may be published (and made available) to the highest bidder. This is the thinking behind the WABISabiLabi [9] vulnerability marketplace. More conventional sources of revenue are made by so-called ethical hackers who test products, components and entire systems for vulnerabilities [8].

The losers in the third-party vulnerability business are often the vendors themselves. A third-party vulnerability business will not divulge new exploits to vendors, but will instead publicise them to clients for profit [8]. This situation does nothing to standardise a pro-active defence response to virus’, exploits and attacks.

#### **Brand Damage and Extortion**

Extortion and damaging high-profile brands is the motivation for threatening to launch attacks on sites and services offered by well-known brands. After the NoChex online payment website was targeted by extortionists [18], third-party packet filtering technology was employed to remove the majority of the malicious traffic. The attack also affected the infrastructure of the Internet Service Provider which hosted the NoChex site. The only way for targeted online services to survive brand damage is to work with authorities and factor in resources for repelling attacks.

## **Phishing**

Although phishing is not a direct means of instigating a DDoS, theft of personal information is a side-effect of malware infection. After a computer system becomes infected by malware, personal information may be surreptitiously taken and used for the purposes of extortion or phishing. Targeting particular individuals or business customers with a phishing email encouraging them to confirm passwords or reveal further valuable information is a common tactic.

The following email was sent to the author. It contains characteristics of a phishing email, especially the IP address being used “to take the trusted customer” to the Ocado website. It was later found that the email was legitimate, but sending email to registered customers does nothing for brand confidence and has all the traits of a phishing attempt:

Dear Mr Davenport,

Wednesday 25th October 2006.

Our technical team has made us aware that some customers are currently experiencing problems accessing the Ocado website.

This has been caused by a technical issue in a third party domain name server facility in New York in the USA. The domain name servers translate URLs like [www.ocado.com](http://www.ocado.com) into real internet (IP) addresses, which are just numbers. One of the third party domain name servers which translate many URLs, including Ocado's, failed this morning and some customers are having difficulty reaching our site as a result. We are one of many businesses around the world that has been affected.

If you are having trouble reaching [www.ocado.com](http://www.ocado.com) you can use this real IP address today instead <http://213.86.44.71>. You will then be able to access the site as normal.

Yours faithfully,  
Mark Bentley  
Head of Customer Services

### 2.3.2 Malware and Botnet Creation

A large number of computers accessible from the Internet will become the victim of malicious software, known as malware, which is surreptitiously installed without the user knowledge. Once the malware becomes functional, the computer will become a malicious node also known as a zombie. The zombie will be connected to a network of zombies called a botnet.

The user habits that lead to a computer becoming a zombie are exploitative. Responding to phishing attempts, installing unknown software, visiting suspicious websites are all activities which will likely lead to the installation of malware. These activities combined with a computer system which is lacking in anti-virus, firewall, and operating system updates becomes a likely target. The best zombies are computers that are well resourced and have a fast connection to the Internet.

The characteristics of DDoS malware include the ability for the compromised system to communicate with a control centre which will co-ordinate DDoS attacks and provide a way for sensitive details to be passed back to the control centre.

Other malware features (using the SpamThru Trojan [3] as an example) include anti-virus as part of the malware (software which allows a computer to become a zombie (single malware infected computer) which removes competing malware, the ability for a botnet controller (also called a *herder*) to regain control of the botnet even though the control centre for the botnet has been disabled. Once a new control centre has been established, gaining control of a single peer allows the herder to regain control of the entire botnet.

Statistics which identify a zombie has participated in DDoS attacks is available to the bot herder. A database of zombie target information allows the botnet to select a target based on specific information relating to the targets operating environment. For example, if the target were involved in stocks and shares a DDoS attack which caused millions of emails to fictitiously come from the target would be stealthy in that it would not result in a denial of service by traffic load, but would damage the reputation of the target, especially if legitimate users were encouraged to enter their details as part of a phishing attack [4]

Creating a botnet, attacking a target and getting caught does not automatically mean that legal proceedings will be successful. Legal legislation for response to DDoS attacks is not well defined. In 2005 Wimbledon youth court dismissed a case [2] against a charge of sending millions of emails to an email server which become overloaded. The defence argued that email servers existed to receive emails and that there was no maximum limit for receiving emails. The overwhelmed email server was the victim of a Denial of Service Attack.

### 2.3.3 Target Selection and Attack Initiation

Target selection for Distributed Denial of Service attacks is often on the basis of whoever controls the botnet and which sites or services have a high-profile or are designated successful. A DDoS Botnet may be for hire, with a specific target [19].

Where a target has not been manually “targeted”, port and vulnerability scanners are used for target selection. Huge numbers of sites (IP addresses) are selected to see if any vulnerabilities can be located which will make it a DDoS target.

Vulnerability scanning is not only limited to selecting DDoS targets. It is an offensive and defensive strategy used to check how vulnerable a computer system is to a break-in. This approach too target selection is not new. War dialling [22] involves systematically trying to locate the numbers associated with modems through testing each extension of a telephone system in turn. This is equivalent to port scanning a directory of websites.

For the majority of attacks, target selection is based on more subjective or profit-focussed reasoning. Traits like brand success, jealousy and extortion have already selected the target. At times there is no specific target for a DDoS attack. A compromised server will be used to send huge amounts of email spam to completely untargeted individuals. Addresses will have been collected from blogs, forums and mailing lists. Email addresses are freely available; individuals will not be consulted and are not specifically targeted. The spam DDoS attack will send out the emails regardless.

The botnet communication channel for the botnet will be via recognised protocols including Internet Relay Chat (IRC) [21], and Hypertext Transfer Protocol (HTTP) [20].

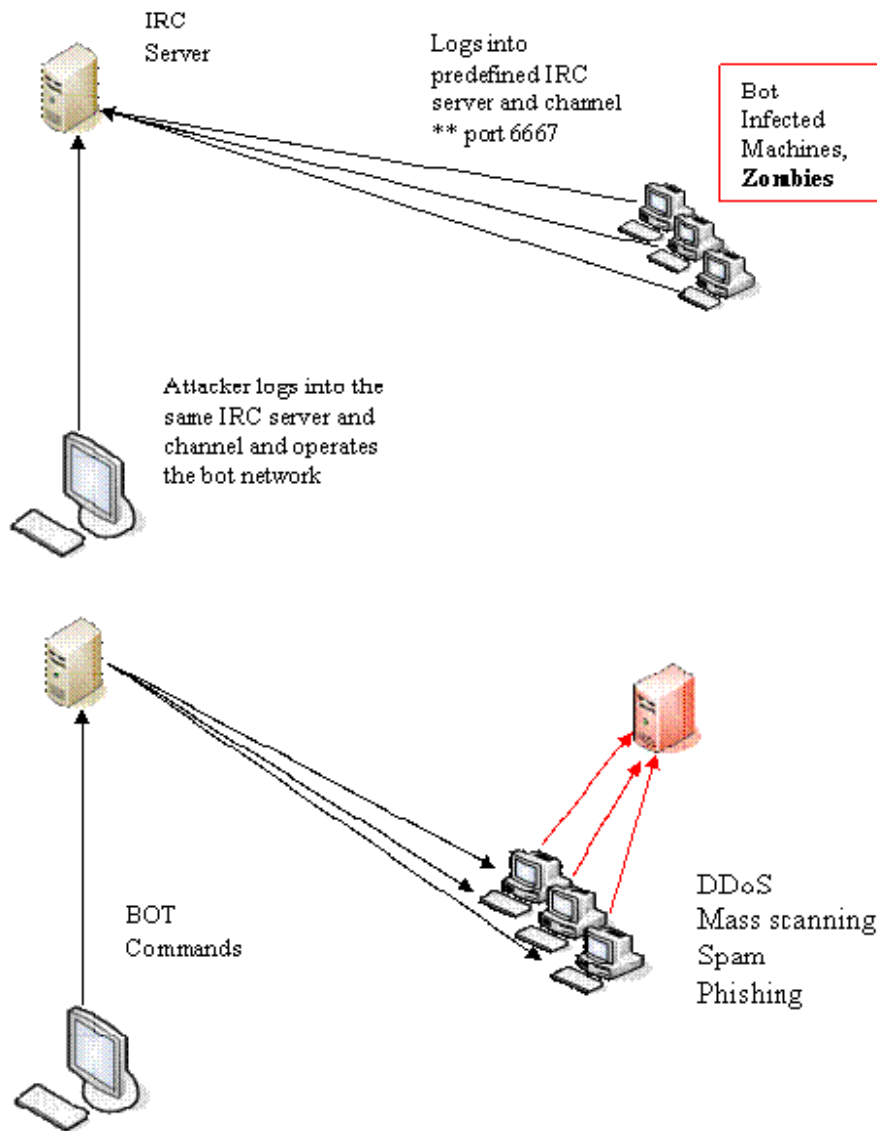


Figure 2.5: Internet Relay Chat (IRC) controlling a Botnet [23]

Internet relay chat utilises channels where users meet, nicknames are a username which a user gives themselves and bots are automatic programs which are used for administering a channel. Bots look like a normal user, and carry out tasks including moderation, the prevention of channel abuse and automated assistance. Bots can also be manipulated to log into a specific channel and listen for commands issued by a controller. This abuse of IRC is illustrated in Figure 2.5.

Each zombie has malware installed on the compromised machine. This includes IRC bot functionality. These listen for commands (administrative and malicious) with specific syntax, which are executed by all zombies “listening” to that channel.

### 2.3.4 Attacking the Selected Target

The aim of a DDoS attack is to disrupt the availability and quality of service provided by distributed computer systems including Internet, private networks, mobile, and combinations of all three. The vast majority of these systems will utilise a connection-based protocol, for example Transport Control Protocol (TCP), or connectionless protocol such as User Datagram Protocol. TCP-based applications utilise a handshake process to establish a connection between client and host, and require a reliable, ordered, stream based connection (see Figure 2.6). Applications that use UDP are not looking to make a connection, they send messages as individual packets (not streams) and do not require packet receipt – reliability and correct packet ordering are not guaranteed with UDP (see Figure 2.7).

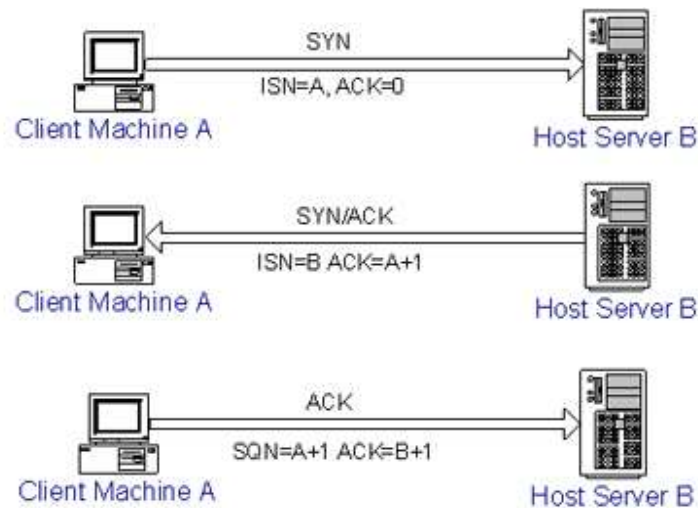


Figure 2.6: TCP Three Way Handshake Connection [24]

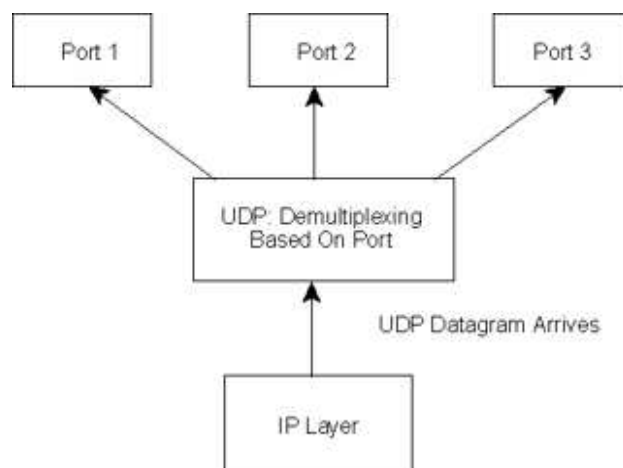


Figure 2.7: UDP Connectionless Delivery [24]

## **DDoS TCP and UDP Attacks**

The number of TCP connections a computer can maintain is finite. Like any other resource, once they are all occupied no further connections are possible. Flooding a network with TCP connection requests will use up all the network connections. Using false source IP addresses will also cause the acknowledgements from the server to look for a client which does not exist.

UDP port based communications can also be flooded. Because the UDP protocol does not guarantee ordering of packets, it may be the job of the application receiving the packets to retain ordering. By replaying packets which have already been sent, this may confuse the application sequencing of packets, and may reduce quality of service.

## **Amplification Attacks**

Domain Name Server (DNS) is the equivalent of a telephone directory for the Internet. The main purpose of a Domain Name Server is to translate hostnames into IP addresses which networks require for the delivery of information. Whenever an Internet enabled computer needs to translate a hostname against an IP address it will query a Domain Name Server. DNS traffic must be allowed to pass freely round the Internet for this essential process to succeed.

Domain Name Servers are either recursive or non-recursive. Requests for domain names can be authoritative, or non-authoritative.

A non-authoritative query involves the local “recursive” DNS querying a top level domain root name server which further delegates the query for the domain query. The information is then returned to the local DNS. This process can only be performed if recursive DNS querying is enabled by the local server.

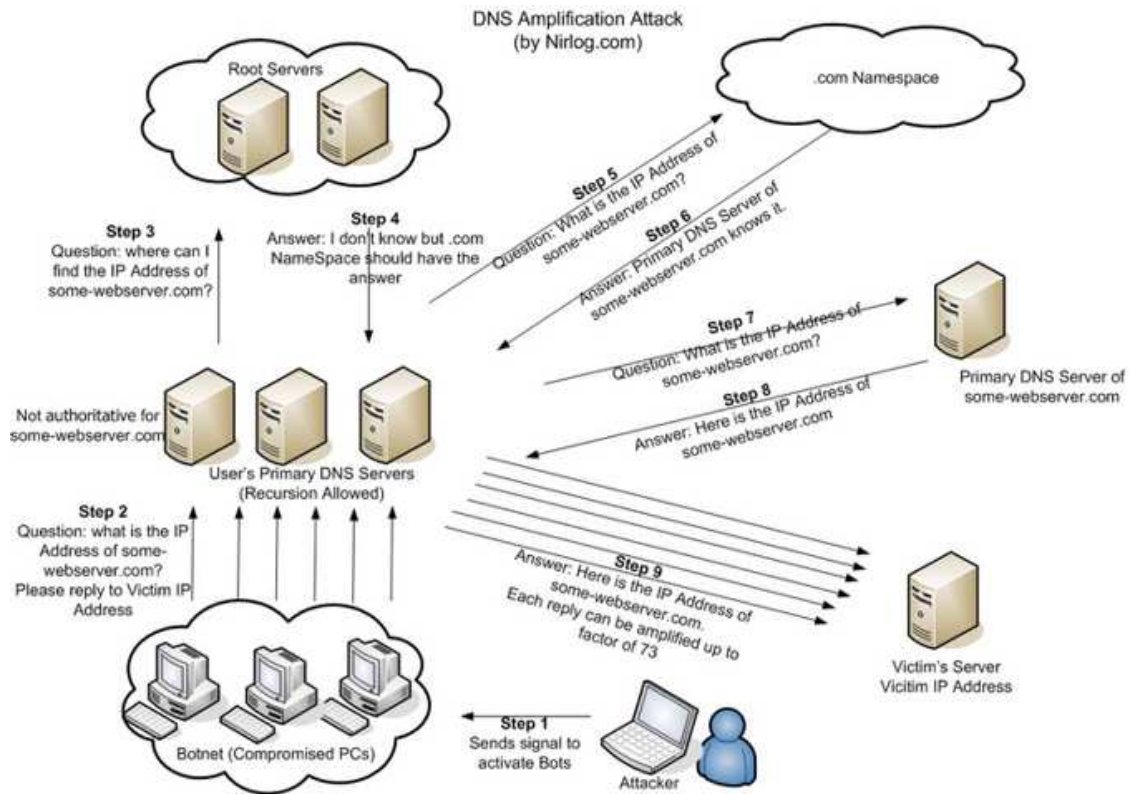


Figure 2.8: DDoS Domain Name Server Amplification Attack [25]

If a large number of DNS requests are simultaneously launched against a recursive Domain Name Server with a forged sourced IP address, the requests will be amplified (see Figure 2.8). The Domain Name Server will generate a large number of replies which are sent to the victim resulting in increased network traffic and resource usage, degrading the services offered by the target [26]. The recursive DNS requests made to root name servers will increase the network and computational load on these critical services, affecting service availability for others using DNS and bandwidth.

### Peer-to-Peer Network Attacks

The same DDoS attack characteristics affect a peer-to-peer network. A peer-to-peer network that has a large percentage of nodes have been compromised by malware has the characteristics of a DDoS engine [11].

The Gnutella peer-to-peer network locates assets by flooding adjacent nodes with queries. These ripple out until the object is found, answering with a reply, which uses the return path to the node which originated the query. Trust is employed as there is no formal way of establishing the authenticity of the response. The QueryHit packet consists of a header that contains the IP address and port number of the node that holds the asset. Trust is

once again employed to ensure that the IP address and port are genuine. Once a response is received, the peer can connect directly to the node and access the asset. The structure of the request and associated response is virtually identical to an HTTP request response:

**Request:**

```
GET /get/<File Index>/<File Name> HTTP/1.1\r\n
User-Agent: Gnutella\r\n
Host: 123.123.123.123:6346\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
\r\n
```

**Response:**

```
HTTP/1.1 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 4356789\r\n
\r\n
```

The mechanism for locating Gnutella assets can be modified allowing internal and external sites to be the victim of a DDoS attack. Because there is no way of ensuring the integrity of responses, the payload can be manipulated to point at a target.

Any web server will try to process a request that contains a carriage return and new-line sequence of characters (`\r\n\r\n`). If a Gnutella file “`../../live HTTP/1.0\r\n\r\nfoo bar.mp3`” is passed to a targeted web server as:

```
GET /get/1/../../live HTTP/1.0\r\n\r\nfoo bar.mp3 HTTP/1.1\r\n
User-Agent: Gnutella\r\n
Host: 123.123.123.123:6346\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
\r\n
```

The web server is obliged to process the “`\r\n\r\n`” changing the request to:

```
GET /live HTTP/1.0\r\n\r\n
```

This will force the Gnutella peer to download a file called “live”. Depending on the size of the file, the external web server will be overwhelmed with requests for the file [27].

By modifying a QueryHit packet header a zombie can change the hops and source information fields. This will prevent neighbouring nodes from identifying the location of the malicious node. It will be forwarded back to its assumed source, but the anonymity of the zombie will be protected.

Another attack involves poisoning the peer-to-peer index system by inserting false records [11]. These bogus records suggest that a number of popular assets are located at a specific IP address and port number. This location does not have to be within the peer-to-peer network; for example, external websites; email, FTP, or any other service could be targeted. In this scenario, when a search request is made for the asset the index will point them at the target. The peer will then attempt to connect and download the asset. The likely result is TCP connections that are held open, wasted bandwidth and resource.

A variation of the index poisoning attack involves poisoning a peers routing table with the addition of non-existent neighbouring peers [11]. The identity of these peers will be that of a victim. Sending a number of join messages will announce the existence of the peers to legitimate neighbouring peers. The non-legitimate peers will then be added to the routing table of the legitimate nodes, poisoning their routing table. Subsequently, by selecting the bogus peer to route request traffic to, the target peer will receive a flood of messages. If the index poisoning is wide spread the messages sent to the victim will be numerous, leading to a DDoS attack. The target can be within the peer-to-peer network or an external service, as before.

### **Passive and Active Attacks**

A common DDoS attack strategies which involve flooding a network or resources with large volumes of traffic over a short period of time. These attacks have more active characteristics as opposed to more passive attacks which are characterised by lower volumes of traffic sent to the target over a longer period of time.

Passive attacks focus on degrading a service steadily over a long period of time, rather than overwhelming a service. This approach has the advantage of being difficult to identify, blending into normal behaviour characteristics. Behaviour patterns can be observed, information can be gathered, business intelligence can be amassed, and if a DDoS attack is launched the information which is gathered will be invaluable in target selection and attack timing for maximum impact.

### **Criminals Plan Stealthy Crimes with the Intention of not Getting Caught**

Criminals who want to commit a Stealthy crime will have extensively researched their target observing the operating environment, paying special attention to what constitutes normality, how normal behaviour changes over a period of time and by measuring these changes. These practices will help to ensuring that the crime is not instantly detected.

Non-stealthy attacks which brute force to impulsively commit a crime are far more likely to get caught.

The Art of Shoplifting [1] focuses on careful selection of your target, investigating the layout of the chosen shop, checking the positioning of security cameras and “blending in”

by not looking suspicious. Stealthy DDoS attacks should adopt similar practices of observing the operating environment.

## ***2.4 Attack Detection and Peer-to-Peer Security***

In a predictable fixed-boundary, non-distributed environment attack detection is made easier due to physical security boundaries which do not change. A firewall which surrounds a centralised server which has a fixed perimeter with well defined incoming and outgoing rules with definite normal traffic levels will be easier to defend against a malicious external attack than a decentralised peer-to-peer network. The main advantage of Napster [36][37] having a centralised server was that it could be managed effectively because the server was on a network which was managed solely by Napster, with an internet service provider who could filter suspicious traffic in an environment which was not distributed.

In a distributed environment a co-ordinated response to an attack from different parts of the peer-to-peer network is essential. If an attack can be managed before quality of service is disrupted (or stops) reactive attack detection will have succeeded. Moving from a reactive to a proactive stance on attack detection involves awareness to changes in the environment.

Analysis should identify and categorise attacks and corresponding defence strategies (see Figure 2.10) using information from the detection phase. Using the detail gained from detection and analysis, strategies for responding to an attack can be developed (see Figure 2.9). It is important to gather information from realistic situations. In a perfect situation, there would be meaningful sensors deployed in every part of the system. The disadvantage of this approach could be information overload.

By analysing information from selected parts of the peer-to-peer network, specific attack scenario detail can be gathered. Specific attack scenarios can be gathered by analysing information from firewall logs, dropped packets, port scanning, outgoing connections, and traffic levels which do not follow usual patterns.

Bees are attracted to pots of honey. A honeypot is a resource placed on a monitored network which has a high likelihood of being compromised by an attacker. They observe suspicious traffic, capture malicious target selection and port scanning mechanisms. Honeypots have two levels of complexity:

1. Low interactive, emulation of a service. An example of this is an email server which when connected to implements the simple mail transfer protocol (SMTP), but is not a fully fledged email server.
2. High interactive functional operating system and environment that can be compromised, with fully implemented protocols and services to support the honey pot

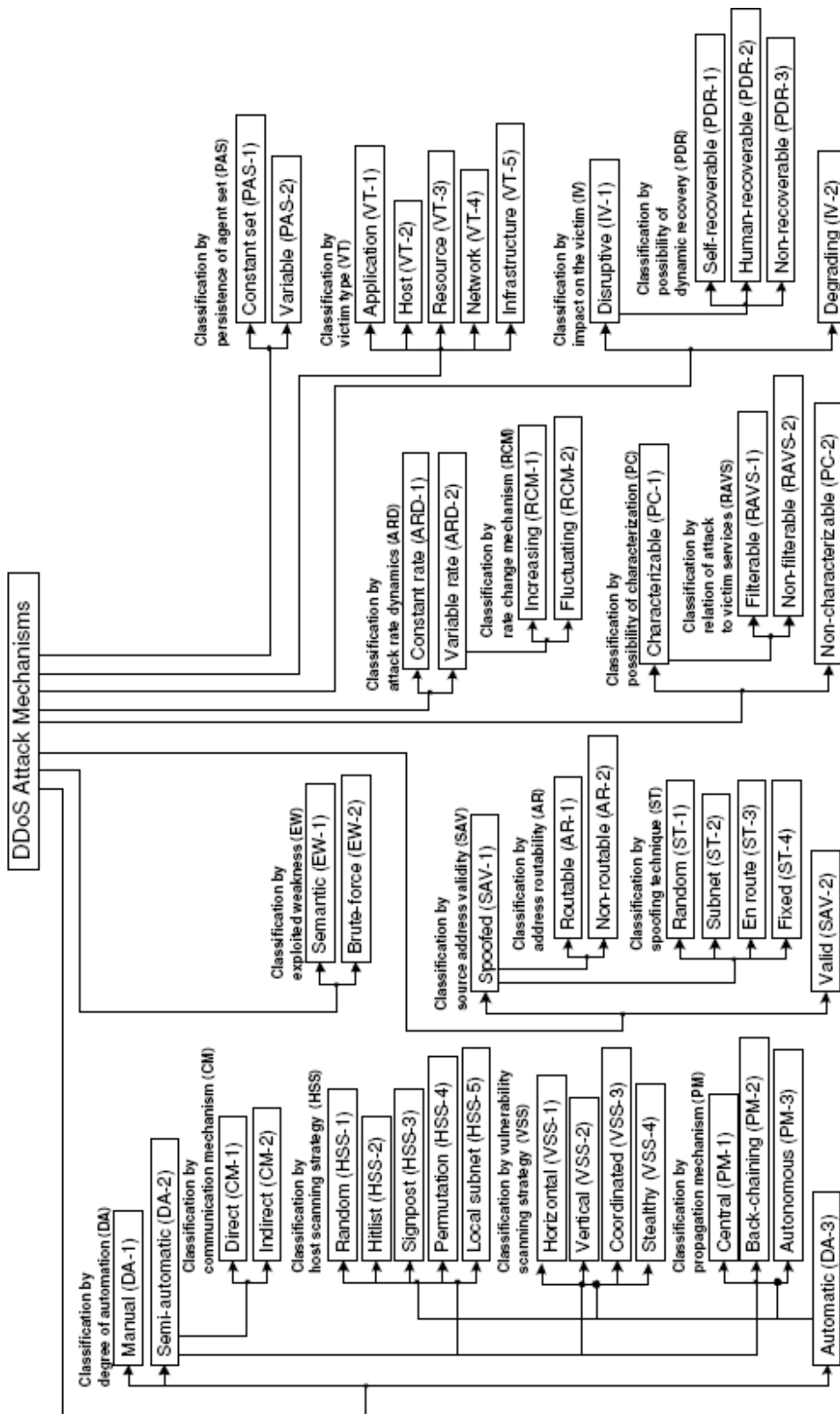


Figure 2.9: Taxonomy of DDoS Attack Mechanisms [28]

Clifford Stoll [29] was an astronomer turned systems engineer who was alerted to an intruder in his employers' computer system. He spent a year observing the intruder by constructing a Honeypot to observe the behaviour of Markus Hess [29]. Valuable information was gathered and disseminated to the authorities and security communities of other compromised sites. This assisted in the eventual capture of Markus Hess.

A honeynet has the same job as a honeypot, but network functionality is implemented. Care should be taken when implementing a honey net. Honeynets can be detected by analysing the timing of responses to network packets. A competent attacker could work out what networks have genuine servers, and which are honeynets. Detection of honeypots or honeynets will lead to them being avoided.

Observing attacks on a computer system which employ a honeynet involves analysis of behaviour, attack patterns, documented exploits, and system logs. This analysis should start to categorise the gathered details into DDoS attack and defence strategies.

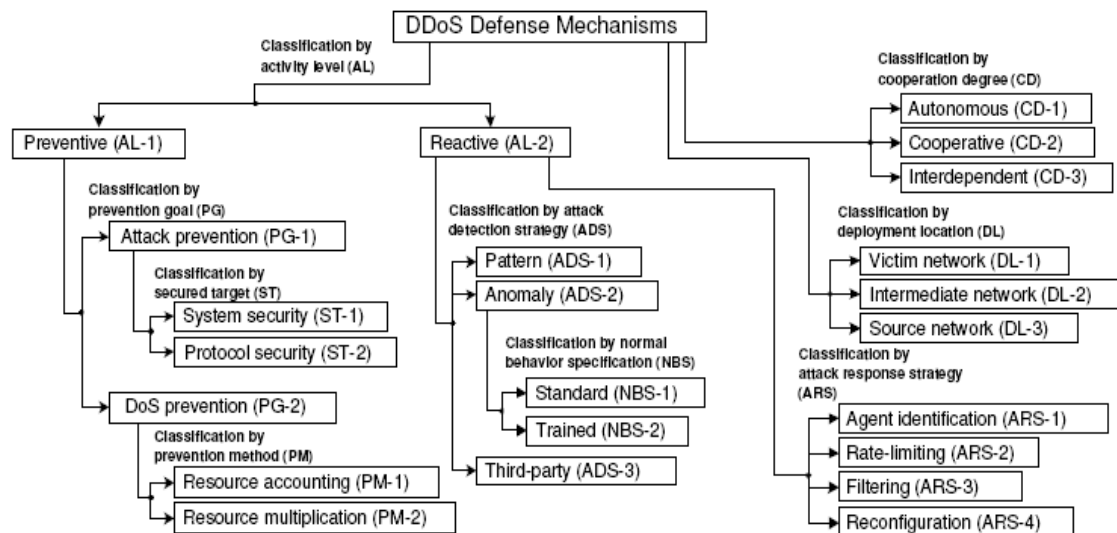


Figure 2.10: Taxonomy of DDoS Defence Mechanisms [28]

Analysing DDoS attacks is a complex task. The operating environment of a peer-to-peer network can change rapidly; with parts of the network having different environments to others, for example a mobile telephone data network will have different characteristics to a wired network. The ability for these local areas to record and analyse local behaviour is critical. The subtle differences between the analysis of one part of a peer-to-peer network compared to another will be critical in finding the stealthy subtle traits of a passive long-term attack. This approach would increase the chance of identifying localised peer groups within a larger botnet, with their own port and control centre structure [3].

## Flash Mob, DoS Attack and Flash Crowd Events

Flash Mobs are social gatherings of individuals who have been instructed to meet at a specific location at an exact time, and perform a task. The task does not have to make any sense to the environment you are in. For example prior instructions were given for a record number of people to meet in St. Anne's Square Manchester on Tuesday 5th June, 2007 to perform the Asahi exercise (a mix of Karate and Yoga).

A flash crowd event is the distributed systems equivalent of a flashmob. This will happen to a service when a large volume of traffic arrives at a Web site. Predictable flash crowd events occur when a Web site is publicised (such as a sporting event) then receives a deluge of requests from interested parties. Both DoS attacks and flash crowd events have the ability to overload a server's bandwidth and resources, resulting in a loss of normal service. Distinguishing between flash crowds and DDoS attack is a challenging area of ongoing research.

If a DoS attack were launched during a flash event, the ideal behaviour would be for the service to ignore the attack, and handle the legitimate requests. The strategy for this to succeed is for the service to implement attainable DoS defence mechanisms. By monitoring connections to the service and their associated request rate a server **may** identify some traffic that may be classed as legitimate. When quality of service degrades to an unacceptable level, traffic that has excessive connections, and requests from the same location should be dropped.

The observations applied to DoS attacks can be applied to DDoS attacks. The diversity of DDoS attacks leads to a situation where it is impossible to anticipate and defeat all attacks. Filtering of the type discussed will marginally improve the effectiveness of defending against attacks, but may not increase the likelihood of flash crowd events not being mistaken for DDoS attacks (see Figure 2.11).

Characteristic	Flash Events	DoS Attacks
Traffic volume	Both have a noticeable increase in terms of the number of requests. The length of peaks can be large or small depending on the episode.	
Number of clients and their distribution	Caused mostly by an increase in the number of clients accessing the site. Client distribution can be expected to follow population distribution among ISPs and networks.	Caused either by an increase in the number of clients or a particular client sending requests in a high rate. Client distribution across ISPs and networks does not follow population distribution.
Cluster Overlap	Significant overlap between clusters a site sees before and during the FE.	Cluster overlap is very small.
Per-client request rates	Because a server usually gets slower during the FE, per-client request rates are lower during the FE than usual. This indicates that legitimate clients are responsive to the performance of a server unlike DoS attackers who generate requests by a pre-determined time distribution.	Some DoS attacks involve a few clients emitting very high request rates and some a large number of clients generating a low request rate, but in both cases per-client request rate is stable during the attack and significantly deviate from normal.
Requested files	File popularity follows Zipf-like distribution.	The distribution of files used by attackers is unlikely to be Zipf-like. In many cases, a large number of compromised hosts stress a Web server by requesting the same small set of files. It is also possible that a particular host repeatedly requests a large set of files (which may not even exist on the server). Both behaviors result in distributions that are not Zipf-like.

Figure 2.11: Comparison DoS Attacks and Flash Event characteristics [31]

## Chapter Three: Design

To identify stealthy attack scenarios for a DDoS attack, normal peer-to-peer environment characteristics must be understood and recorded so they can be used as a basis for observing DDoS target selection behaviour and how it deviates from the norm. Normal behaviour can be measured in a number of ways, including the average number of messages processed (forwarded or resolved locally) by a peer-to-peer node in a specific timeframe, the average round-trip time for a request to be resolved and by looking at the distribution of the key space for targets in request messages.

The ideal environment for observing the behaviour of a peer-to-peer network would be to assess the behaviour of a network which was supporting a live production application used in the real world. Due to time constraints this was not possible for this project. An existing peer-to-peer simulator written in Java is the basis for observing the behaviour of a peer-to-peer application and for selecting scenarios that allow DDoS target selection to remain stealthy.

The simulator implements the Pastry peer-to-peer protocol (see Section 2.2.3). It uses a workload profile which, when run, constructs a network with a specified number of peers populated with a hash of one million (key, value) pairs. Once the network is populated a specific number of key requests will be applied against the network. The workload is logged to a comprehensive trace file.

It is questionable if a peer-to-peer network simulator can realistically simulate network latency and processor delay of nodes [32]. For this reason times for network hops or round trip time (RTT) may not be realistic when compared to a deployed Pastry peer-to-peer network. Alternative measurements, including number of request received, will be used to measure network behaviour.

Rather than weighing the simulator down with plotting functionality an approach which involves parsing the trace file and plotting required behaviour using GNUPlot has been adopted.

To capture standard network behaviour will involve parsing the simulator workload trace log. This contains information on every request and response message resolved by the peer-to-peer network including source and destination nodes addresses. Plotting this behaviour will provide details of how target selection should behave to remain stealthy.

Malware will be installed on different percentages of the peer population in the simulator. When the malware is selecting the most popular attack target and if the plotted profile does not change from that of the normal behaviour the target selection scenario will be deemed to be stealthy.

### 3.1 Simulator Architecture

This chapter will provide an overview of the pastry peer-to-peer network functionality offered by the simulator (see Figure 3.1). The simulator generates a network with hosts, routers and individual shared links with a specific bandwidth. To find the shortest path between two hosts Dijkstra’s algorithm [41] is used producing a routing table. Once the topology has been created it is applied to a simulator workload. The parameters for the workload include unique identifier, seed number and number of peers to create. The KeysTest workload used for target selection joins a specific number of peers to the network which has been constructed by the topology generator, places one million hash (key, value) pairs across the network, and generates a number of random requests for the keys which now populate the network.

Peer-to-peer network and underlying topology is represented by a large number of instantiated objects. An integral part of the workload is the creation of events which relate to peer-to-peer-network operations, including Pastry key requests. Events are placed in an event queue. The simulator starts processing events in the event queue, and finishes when the queue is empty. Running the events exercises the peer-to-peer network. A trace log is generated when the simulation is executed detailing network topology, peer overlay, workload, and results of events.

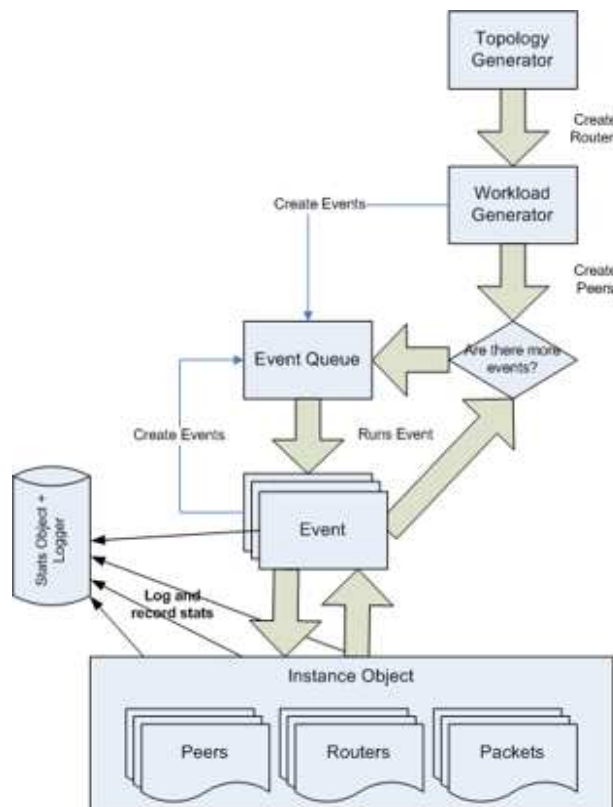


Figure 3.1: Pastry peer-to-peer Simulator Component Overview

For the KeysTest workload, one million hash (key, value) pairs are distributed across the network. A Zipf distribution is used to distribute the keys.

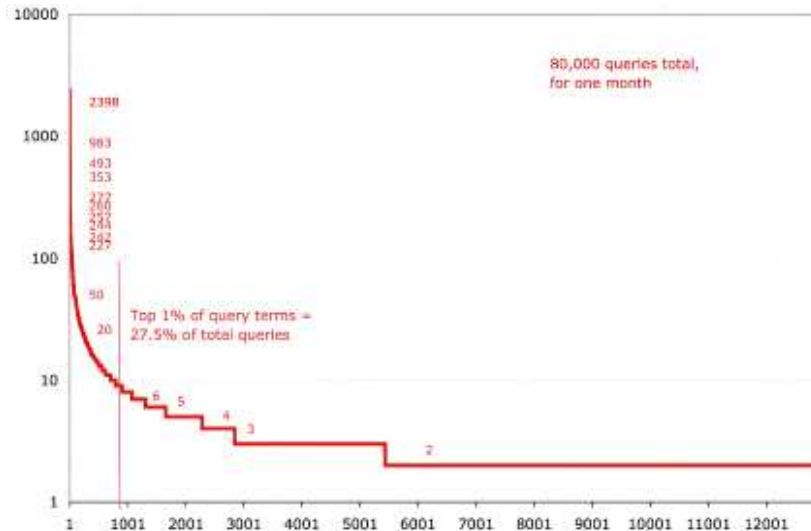


Figure 3.2: Zipf Distribution Unique Search Queries by Frequency [34]

Search keyword queries that follow a Zipf curve (see Figure 3.2), would illustrate that a few queries are very popular, a larger percentage are moderately popular, with the vast majority making up the long-tail. The term can also be used to describe how over a year a large number of small business' sell smaller numbers of items to more customers. This is the opposite of big business who have sales to move stock quickly which are the opposite of the long-tail [33].

The final task of the workload generator is to create a number of events which will exercise the network. For the KeysTest workload there will be one thousand Pastry lookup events. The events interact with the network topology and peers to resolve the key requests. The entire workload progress is logged to a trace file.

### 3.2 Parsing Peer-to-Peer Simulator Trace Log

The simulator trace log contains all the information required to assess the behaviour of the simulator including network topology construction and routing, adding peer-to-peer network overlay, and Pastry request key events.

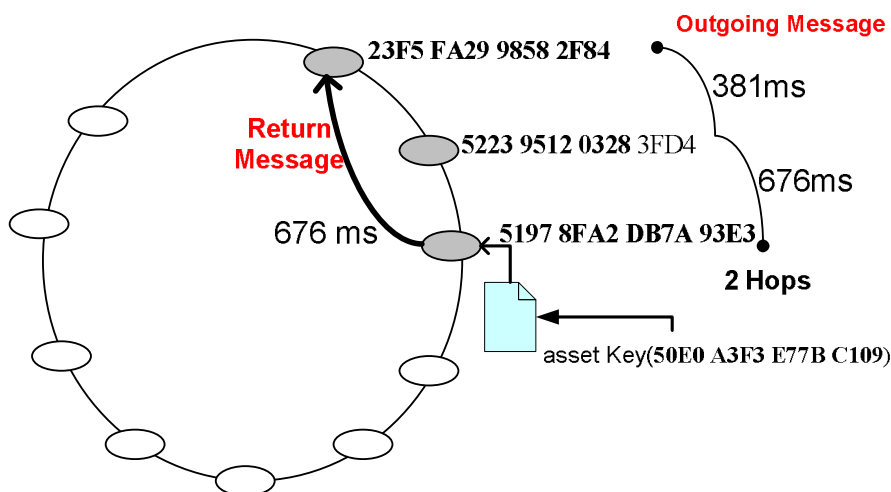


Figure 3.3: Request for Key (**50E0 A3F3 E77B C109**) illustrating hops, and RTT

Each request key event is split into individual network hops which illustrate the progress of the requests as it is deterministically routed. The direct return trip from the destination to the source is also logged. The following extract illustrates (see Figure 3.3) how a request for key 50E0 A3F3 E77B C109 is routed from node 23F5 FA29 9858 2F84, forwarded via node 5223 9512 0328 3FD4, resolved by 5197 8FA2 DB7A 93E3 who has access to 50E0 A3F3 E77B C109. The unique identity of the request is 2395.

The return message illustrates the direct hop back from destination 5197 8FA2 DB7A 93E3 back to the requesting node 23F5 FA29 9858 2F84. The unique identity of the response is 3596.

### Outgoing Message

```

5: 23F5 FA29 9858 2F84 Routing
5: 23F5 FA29 9858 2F84: sent GetMessage(2395) from 23F5 FA29 9858 2F84 for 50E0 A3F3 E77B C109 0 hops, 0 ms, 0 resents, 40 size 5827838185773449481
386: 5223 9512 0328 3FD4 In Range
386: 5223 9512 0328 3FD4 Leaf
386: 5223 9512 0328 3FD4: fwd GetMessage(2395) from 23F5 FA29 9858 2F84 for 50E0 A3F3 E77B C109 1 hops, 381 ms, 0 resents, 40 size
386: 5223 9512 0328 3FD4: sent GetMessage(2395) from 23F5 FA29 9858 2F84 for 50E0 A3F3 E77B C109 1 hops, 381 ms, 0 resents, 40 size 5827838185773449481
681: 5197 8FA2 DB7A 93E3 In Range
681: 5197 8FA2 DB7A 93E3 Me
681: 5197 8FA2 DB7A 93E3: rcv GetMessage(2395) from 23F5 FA29 9858 2F84 for 50E0 A3F3 E77B C109 2 hops, 676 ms, 0 resents, 40 size

```

### Return Message

```

681: 5197 8FA2 DB7A 93E3: sent GetReplyMessage(3596) from 5197 8FA2 DB7A 93E3 for 23F5 FA29 9858 2F84 0 hops, 0 ms, 0 resents, 44 size 2591252217178107780
1033: 23F5 FA29 9858 2F84 Me

```

1033: 23F5 FA29 9858 2F84: recv GetReplyMessage(3596) from 5197 8FA2 DB7A 93E3 for 23F5 FA29 9858 2F84 1 hops, 352 ms, 0 resents, 44 size

Each request can be parsed to identify specific information. For example, to identify the peer-to-peer node that received the most requests, the **recv GetMessage** for a particular node can be counted, with the corresponding **recv GetReplyMessage** to ensure the reply message was received, and completed.

For detection of peer-to-peer stealthy target selection behaviour, additional logging is added to the trace log.

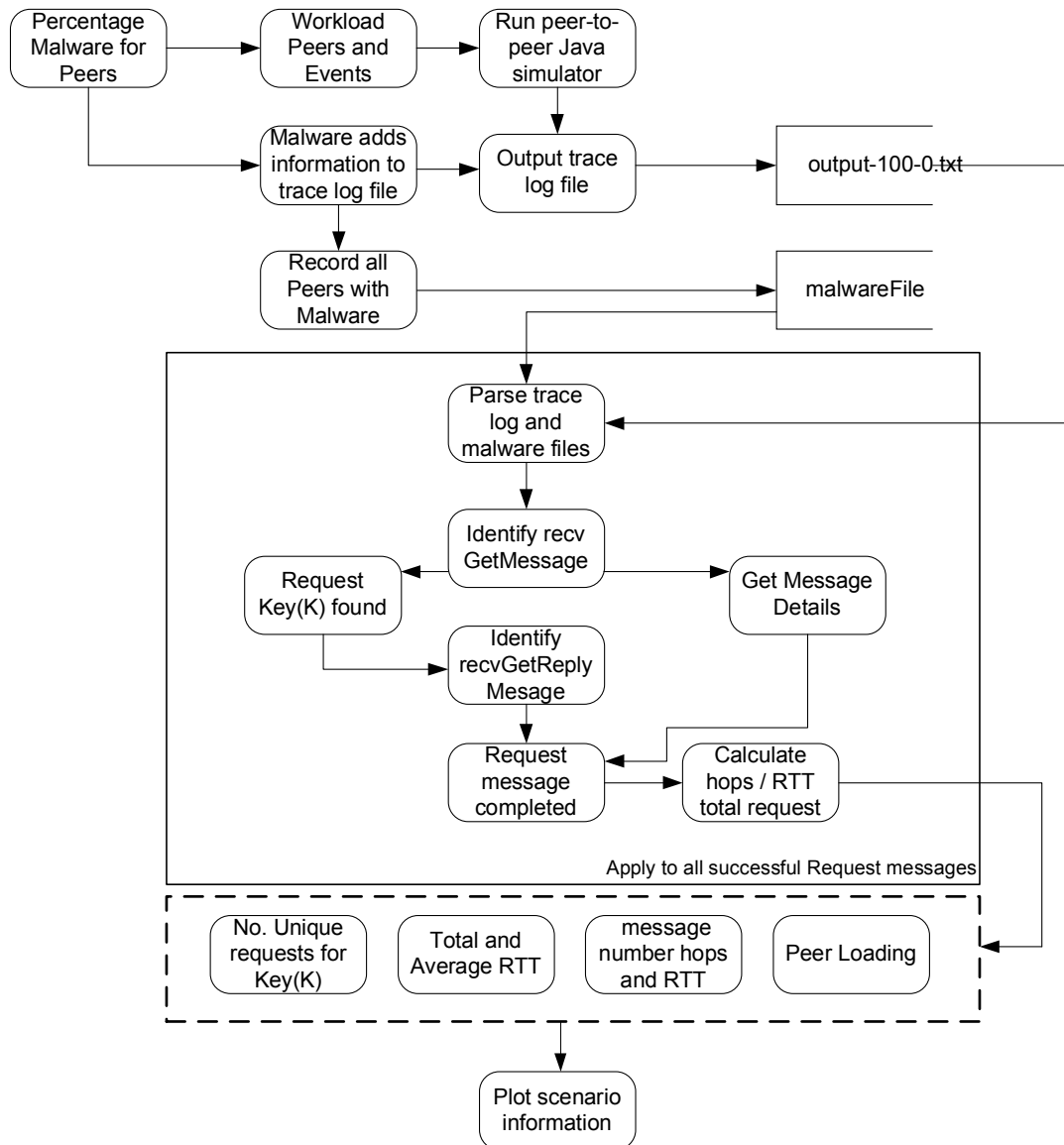


Figure 3.4: Most popular request Key (K) and round trip time (RTT) for messages

Figure 3.4 describes how data is parsed from the trace log file. Specifically how calculating unique requests, and round trip time (RTT) will be applied to all messages request messages which have successfully found the key(k) and have returned successfully to the source peer.

By randomly populating a specific number of peers with malware a botnet will be created. By gathering information on key request popularity and round-trip time botnet target selection will be accomplished.

### 3.3 Botnet Creation

As discussed earlier, botnet creation involves malicious software (malware) being introduced in computer systems that are connected to a network (Internet). The malware can vary in complexity, but allows the computer to be controlled by a third-party for the common attacks associated with a DDoS attack.

Functionality which creates a specific percentage of malicious peers (botnet) was added to the peer-to-peer simulator. The malware functionality was achieved by extending the Peer Java class to include malicious behaviour. This functionality can be applied to different percentages of nodes in the peer-to-peer network, which can then be compared with normal behaviour. The simulator class “sim.net.overlay.dht.ddos.Peer” implements peer behaviour including send and receive methods, which extends the “sim.net.overlay.dht.pastry.Peerbase” class. The Peerbase class contains specific information including routing and leafset detail. By adding a boolean hasMalware “flag”, and totalNumberMessagesReceived variables the behaviour of certain peers can be changed.

To enable this functionality an additional class “sim.net.overlay.dht.ddos.malware.PopulateMalware” has been created to add malware to certain peers. PopulateMalware randomly selects a specific percentage of peers.

If a peer is selected the hasMalware variable will be set to true, and the identity of the peer will be recorded in a file (named malwareFile).

The “totalNumberMessagesReceived” variable will be incremented keeping a total of all messages which are received, and **not** forwarded. This will be logged in the trace file. These are the only changes which been made to the peer-to-peer simulator.

Within the peer-to-peer simulator environment the passive nature of all three scenarios will not generate any additional Pastry protocol traffic. Analysis of data for target selection will use the trace log file.

In a real-life Pastry peer-to-peer network the target selection information would have to be analysed taking the distributed environment into consideration. This is where a stealthy scenario could suddenly become detectable.

For each peer in a network to send back information about what requests have been received, and what the average round trip time is will involve communicating with a control centre. This control centre may be outside the peer-to-peer network, or may be located at a specific internal peer. The way in which target selection information can be communicated to the control centre may involve using internet relay chat (IRC), HTTP, or a proprietary protocol.

With each node actively sending data to the control centre, which then issues details of chosen attack location, which attack to apply and attack duration the volume of data being sent may be enough for passive stealthy characteristics to be lost. This situation will be discussed in the evaluation section.

### **3.4 DDoS Target Selection Scenarios**

There will be three target selection scenarios: Scenario one will be based on largest number of successful requests for a specific Key (K). Scenario two will use the average round trip time for the results from scenario one. Scenario three will combine the results from scenarios one and two with the intention of illustrating that target selection where results based on popularity and RTT should consider the network and computational characteristics of a peer that is busy (under load).

If two peers have similar levels of popularity (scenario one), a better target for a DDoS attack will be a peer which is under load rather than selecting a peer with a quicker RTT. Figure 3.5 illustrates the computational and network characteristics of a peer under load

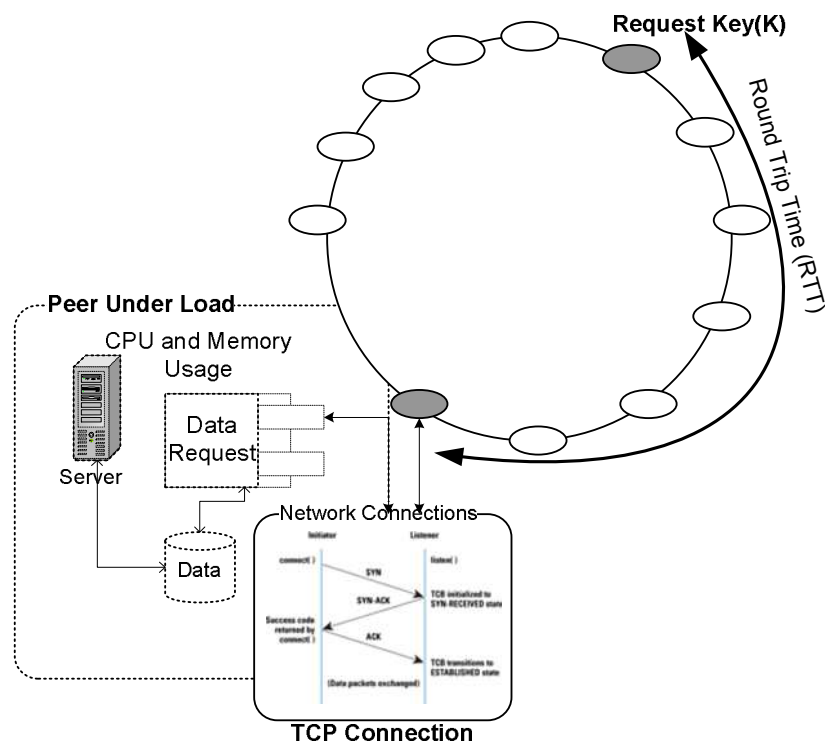


Figure 3.5: RTT, Network, and Peer load for resolving a Key request

For the scenarios to be implemented the notion of malware must be added to the simulator. A specific percentage of peers must be classed as having malware. This can be achieved by checking if a specific peer number matches a list of randomly generated numbers that represent a percentage of malware (illustrated in Figure 3.6). The specified malware percentage will be passed to peers as they are created. This will be achieved by creating a random list of numbers which will be passed to the simulator workload. Within the “KeysTest” class there is a loop which generates the appropriate number of peers before they join the peer-to-peer network “fastJoinAllNodes”. As the loop generates peers the peer number can be checked with the corresponding malware number (in the malware list). If the numbers match, the boolean value for “does the peer have malware” can be set to true.

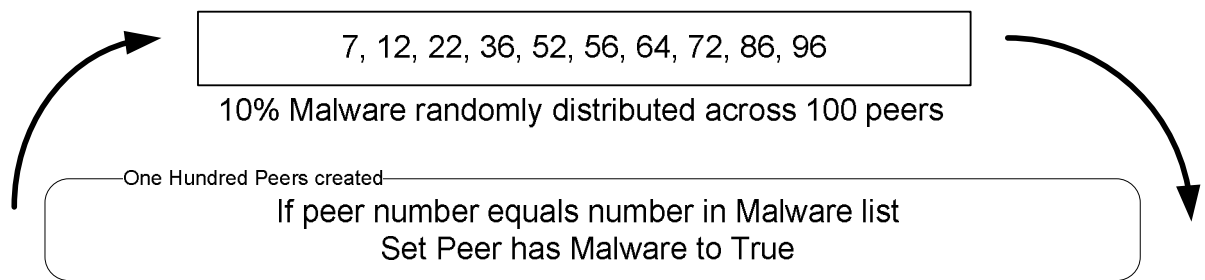


Figure 3.6: Botnet creation using Random percentage applied to new Peers

### Simulator trace log file parser

For the three target selection scenarios to be accurate they will need data from the trace log file which the simulator generates. A parser must be created which identifies when a message has found the target node for a key request and when the return message has completed successfully.

When a key request message finds the target node the text “*recv GetMessage*” is written to the trace log. Immediately this text has been found a return message is sent back to the source of the request. When this message is received successfully the text “*recv GetReplyMessage*” is written to the trace log.

After these two events have occurred the `nodeId` of the source request should match the `nodeId` of the return message. This confirms that the request message has been completed, allowing for the total request message counter for that key to be incremented. The number of hops and RTT for the request message is placed in a hashmap with a key of request key, identity of message. Once all request messages have been parsed the hashmap data is used to calculate the largest number of requests for scenario one, average RTT for scenario two, and a combination of two for scenario three.

The following functionality will be the basis for the parser:

Open trace log file

While there are still Events (getKey(K)) to Process

**Look** for the following text proving that the Request Key, Value has been found):

“xxxx xxxx xxxx xxxx”: recv GetMessage(xxxx<sup>4</sup>) from “xxxx  
xxxx xxxx xxxx”<sup>1</sup> for “xxxx xxxx xxxx xxxx”<sup>2</sup>

End If

Record the “xxxx xxxx xxxx xxxx” source node.

**Look** for the following text (Request message returned to source successfully):

recv GetReplyMessage(xxxx) from xxxx xxxx xxxx xxxx for xxxx  
xxxx xxxx xxxx<sup>3</sup>

If Return Peer identity<sup>3</sup> == Source peer identity<sup>1</sup> then

Message has been completed.

Increment total Counter for Request Key (K) identity<sup>2</sup>

Parse number total number of **hops** and total **RTT**

Store message information including total hops and RTT in -  
Programmatic Hash with a **key of { Request Key (K) identity<sup>2</sup>}{  
Outward message identity<sup>4</sup>}**

Store message **RTT** information in Programmatic Hash with a  
**Key of { Request Key (K) identity<sup>2</sup>}{ Outward message  
identity<sup>4</sup>}**

End If

End While

**Source peer identity** <sup>1</sup>  
**Request Key (K) identity** <sup>2</sup>  
**Return Peer identity (should be the same as <sup>1</sup>)** <sup>3</sup>  
**Outward message identity** <sup>4</sup>

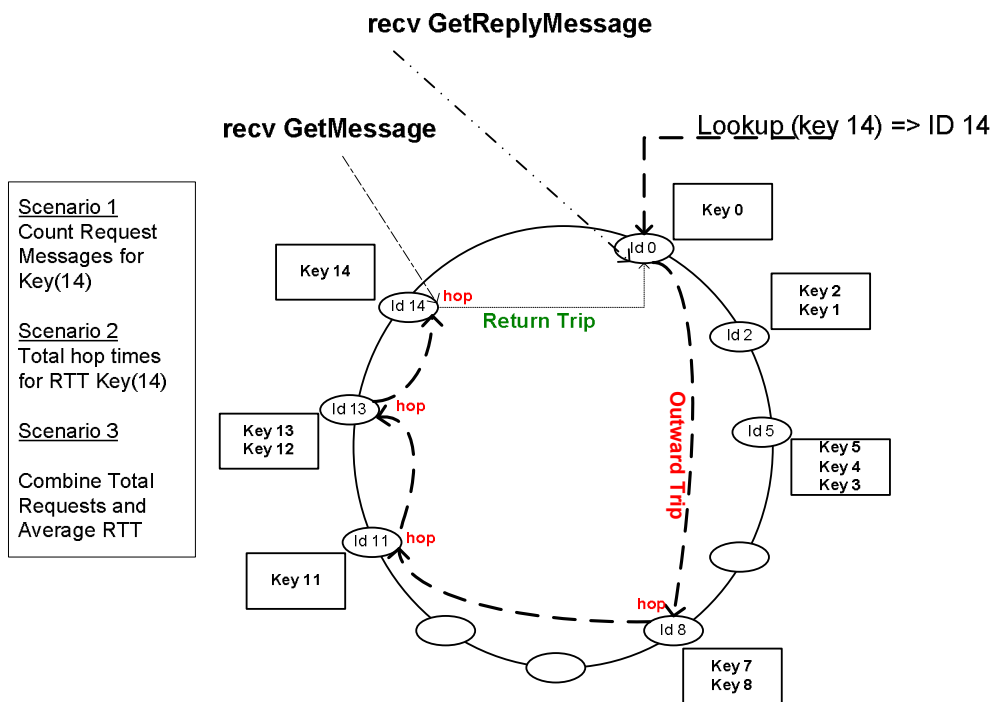


Figure 3.7: Illustration of the Data Parsed for Scenarios One, Two, and Three

### Scenario One: Target selection based on Key (K) Requests

Figure 3.7 illustrates the hop and RTT data which is parsed for the three scenarios by the parser. Scenario one involves all key requests being counted, grouped, and totalled. By counting the number of requests for a specific key using the log trace file, a target based on popularity can be identified. By identifying the most popular target with no malware installed this provided a basis for target selection. The following functionality must be added to the trace log file parser to find the most popular target:

Sort Hash with a **key of {Request Key (K) identity<sup>2</sup>}** into ascending order.

## Scenario Two: Target selection based on average Round Trip Time

Taking the information gathered from scenario one, calculate the average round trip time for each key request. The following functionality must be added to the trace log file parser to find a target based on average round trip time. The total RTT for each successful key request message has already been placed in a hashmap. The hashmap is referenced by two pieces of information request key identity, and the identity of the outward key request message. An inner loop will total all RTT values for a specific request key identity. This will be divided by the number of requests for the key which will provide an average RTT. This will be repeated for all unique keys.

```
Loop through Hash with a key of {Request Key (K) identity2}
  Inner Loop through with a key of {Request Key (K) identity2}{ Outward
  message identity4}
    Total all the RTT amounts for Request Key (K) by accessing:
    RTT information in Hash
    Total Request Key (K) messages
  End Loop
  Store Average RTT in Hash with a key of {Request Key (K) identity2}
  Store Total of Request for Key (K) messages in a Hash with key of {Request Key
  (K) identity2}
End Loop
```

## Scenario Three: Target selection based on Scenarios One and Two

By combining popularity and RTT values from scenarios one and two introduce a loading factor which takes the peer under load factors into account (see Figure 3.7). The following functionality must be added to the trace log file parser to find a target based on popularity and round trip time. The value for average RTT for a specific key request will be divided by the number of requests for the key:

```
Loop through each Hash with a key of {Request Key (K) identity2}
  Target selection value = (average RTT for Key (K) / Total number of Requests for
  Key (K))
End Loop
```

If a unique key request count exceeds 60 then an additional five hundred milliseconds will be added to the RTT of every request message (over sixty) which has been successfully completed. This is to allow for peer under load factors (see Figure 3.7).

### 3.5 Testing Strategy

The testing strategy for all three scenarios will be to run the simulator workload with no malware. By plotting the most popular key requests, the plotted data should follow a

Zipf distribution. This will highlight that the target selection follows the Zipf Key distribution used by the simulator workload to distribute key, value pairs across the Pastry peer-to-peer network. After the normal behaviour has been established, the simulator workload will be run with increasing percentages of malware present in the peer-to-peer network. The intention of this approach will be to discover the percentage of malware which selects the same target as normal behaviour, and whose distribution is “Zipf like”.

### 3.6 Instigating a DDoS Attack

Once a target has been selected, an attack could be launched on that target. Forged request messages could be generated in large enough volumes to overwhelm the target. The main disadvantage with this attack is that it is not stealthy. The volume of traffic combined with sudden loss of service will likely mean the target stops responding to requests of any kind from neighbouring peers. The result will be that leaf tables of neighbouring peers remove the target node. A more stealthy approach to attacking the target will be not to forward **any** message for the intended target. If the size of the botnet is large enough a malware will have to be used to forward a request message. If the malware node does not forward any messages then the node will be removed from leaf and routing tables. This attack is stealthy in that peer-to-peer service will be degraded over a period of time.

The malware can have other stealthy characteristics. If a botnet is not of significant size an attack can be suspended until the percentage of malware peers is large enough to accurately select a target node. This feature has the advantage of retaining the stealthy characteristics of the botnet until a target has been correctly selected.

### 3.7 Controlling a Botnet

A botnet will need a way of sending and receiving messages to relevant peers in the botnet. A number of ways exist for co-ordinating a botnet. Internet relay chat (IRC) or HTTP are common protocols used receiving information from malware peers, and co-ordinating the botnet as a whole. For a botnet to be completely stealthy no messages should ever be sent or received by zombies. Although this approach is stealthy, the DDoS approach could not be implemented. The simplest DDoS attacks need a degree of co-ordination. Removing this control would redefine the DDoS approach to a number of autonomous DoS attacks.

For a botnet to be stealthy the control centre must be able to relocate internally within the confines of the peer-to-peer network. For this to be a reality every zombie must have the potential to be a control centre. When the control centre moves, all other zombies in the botnet should be told about the change. Modified routing tables could be used to let all zombies know about the change. Part of the Pastry protocol involves a request message

being passed round neighbouring nodes so that a routing table can be constructed for the newly joined node. This behaviour could be modified by zombies forwarding a modified join message to other malware nodes. If the botnet was split into local groups, multiple join messages could be sent by the new control centre to different groups, minimising the timeframe that was needed to alert all zombies to the location of the control centre. The malicious join messages would have a greater chance of being treated as legitimate traffic as they are essentially part of the Pastry peer-to-peer protocol.

An alternate approach to managing the botnet would be for all zombies to implement a light weight web server which would have the ability to accept (and forward) HTTP requests. By once again splitting the botnet into local groups messages could be redirected from one zombie to the next until the newly located control centre was located. Once located, the control will be able to respond directly to the zombie. This is the HTTP redirect equivalent of the Pastry deterministic lookup for key requests.

# Chapter Four: Implementation of the DDoS Target Selection Scenarios

This chapter will describe the implementation of DDoS target selection scenarios. The added Java simulator functionality and the way in which log trace file parser results can be corroborated will be described.

## 4.1 Assessing Normal Pastry Peer-to-Peer Network Behaviour

A Zipf distribution is used to distribute key, value pairs across the Pastry peer-to-peer network. By plotting the most popular request key node, the Zipf distribution will be highlighted illustrating one way of defining normal behaviour for the network.

The following results (the first twenty) were parsed from the network simulator trace log:

1	66	E8DC 057D 3346 E56A <sup>2</sup>
2	51	7014 2F66 475A E2FB
3	18	743D A23F 34EE 7ADC
4	18	A33B EB13 98D6 87CE
5	18	BDC1 408A 91F1 61DA
6	12	4FDE 6B87 A266 EECE
7	11	F747 5B81 FDC4 723E
8	9	2CE6 DDC8 4D93 A3A1
9	9	5E56 CEB6 A3EE 0EEA
10	8	4C7A 18C8 BAA8 9197
11	8	DB63 FCA6 2705 7D07
12	7	B85A B32E AA57 2C80
13	7	E897 FCA0 B3C8 AB2A
14	5	18B6 6E1E E3DC 1425
15	5	8CE5 AF29 FDC1 BBA6
16	4	037D 6584 7C20 83E7
17	4	13F3 0892 06D6 A7AF
18	4	15C4 97CD B218 2AB6
19	4	1E86 56EA 42E4 19DC
20	4	5656 E9EF 8C78 78FB

The first column will be used as the X axis for the graph, and is an abbreviation for the target key(k). The most popular request key is E8DC 057D 3346 E56A<sup>2</sup>.

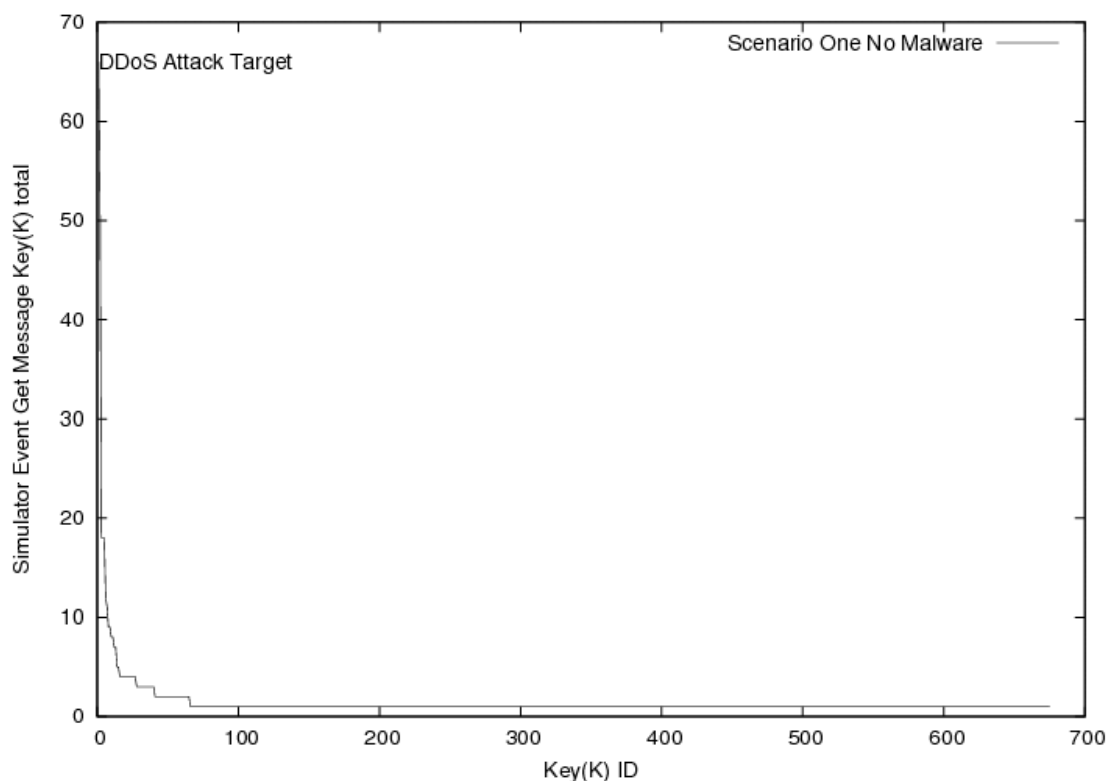


Figure 4.1: Pastry Peer-to-Peer Network Simulator Zipf Message Distribution

The graph illustrates how total get messages key (k) follows a Zipf distribution, which allows peer-to-peer simulator normal behaviour to be measurable.

## 4.2 Implementation Tools and Environment

The implementation environment for the Pastry peer-to-peer simulator was Java Runtime Environment 1.6.0\_02-b06. Because the simulator uses Java Generics, Java 5 is the minimum requirement for running the simulator. Eclipse 3.2.2 was used to develop and test the simulator. Perl v5.8.8 is used to parse the simulator trace log file. GNUPlot 4.2 is used to generate and display plotted graphs of simulator and scenario behaviour. ImageMagick::convert 6.3.0 converts gnuplot postscript output to gif files.

## 4.2.1 Java Simulator Implementation

The normal behaviour of the peer-to-peer simulator is started from the `sim.main.MilesSimulator` class. The main method defines parameters including “simulator run identity, workload name, filename to save trace log files too, seed, and number of Nodes in the peer-to-peer network”.

The `sim.main.Simulator.main` method is then called with the defined parameters. These are checked to ensure they are valid. If the parameters are acceptable a network topology, and simulator workload is created. The simulator starts processing events:

```
// load nodes appropriately
topology.load((NodeLoader)Class.forName(nodeloader).getConstructor().newInstance());

// initialise workload
Object[] workloadParams = { params };
Workload w = (Workload)
Class.forName(workload).getConstructor(String[].class).newInstance(workloadParams);

w.simulationStart();

// simulate!
try {
    while (Events.runNextEvent()) {}
} catch (Exception e) {
    e.printStackTrace(System.err);
    Global.stats.logCount("Sim" + SEPARATOR + "FatalError");
}
```

Once events have been exhausted, the `workload.simulationFinished` method is called.

The “`sim.net.overlay.dht.ddos.KeysTest`” is the workload which has been used during implementation. The normal behaviour is to create a specific number of peers which are then connected to the simulated overlay network.

Once all peers have been created the `sim.workload.Default.fastJoinAllNodes` method is called. This ensures that all routers and peers are joined together.

One million keys are then distributed across the peer-to-peer network by calling the `sim.net.overlay.dht.PeerData` method using a Zipf distribution.

One thousand requests key (k) events are then performed to exercise the peer to peer network by calling the `sim.workload.Default.generateRandomGets` method. Information on each request is logged to a trace file.

**The additional behaviour** is called from the `sim.net.overlay.dht.ddos.KeysTest` class.

The `sim.net.overlay.dht.ddos.malware.PopulateMalware` class constructor takes a malware percentage, and number of peers as parameters. A `HashMap` is created with a number of slots equal to the number of peers' parameter. A `java.util.Random` number generator creates a range of numbers which equals the malware percentage parameter. Please refer to *Figure 3.6* for a visual description of this process. These numbers set the equivalent "slot" in the `HashMap` recording that a specific peer has malware.

Peers are created in a "for loop" with a integer counter reference. Every peer has its counter reference checked with `PopulateMalware.isItMalware` which checks to see if malware has been assigned to the peer. If the test is true, a value is set in the `sim.net.overlay.dht.pastry.PeerBase` class. The `PeerBase` super class contains core values and functionality for the `Peer` class. The `hasMalware` Boolean value is set to true if the peer has malware.

Once all peers have been created, a file named "malwareFile" containing all the peer identities which have malware will be closed. This file is used by the Perl parser to see if a peer in the trace log file has malware.

The final piece of additional behaviour has been added to the `PeerBase` class. The integer variable `totalNumberMessagesReceived` is used to store how many messages a node receives and does not forward. This is incremented and logged to the trace log whenever an outward request message finds its destination or a return reply message is received by the original source of the request key(k).

## 4.2.2 Simulator Trace Log Parser Implementation

The implementation of the trace log parser started off with more logging being added to the `output-100-0.txt` simulator trace log. This had the advantage of assisting in understanding the Java simulator by highlighting the best way of parsing outward and return request messages. There was also a disadvantage of making the logs more difficult to read.

Once the structure of the outward and return request messages was understood a conscious implementation decision was made to make minimal changes to the Java simulator code. By creating an additional file which only contained malware peers identities, the existing trace log file could remain unchanged. This decision meant that the simulator could be ring-fenced, with the trace log parser being more self-contained.

The volume of data in the simulator trace log file led to a number of validation files being added to the implementation. All files have the format "filename"[**optional**\_percentage of malware figure].

“**numRequests**” provides sorted details on how many requests have been made for a specific key. They are referenced by outgoing message identity.

“**timing**” allows number of hops and RTT for individual messages to be identified.

“**avgRTT**” provides average RTT and number for messages grouped by request key(k).

“**plotter**” single file provides all plotting information for all three scenarios.

The validation files allow for testing of the parser.

For example the seventh result for plotting scenario one with no malware is key(F747 5B81 FDC4 723E) with eleven requests.

By examining **numRequests**, the number of requests sent to key (F747 5B81 FDC4 723E) are:

```
7133 >>> F747 5B81 FDC4 723E
7257 >>> F747 5B81 FDC4 723E
3987 >>> F747 5B81 FDC4 723E
7908 >>> F747 5B81 FDC4 723E
5487 >>> F747 5B81 FDC4 723E
4659 >>> F747 5B81 FDC4 723E
7036 >>> F747 5B81 FDC4 723E
7378 >>> F747 5B81 FDC4 723E
9218 >>> F747 5B81 FDC4 723E
7692 >>> F747 5B81 FDC4 723E
6328 >>> F747 5B81 FDC4 723E
```

By examining the **timing** file:

```
7133 >>> 7133|0457 F467 5A1D 17F9|F747 5B81 FDC4 723E|2|816|0|40
7257 >>> 7257|C0A6 6A3E 5293 9ED7|F747 5B81 FDC4 723E|3|1195|0|40
3987 >>> 3987|D446 2BE5 8556 BCD2|F747 5B81 FDC4 723E|3|1041|0|40
7908 >>> 7908|C27B A750 2E01 7384|F747 5B81 FDC4 723E|3|937|0|40
5487 >>> 5487|2B19 FDA9 8109 DC46|F747 5B81 FDC4 723E|3|954|0|40
4659 >>> 4659|BABD 0966 30B3 18A0|F747 5B81 FDC4 723E|2|690|0|40
7036 >>> 7036|69C4 1216 DD84 1874|F747 5B81 FDC4 723E|3|904|0|40
7378 >>> 7378|F8BE C1B0 34E4 2797|F747 5B81 FDC4 723E|2|678|0|40
9218 >>> 9218|D127 BE08 94B3 A5F9|F747 5B81 FDC4 723E|3|1077|0|40
7692 >>> 7692|F8BE C1B0 34E4 2797|F747 5B81 FDC4 723E|2|678|0|40
6328 >>> 6328|93FC 5CF6 C658 1E4B|F747 5B81 FDC4 723E|3|1018|0|40
```

By totalling the RRT for the eleven messages, and dividing by the popularity an average RTT of 908 is arrived at.

This corroborates the entry in the **plotter** file which is:

**11 908 F747 5B81 FDC4 723E**

### 4.2.3 Implementing Scenarios Two and Three

Whilst implementing scenario two which involved plotting request key (k) messages using average message RTT plotting results was different (please refer to evaluation section) from a Zipf distribution.

In a non-simulator peer-to-peer network environment the characteristics of server load, connection negotiation, and network latency (please refer to Figure 3.5) would be more realistically represented by the real-world environment.

It is questionable if the simulator represents the real world characteristics of a peer under load. A decision was made to add a peer loading factor which would make the scenario more realistic. This loading factor is illustrated in scenario three.

**Scenario three** involved adding a “total messages received” counter for any request message which were received and not forwarded to the Java simulator trace log. These included both outward and reply messages. For example:

```
recv GetMessage(2395) from 23F5 FA29 9858 2F84 for 50E0 A3F3 E77B C109 2 hops, 676 ms, 0 resents, 40 size 1
```

```
recv GetReplyMessage(3596) from 5197 8FA2 DB7A 93E3 for 23F5 FA29 9858 2F84 1 hops, 352 ms, 0 resents, 44 size 1
```

One outward and one reply message have been recorded in the trace log file. These relate to a peer, not a key. Rather than recording the additional peer loading factor in the trace log, it was added by the parser.

The parser had additional functionality added. If a message count exceeds sixty, an additional five hundred mill-seconds are added to the individual messages RTT. This will have the effect of increasing the average RTT for all message requests for Key (K).

The information being plotted for scenario three is based on message popularity from scenario one **divided by** the average RTT (with loading factor if applicable). The reasoning behind the loading factor is that it is better to select a target which is genuinely busy rather than a peer with a poor RTT network connection.

The loading factor is enabled by using the --timing parser command line option. For example running the parser with no Malware gives you the following results:

No loading factor

1	66	767.727272727273	<b>0.0859680284191829</b>	E8DC 057D 3346 E56A
2	51	732.490196078431	<b>0.0696255052600584</b>	7014 2F66 475A E2FB

The average RTT times are very similar. The result clearly indicates that key (E8DC 057D 3346 E56A) is the target.

With loading factor

1	66	957.121212121212	<b>0.0689567832832041</b>	E8DC 057D 3346 E56A
2	51	732.490196078431	<b>0.0696255052600584</b>	7014 2F66 475A E2FB

Key (E8DC 057D 3346 E56A) has had the loading factor applied to messages over 60. The average RTT has increased with the addition of the loading factor. The results of popularity / average RTT indicate that key (**7014 2F66 475A E2FB**) is now the target.

By adding a load factor to the average RTT for key (E8DC 057D 3346 E56A) it has lost its popularity.

### 4.3 Removing Irrelevant data from the Simulator trace log

A small number of individual messages have a hop count, and RTT of zero. This occurs when the node sending the request is the holder of the key (k) which is being looked for.

For accuracy in calculating average RTT groups of request messages (even if there is only one) must have a RTT great than zero. This approach avoids division by zero errors, and plotting errors.

## Chapter Five: Evaluation

The evaluation chapter will assess the results of parsing the simulator trace log, applying the gathered data to scenarios for request Key (k) popularity, average round trip time (RTT), and loading factor. The number of peers in the pastry network is one hundred. One thousand request messages are used in section 5.1.1, 5.1.2, and 5.1.3. Larger volumes of request messages (two, four and eight thousand) are used in section 5.2. Normal and percentage malware peer-to-peer network behaviour will be plotted using GNUPlot. Tab delimited data will be parsed from the simulator trace file using Perl 5.

### 5.1 Target Selection Scenarios

#### 5.1.1 Scenario One

Scenario one involves grouping request messages for the same key (k), totalling them and sorting the results into descending order. Figure 5.1 illustrates a Zipf distribution for key requests. This provides a definitive target of key **E8DC 057D 3346 E56A**. The graph in Figure 5.2 illustrates malware populations from minor to total. The results the two most popular keys for malware coverage for ten, twenty, and thirty percent are close together. This makes the selection of a single consistent target unreliable for malware coverage under forty percent.

Malware Coverage	First Position Key, (Total)	Second Position Key, Total
10%	7014 2F66 475A E2FB (7)	E8DC 057D 3346 E56A (7)
20%	7014 2F66 475A E2FB (14)	E8DC 057D 3346 E56A (12)
30%	E8DC 057D 3346 E56A (19)	7014 2F66 475A E2FB (17)
40%	E8DC 057D 3346 E56A (34)	7014 2F66 475A E2FB (15)

Once malware coverage reaches forty percent there is a greater difference between the two most popular keys, making the selection of a single target easier.

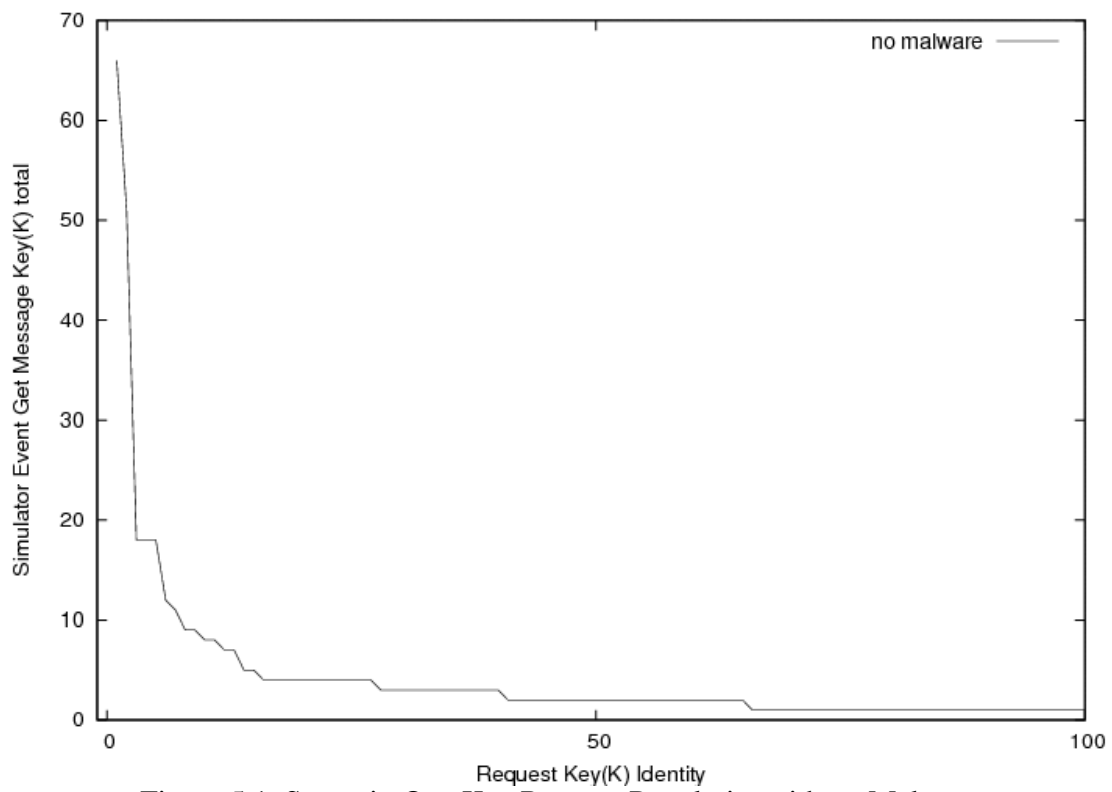


Figure 5.1: Scenario One Key Request Popularity with no Malware

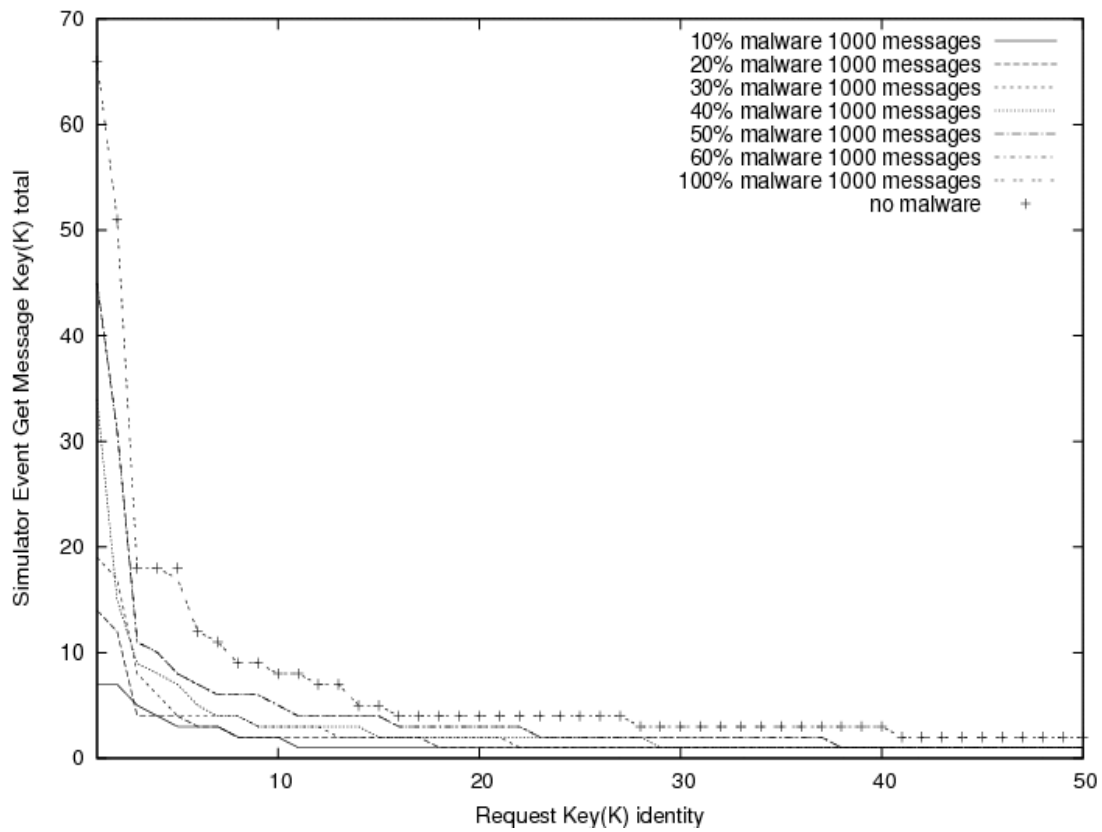


Figure 5.2: Scenario One Key Request Popularity with Varying Malware

### 5.1.2 Scenario Two

The second scenario uses average round trip time (RTT) for each group of results based on popularity of key request which was the criteria for scenario one. The results for no malware and varied percentages of malware coverage show an inconsistent graph with no clear distribution of results. This is illustrated by Figure 5.3. It would be convenient to blame this situation on the lack of simulated peer and network load, but even with a production environment with realistic delays for hardware, software, and network average round trip time may not be consistent enough for a peer-to-peer network to use average RTT as a way of selecting a target.

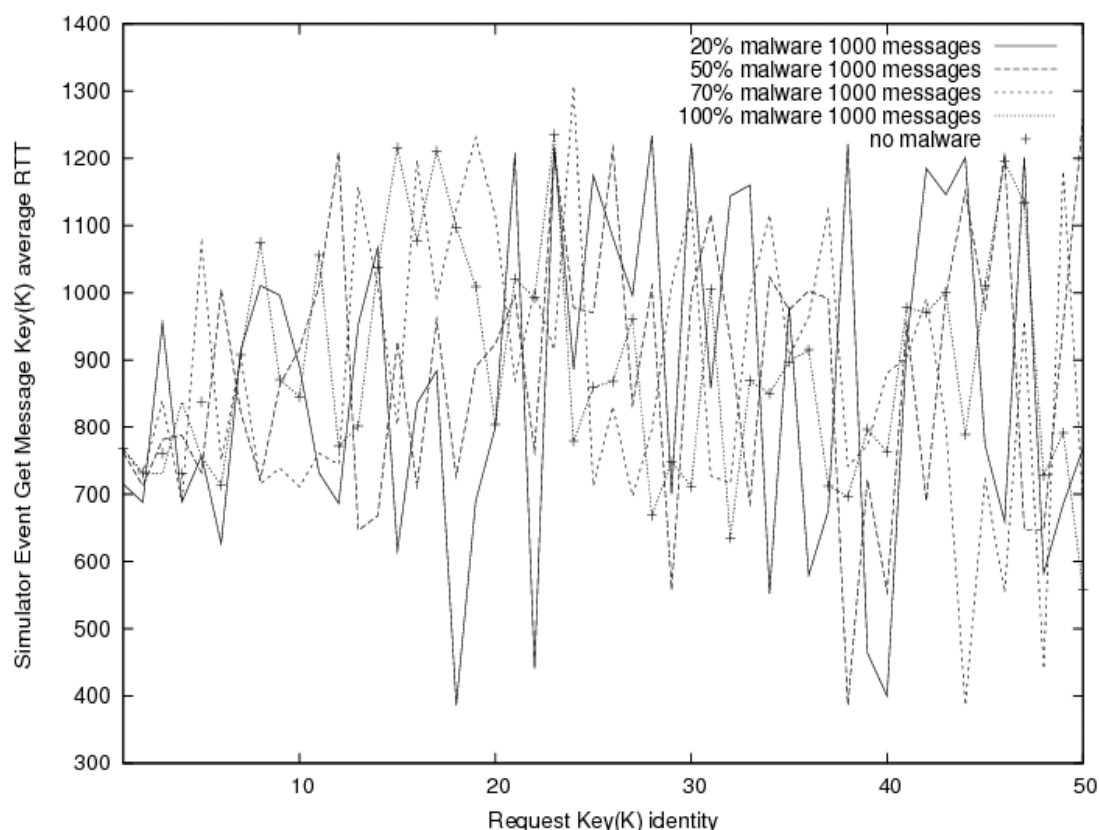


Figure 5.3: Scenario Two Key Request Average RTT with Varying Malware

The sort of factors which will make average RTT inconsistent in a non-simulator peer-to-peer environment include the way routers and network interfaces are configured, the operating system priority given to each individual peer or the traffic load on a local area network which is compromised by a bottleneck caused by a bridge to a neighbouring network. All these measurements assume a wired network. When a peer is located on a mobile device in a wireless environment changes will be more extreme. Therefore plotting average RTT for a non-simulated environment is likely to be as varied as Figure 5.3 and is therefore not good for target selection. Scenario three presents an alternative way of utilising RTT for target selection.

### 5.1.3 Scenario Three

By dividing the total number of requests messages for a particular key with the average round trip time for the messages, an improvement in target selection measurement can be achieved compared to scenario two.

Malware	First Position Key, (loading)	Second Position Key (loading)
10%	7014 2F66 475A E2FB (0.010)	E8DC 057D 3346 E56A (0.006)
20%	7014 2F66 475A E2FB (0.019)	E8DC 057D 3346 E56A (0.014)

30%	7014 2F66 475A E2FB (0.026)	E8DC 057D 3346 E56A (0.020)
40%	E8DC 057D 3346 E56A (0.034)	7014 2F66 475A E2FB (0.021)

The results for first and second target ranking are very similar to scenario one with an alternative target for ten and twenty percent coverage compared to no malware. It is unclear how accurately average RTT is represented by the simulator but taking two measurements of popularity and average RTT improve on the results for scenario one by taking average RTT into consideration as well as popularity. Because of the distributed environment average RTT is important to consider. A definite way of improving the results for scenario three would be to change the way that average RTT and popularity are used to calculate the plotted result. By placing more or less emphasis on the average RTT a greater degree of accuracy may be obtainable. The adjustment of the artificial five hundred millisecond loading for message counts over sixty may also improve accuracy.

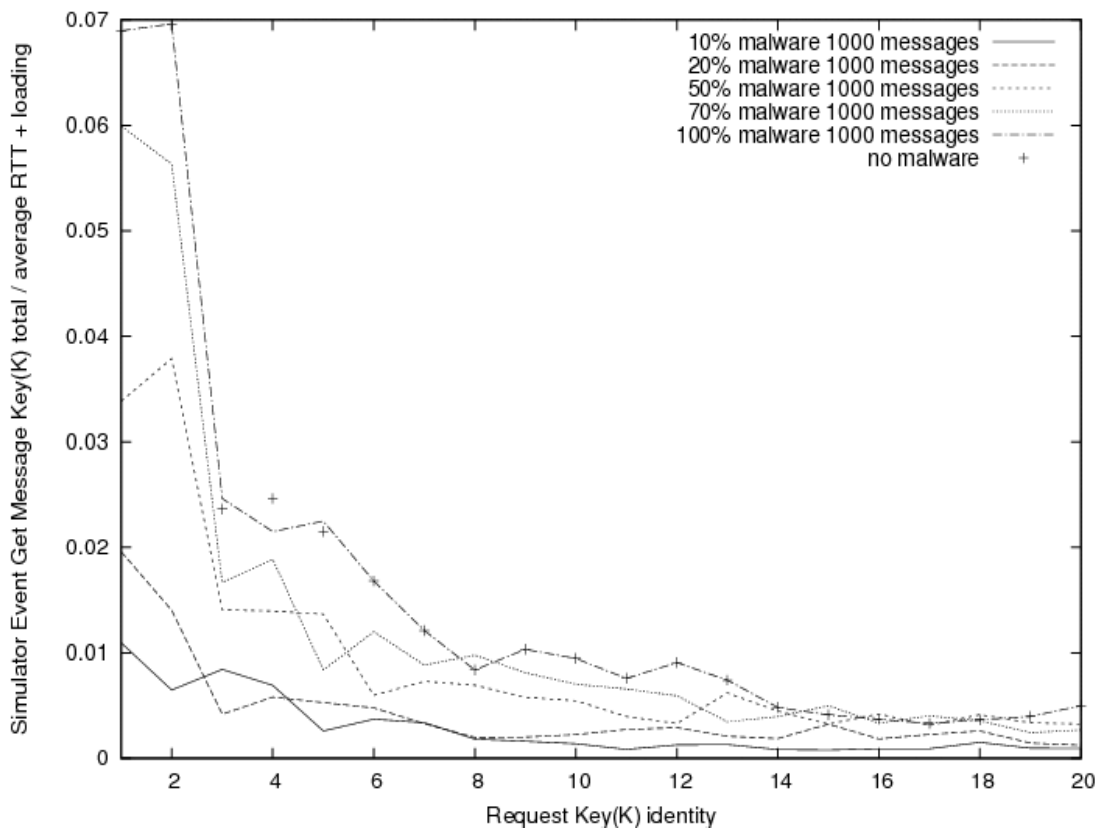


Figure 5.4: Scenario Three Key (Request Popularity divided / Average RTT) Malware

## 5.2 Target Selection Scenarios - Larger Numbers of Request Messages

### 5.2.1 Scenario One

By increasing the number of request messages for scenario one the characteristics for target selection change. For message requests of two, four, and eight thousand the most popular key E8DC 057D 3346 E56A is selected consistently with ten percent malware coverage of the network.

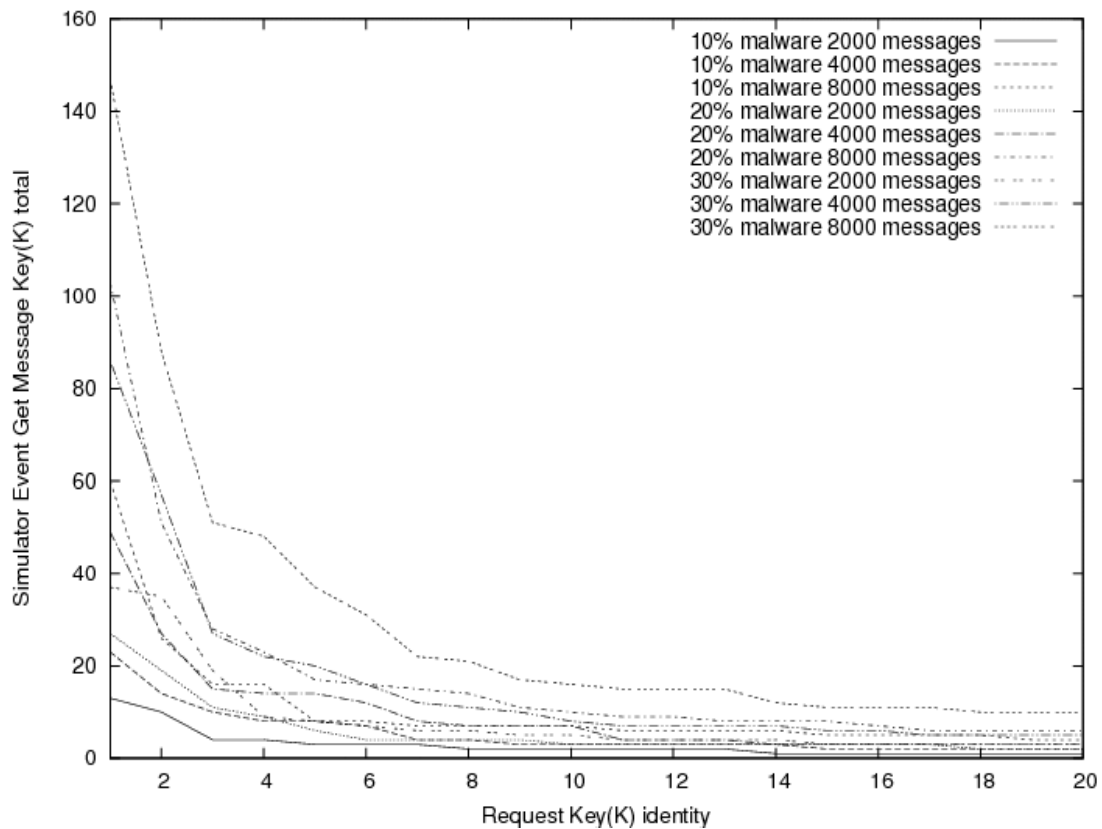


Figure 5.5: Scenario One Key Request Popularity with varying Messages and Malware

#### Two Thousand Request Messages

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (13)	7014 2F66 475A E2FB (10)
20%	E8DC 057D 3346 E56A (27)	7014 2F66 475A E2FB (19)
30%	E8DC 057D 3346 E56A (37)	7014 2F66 475A E2FB (35)

### Four Thousand Request Messages

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (23)	7014 2F66 475A E2FB (14)
20%	E8DC 057D 3346 E56A (49)	7014 2F66 475A E2FB (27)
30%	E8DC 057D 3346 E56A (86)	7014 2F66 475A E2FB (57)

### Eight Thousand Request Messages

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (60)	7014 2F66 475A E2FB (26)
20%	E8DC 057D 3346 E56A (103)	7014 2F66 475A E2FB (51)
30%	E8DC 057D 3346 E56A (147)	7014 2F66 475A E2FB (88)

## 5.2.2 Scenario Three

By increasing the number of request messages for scenario three the characteristics for target selection change. For message requests of two, four, and eight thousand the most popular key E8DC 057D 3346 E56A is selected consistently with ten percent malware coverage of the network. The only exception to this condition is the two thousand request messages with thirty percent malware coverage.

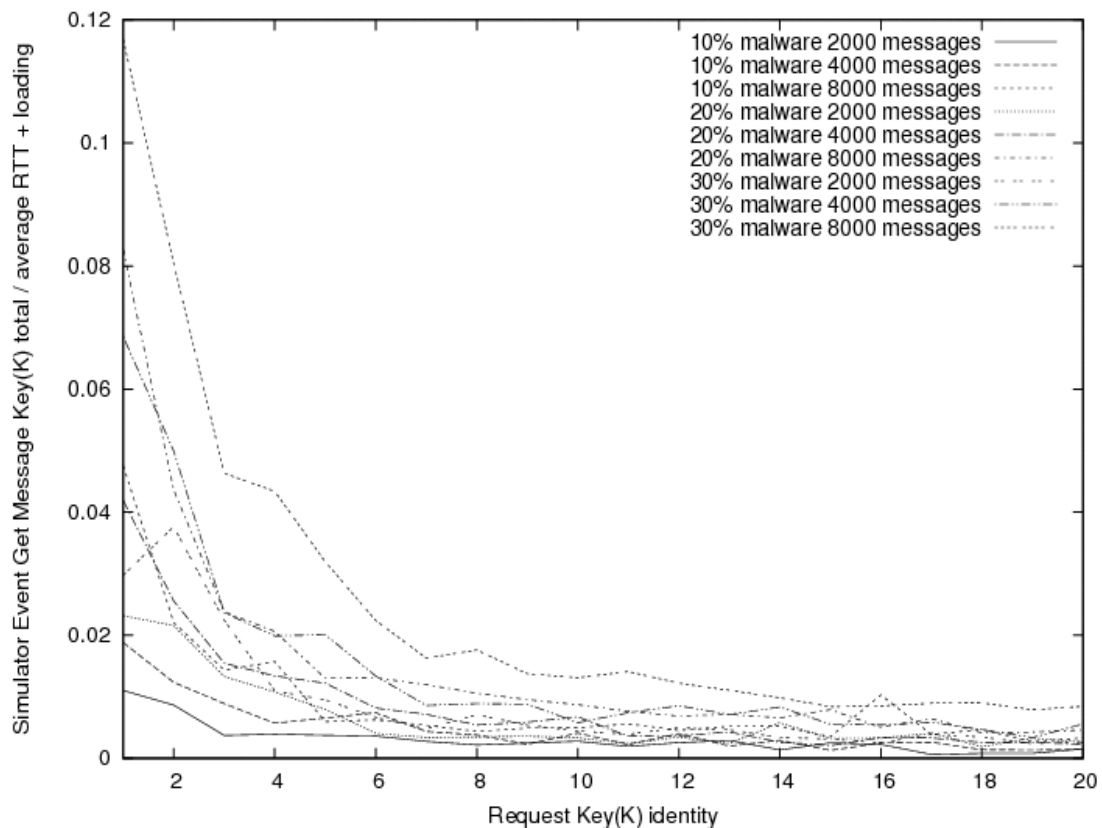


Figure 5.6: Scenario Three Key (Request Popularity divided / Average RTT) with varying Messages and Malware

### **Two Thousand Request Messages**

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (0.011)	7014 2F66 475A E2FB (0.008)
20%	E8DC 057D 3346 E56A (0.023)	7014 2F66 475A E2FB (0.021)
30%	<b>7014 2F66 475A E2FB</b> (0.037)	E8DC 057D 3346 E56A (0.030)

### **Four Thousand Request Messages**

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (0.019)	7014 2F66 475A E2FB (0.012)
20%	E8DC 057D 3346 E56A (0.042)	7014 2F66 475A E2FB (0.026)
30%	E8DC 057D 3346 E56A (0.069)	7014 2F66 475A E2FB (0.050)

### **Eight Thousand Request Messages**

Malware	First Position Key	Second Position Key
10%	E8DC 057D 3346 E56A (0.048)	7014 2F66 475A E2FB (0.022)
20%	E8DC 057D 3346 E56A (0.083)	7014 2F66 475A E2FB (0.044)
30%	E8DC 057D 3346 E56A (0.117)	7014 2F66 475A E2FB (0.080)

## **5.3 Botnet Creation**

Within the Pastry peer-to-peer simulator a botnet is created by selecting a specific percentage of randomly selected peers from the total number of peers. These represent a malware percentage (illustrated in Figure 3.6). This simulator workload creates a maximum number of peers sequentially which are then joined to the peer-to-peer overlay network. The overlay is in turn joined to the simulated physical network. This approach does not fulfil two characteristics of real-world botnet creation in a peer-to-peer environment.

The behaviour of nodes in peer-to-peers networks reflects the habits or users who utilise the applications the network supports. Over a period of time, a peer is actively connected to other peers in the network contributing in query forwarding, response, and initiation; or disconnected from the network. The behaviour of the selected simulator workload does not allow for repeated joining and leaving of peers. The percentage that a peer is available over a particular timeframe will be a factor in whether a peer becomes infected with malware. Peer popularity is another factor which leads to the spread of malware in a peer-to-peer network environment. If a peer has a role of forwarding a large number of requests, or is the destination for large numbers of popular key assets, the likelihood of the peer becoming infected with malware is higher than a peer whose assets are only requested rarely.

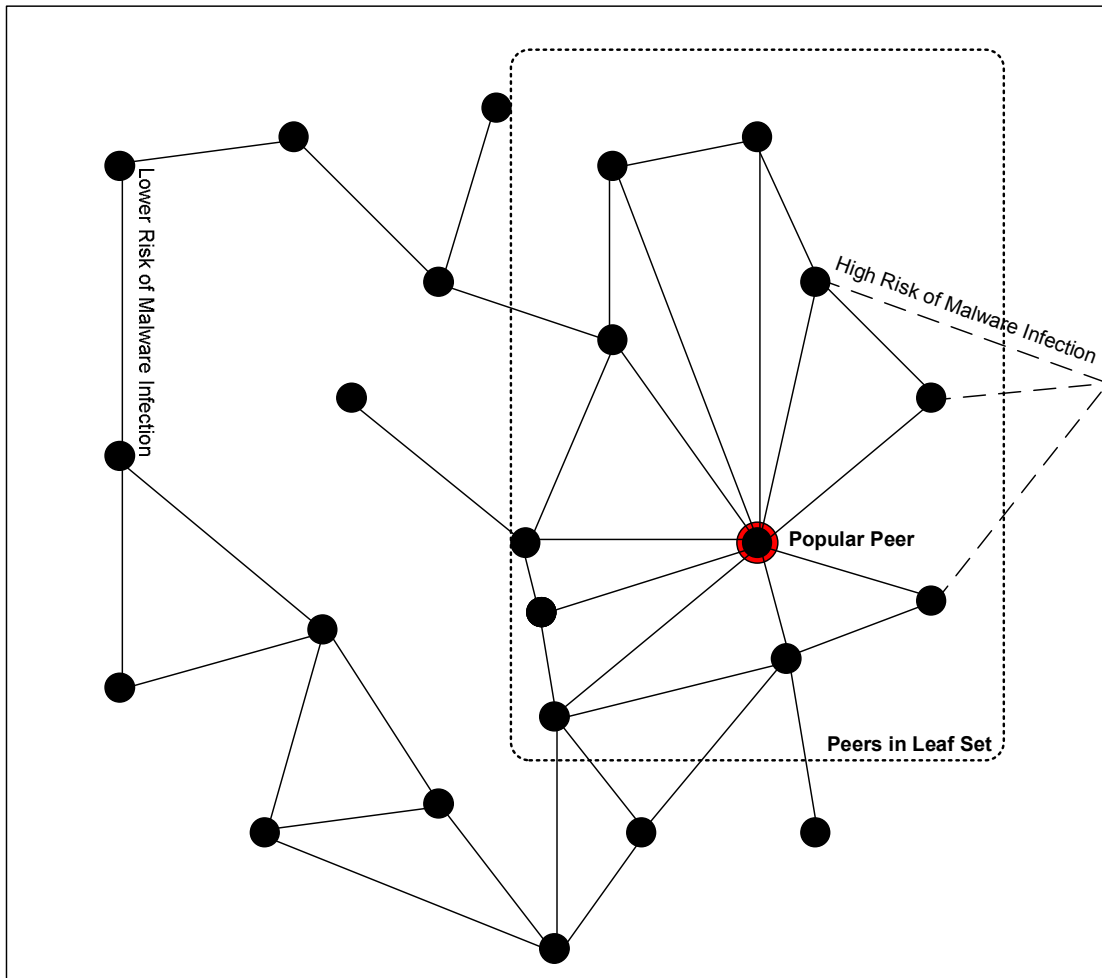


Figure 5.7: Malware Infection is higher with Popular Peers

Figure 5.7 illustrates part of a peer-to-peer network. The popular peer forwards and services a large number of requests. The popular node has a large number of leaf nodes which forwards request to the popular peer. If any of the leaf nodes were infected with malware, there is high risk of the popular, and leaf set peers becoming infected. The less popular peers which have fewer peer connections will have a better chance of avoiding infection by malware [35]. By using a workload which did not take peer joining and leaving habits, and spread of malware due to popularity into account more effort could be focussed on the target selection objective.

## Conclusion

By plotting graphs which present scenarios which illustrate how a suitable peer target can be selected, a view on the size of botnet and number of request messages needed to consistently select a target can be arrived at. If the botnet is too small the target selection will not be realistic. If the botnet is too large the target selection will be accurate but stealthy characteristics will be reduced because the larger the botnet, the greater the chance of discovery.

The botnet creation phase involved a percentage of the total peer-to-peer network size being used to define the size of the botnet. The pastry peer-to-peer java simulator allowed a workload which allowed random peers to be classified as having malware as they were created. This approach allowed for minimum changes to the simulator with more effort assigned to the simulator workload event trace log parser which analysed network behaviour and extracted relevant data for plotting target selection.

The background research section describes how the Pastry protocol changes routing information as peers join and leave the network. A realistic event within botnet creation is for malware peers to be created over a period of time which is not the same as the simulator workload creating all malware peers at the same time. Another simulator workload characteristic allows a realistic percentage of request messages to be lost, perhaps by a peer leaving the network. This realistic simulator feature has not been used by the implemented workload. By omitting peer leaving and joining behaviour, and missing request messages from the implemented workload effort could be focussed on meeting the primary project aim of researching stealthy target selection within a Pastry peer-to-peer network.

The second target selection scenario utilised average round trip time of request messages for the same key. This metric was found to be inconsistent and did not follow the Zipf distribution which was used by the simulator for populating the peer-to-peer network with (key, value) pairs. The simulator did implement network link timing information, but it is not realistic to expect the simulator to implement the equivalent timing conditions found in a real world peer-to-peer network. These would include network latency, router delay and computational load experienced by busy peers. The secondary aim of implementing stealthy control centre is discussed in this document, but has not been physically implemented because of time constraints.

A good starting point for future work would be to compare the target selection findings illustrated in this document with an equivalent real world peer-to-peer network. Including peers leaving and joining, request message loss, and analysis of conditions which increase round trip time would provide an invaluable comparison to the simulator target selection analysis and findings. By implementing stealthy control centre functionality, and completing the DDoS lifecycle by launching a realistic DDoS attack would allow the research presented in this document to be more fully implemented and observed than is currently the case.

## Bibliography

1. *The Art of Shoplifting* <http://danny.oz.au/freedom/cases/Rabelais.html> Rabelais (student newspaper) Viewed July 2, 2007.
2. Clayton R., February 2006 *Complexities in Criminalising Denial of Service Attacks*
3. Naraine R., November 16, 2006 “Pump and Dump” Spam Surge Linked to Russian Bot Herders. <http://www.eweek.com/article2/0,1895,2060235,00.asp> Viewed August 1, 2007.
4. *Recognize phishing scams and fraudulent e-mails* <http://www.microsoft.com/protect/yourself/phishing/identify.msp> September 14, 2006 Microsoft Viewed August 1, 2007
5. Delio M., March 18, 2004 *Cashing In on Virus Infections* <http://www.wired.com/techbiz/it/news/2004/03/62558> Viewed August 5, 2007.
6. Want R., Pering T., Tennenhouse D., IBM Systems Journal, Volume 42, Number 1, 2003. *Comparing autonomic and proactive computing.*
7. Rowstron A. and Druschel P., *Pastry: Scaleable, decentralized, object location and routing for large-scale peer-to-peer networks.* Houston: Rice University. Cambridge UK: Microsoft Research Ltd.
8. Orion E., August 9 2007, the Enquirer. *Exploit Hunter Ponders Ethical Hacking – the twilight world of internet insecurity* <http://www.theinquirer.net/default.aspx?article=41589> Viewed August 14, 2007.
9. WABISabiLabi Vulnerability Market Place. Switzerland <http://www.wslabi.com/wabisabilabi/initPublishedBid.do?> Viewed August 10, 2007.
10. *Melissa virus creator jailed* <http://news.bbc.co.uk/1/hi/world/americas/1963371.stm> May 2, 2002. Viewed August 20.
11. Naoumov N., Ross K, *Exploiting P2P Systems for DDoS Attacks.* Brooklyn, NY: Polytechnic University.
12. *Autonomic Computing, IBM.* [http://researchweb.watson.ibm.com/autonomic/research/P2P\\_File\\_Sharing\\_History\\_and\\_Introduction](http://researchweb.watson.ibm.com/autonomic/research/P2P_File_Sharing_History_and_Introduction) [http://filesharingplace.be/guides/filesharing\\_history.php](http://filesharingplace.be/guides/filesharing_history.php) Viewed September 5, 2007.
13. Ratnasamy S, Francis P, Handley M, Karp R, and Shenker S. *A Scalable Content-Addressable Network.* USA: University of California. USA: AT&T Center for Internet Research at ICSI.
14. Stoica I., Morris R., Karger D., Kaashoek Frans M., Balakrishnan H. *Chord: A Scaleable Peer-to-Peer Lookup Service for Internet Applications.* USA: MIT Laboratory for Computer Science.
15. Houle J. and Weaver G., October 2001, *Trends in Denial of Service Attack Technology.* CERT Coordination Center.

16. Malkin M. February 15, 2006, The Islamists War on the Internet  
<http://michellemalkin.com/2006/02/15/the-islamists-war-on-the-internet/>  
Viewed September 5, 2007.
17. Knight W., March 13, 2000, *Hackers question Denial of Service as political protest.*  
<http://news.zdnet.co.uk/security/0,1000000189,2077647,00.htm>  
Viewed August 15, 2007
18. Thomas D., June 1, 2005. *Deterrence must be the key to avoiding DDoS attacks.*  
<http://www.vnunet.com/computing/analysis/2137395/deterrence-key-avoiding-ddos-attacks>  
Viewed August 15, 2007
19. Poulsen K, August 26, 2004. *FBI busts alleged DDoS Mafia*  
<http://www.securityfocus.com/news/9411>  
Viewed August 20, 2007
20. HTTP – Hypertext Transfer Protocol  
<http://www.w3.org/Protocols/>  
Viewed August 20, 2007
21. Internet Relay Chat  
[http://en.wikipedia.org/wiki/Internet\\_Relay\\_Chat](http://en.wikipedia.org/wiki/Internet_Relay_Chat)  
Viewed August 20, 2007
22. Leyden J., April 2006, 2002. *Crackers favour war dialling and weak passwords.*  
[http://www.theregister.co.uk/2002/04/26/crackers\\_favour\\_war\\_dialling/](http://www.theregister.co.uk/2002/04/26/crackers_favour_war_dialling/)  
Viewed August 15, 2007.
23. Saha B., Gairola A., *Botnet: An Overview*  
<http://www.cert-in.org.in/knowledgebase/whitepapers/ciwp-2005-05.htm>  
Viewed August 15, 2007.
24. TCP/IP Fundamentals  
<http://www.uga.edu/~ucns/lans/tcpipsem/>  
Viewed August 15, 2007.
25. Kunwar N., March 28, 2006 *DNS Amplification Attack*  
<http://nirlog.com/2006/03/28/dns-amplification-attack/>  
Viewed August 17, 2007.
26. *The Continuing Denial of Service Threat Posed by DNS Recursion (v2.0)*, 2006. US-CERT
27. Athanasopoulos E., Anagnostakis., Markatos E, 2006. *Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets.* Greece: Hellas, Institute of Computer Science.
28. Mirkovic J., Reiher P., *A Taxonomy of DDoS Attack and DDoS Defence Mechanisms.* USA: University of Delaware, UCLA.
29. Stoll C., *The Cuckoo's Egg* ISBN-10: 0743411463
30. Asahi exercise in Manchester, Flashmob  
[http://www.flashmob.co.uk/mt/2007/06/asahi\\_exercise\\_in\\_manchester.php](http://www.flashmob.co.uk/mt/2007/06/asahi_exercise_in_manchester.php)  
Viewed September 12, 2007.

31. Jung J., Krishnamurthy B., Rabinovich R. *Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites*. USA: Cambridge, MIT Laboratory for Computer Science. AT&T Labs-Research.
32. Ting N., *A Generic Peer-to-Peer Network Simulator* USA: Saskatchewan University of Saskatchewan.
33. Anderson C., *The Long Tail*. June 2006. ISBN: 1401302378
34. Rappoport A., *Tuning Search: Analytics, Search Logs, and Best Bets*  
<http://www.searchtools.com/slides/ess06/tuning-search.html>  
Viewed August 5th, 2007
35. Ramachandran K, Sikdar B., *Modeling Malware Propagation in Gnutella Type Peer-to-Peer Networks*. USA: New York: Rensselaer Polytechnic Institute
36. *Filesharing - Napster – History*  
[http://wiki.media-culture.org.au/index.php/Filesharing - Napster - History](http://wiki.media-culture.org.au/index.php/Filesharing_-_Napster_-_History)  
Viewed September 27, 2007
37. *P2P and networking technologies*  
<http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/Intro.html>  
Viewed September 27, 2007
38. *BitTorrent tracker*  
[http://en.wikipedia.org/wiki/BitTorrent\\_tracker](http://en.wikipedia.org/wiki/BitTorrent_tracker)  
Viewed September 27, 2007
39. <http://www.prolexic.com/>  
Viewed September 27, 2007
40. *Skype: What happened on August 16, 2007*  
[http://heartbeat.skype.com/2007/08/what\\_happened\\_on\\_august\\_16.html](http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html)  
Viewed August 25, 2007
41. *Dijkstra's Algorithm*  
[http://en.wikipedia.org/wiki/Dijkstra's\\_algorithm](http://en.wikipedia.org/wiki/Dijkstra's_algorithm)

## Glossary

The following terms have been used in this document. The associated definitions have been provided

*Malware* – Malicious behaviour added to a computer system. Computers which are vulnerable with out-of-date security, no antivirus, or lack of perimeter security such as firewalls are likely targets. Malware is likely to be distributed via malicious email attachments, virus, or compromised websites.

*Zombie* – Once a computer has been infected with Malware it becomes a Zombie.

*Botnet* – A network of zombies which are likely to be connected and controlled using the Internet.

*Bot Herder* – Individuals who control a Botnet, sending instruction on when and how to attack a target.

*Target* - The victim of a Distributed Denial of Service Attack (DDoS). The target will likely provide a service which will have a presence on the Internet. The objective of the attack will be to completely or partially disrupt this service.

*Denial of service attack (DoS)* – An attack on a specific target will originate from one specific source. The source will have been compromised by Malware.

*Distributed Denial of Service Attack (DDoS)* – Same characteristics as DoS attack, but the attack will originate from multiple sources simultaneously. The importance of a Bot Herder who co-ordinates the attack from a control centre enable the DDoS attack to be synchronised.

*Control Centre* – A mechanism for instructing a Botnet to attack a target. IRC and HTTP are often used to communicate with the Botnet.

*IRC* – Internet relay chat is an open client server protocol which allows individuals to exchange text messages, and chat to each other. A large percentage of Malware has the ability for a zombie to connect to a malicious IRC server and receive instructions from a control centre.

*HTTP* – Hypertext Transfer Protocol is an open protocol used to communicate with websites, and retrieve web pages. This protocol can also be used by Malware to retrieve control centre instructions.

*Protocol* – A set of published guidelines which define and govern how a specific widely used operation should work across a network or Internet.

*Peer* – Single node within a peer-to-peer network.

*Peer-to-Peer Network* – A way of decentralising and distributing traditionally centralised services across a number of peers, interconnected by a shared network (Internet). A peer is likely to provide a small part of a larger service which is provided by the whole network. Other peers can request information from a peer. If the peer cannot fulfil the request, it will be forwarded to a peer who is more likely to help.

Website address for Java simulator, Perl trace log parser instructions, scenario test data, and operating instructions:

**<http://www.milesdavenport.com>**