ON THE APPLICATION OF LOCALITY TO NETWORK INTRUSION DETECTION: WORKING-SET ANALYSIS OF REAL AND SYNTHETIC NETWORK SERVER TRAFFIC

by

ROBERT LEE
A.B. Harvard College, 2002
M.S. University of Central Florida, 2005

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2009

Major Professor: Sheau-Dong Lang

ABSTRACT

Keeping computer networks safe from attack requires ever-increasing vigilance. Our work on applying locality to network intrusion detection is presented in this dissertation. Network servers that allow connections from both the internal network and the Internet are vulnerable to attack from all sides. Analysis of the behavior of incoming connections for properties of locality can be used to create a normal profile for such network servers. Intrusions can then be detected due to their abnormal behavior. Data was collected from a typical network server both under normal conditions and under specific attacks. Experiments show that connections to the server do in fact exhibit locality, and attacks on the server can be detected through their violation of locality.

Key to the detection of locality is a data structure called a working-set, which is a kind of cache of certain data related to network connections. Under real network conditions, we have demonstrated that the working-set behaves in a manner consistent with locality. Determining the reasons for this behavior is our next goal. A model that generates synthetic traffic based on actual network traffic allows us to study basic traffic characteristics. Simulation of working-set processing of the synthetic traffic shows that it behaves much like actual traffic. Attacks inserted into a replay of the synthetic traffic produce working-set responses similar to those produced in actual traffic. In the future, our model can be used to further the development of intrusion detection strategies.

ACKNOWLEDGMENTS

I would like to thank the Committee for their support. In particular, my advisor, Sheau-Dong Lang, has given me a tremendous amount of help and encouragement, and for that I am very grateful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION AND RELATED WORK

1.1 Introduction

The number of computers connected to the Internet continues to increase at a rapid pace. These computers are ready targets for malicious attacks. While network firewalls protect most computers, firewalls cannot stop all intrusions. Furthermore, if an intrusion is not detected, a trusted computer can be hijacked for use as a drone in a botnet or for other undesirable purposes. In particular, network servers are vulnerable to intrusions because they often sit at the boundary between the inside and outside of a firewall and receive connections from both sides. If such a server is hijacked, the attacker then has access to computers inside the firewall. Typically, a large number of users rely on the continuous availability of a server. Thus, it is important to detect server intrusions when they do occur.

The approaches to detecting malicious intrusions can be divided into two major categories: signature-based detection and behavior-based detection. Signature-based detection relies on past observation of the characteristics of a network connection that was known to be an intrusion. Those characteristics make up a signature for that type of intrusion. If a new connection matches an existing signature, it is probable that the connection represents an intrusion. One advantage of this approach is a high confidence that the new connection is indeed an intrusion. One

disadvantage is the inability to recognize an intrusion that does not match any of the existing signatures.

Behavior-based detection does not look for the characteristics of a specific type of intrusion. Instead, a profile of normal network traffic is created. A network connection that does not fit the profile is then suspect. Note that behavior-based detection will easily coexist with signature-based detection. One need not be used in lieu of the other; in fact, the two types of detection complement each other.

The particular type of behavior we wish to investigate involves the principle of locality. A practical application of locality arises in the memory cache of a modern computer. A piece of data that has been retrieved recently from memory is likely to be accessed again in the near future, so it is worthwhile to store it temporarily in the cache, where it can be accessed with less delay. Periodically, a piece of data that has not been accessed for a long time is removed from the cache to make room for newer data. For intrusion detection, we use two types of working-sets: one type contains IP addresses of incoming network connections and the other type contains the ports used by the connections. Both types function as a kind of cache, with the expectation that connections are likely to be made from IP addresses that have connected recently, and that these connections will likely reuse ports that have been used recently.

Having demonstrated that our intrusion detection system is capable of detecting locality and violations thereof, we proceed to investigate why the network traffic shows the properties of locality. To that end, we build a synthetic model of the network traffic.

1.2 Related Work

In [1], locality-based analysis was used to build a profile of outgoing network traffic from a personal computer on a local area network. The profile included two important parameters for the working-set of recently contacted IP addresses: the removal interval, or how often an IP address was removed from the set, and the size threshold. If the working-set grew above the size threshold, this violation of locality indicated that suspicious behavior was occurring. The analysis used the fact that the pattern of outgoing network traffic from a personal computer was closely correlated with human behavior, including common tasks such as email and web surfing.

A behavior-based approach has been used to detect the spread of worms [3] and viruses [5]. The authors of [3] noted that outgoing connections created by worms fail to connect more often than those from legitimate users or applications, so worms could be detected by counting these failed connection attempts. The virus detection method in [5] relied on the observation that, under normal conditions, there is a limit to the rate at which a computer makes new outgoing connections. In [6], malicious network behavior was detected by correlating network traffic with

user activity. [4] found that a human user is likely to make network connections to recently visited machines, in accordance with the principle of locality.

[7] and [8] incorporated the principle of locality to develop a worm detection system using a sliding time window of different sizes. The window included the connections made during the time interval, and the different sizes were designed to capture attacks with varying rates of connection.

Specific types of attacks with which we are concerned include network probes and denial of service (DOS) attacks. Network probes are information-gathering expeditions by an attacker prior to the launch of another, likely more damaging attack. Types of network probes include a fingerprinting attack, which is designed to identify the operating system on the target machine, and a port scan, which tries to pinpoint the ports that are vulnerable on the target machine.

A fingerprinting attack relies on malformed packets or unorthodox queries, to which various operating systems give different answers, thus identifying the particular operating system. In [11], Smart designed and tested a fingerprint "scrubber" that modifies the packet headers of all packets outgoing from a network so they are consistent and thus cannot identify the operating system of the machine that sent them.

DOS attacks target a limited resource on a server. By occupying all available instances of that resource, the attack prevents authorized users from gaining access to the resource. Prior work has investigated methods for the detection of DOS attacks.

In [12], Gil designed the MULTOPS system to monitor the rate of incoming and outgoing traffic at routers near the origin of the DOS attack, based on the observation that there is much more outgoing traffic than incoming traffic when a DOS attack is taking place. The system is designed to stop an attack soon after it begins.

In [13], Cheng observed that a legitimate user of TCP would wait for an acknowledgement to arrive before sending another set of packets. Thus, a set of packets should be sent with a period approximately equal to the roundtrip time. In contrast, a DOS attacker would send packets continuously without bothering to wait for an acknowledgement. By searching for a violation of the periodicity property, Cheng was able to detect suspicious traffic.

In [14], Feinstein analyzed statistics of specific fields in the headers of packets, such as the source IP address field, and found that the packets involved in a DOS attack did not generate values in the normal range for those statistics. The conclusion was that a DOS attack could be detected using a statistical analysis of packet headers.

A specific type of DOS attack is the SYN flood attack, which creates many half-open connections at the target to deplete system resources dedicated to such connections, and thus

prevent authorized users from connecting. In [15], Wang observes that in normal TCP traffic, the number of SYN packets is approximately equal to the number of FIN packets. Because there are many more of the former than the latter during a SYN flood, such an attack can be detected.

In [16], Lemon proposes that a system should not allocate system resources when it receives the initial SYN packet, but instead wait for the three-way TCP handshake to be completed. In this way, system resources cannot be depleted by a SYN flood.

CHAPTER 2: LOCALITY-BASED SERVER PROFILING

2.1 Introduction

As mentioned in Section 1.2 above, past studies focused on the behavior of outgoing traffic. In contrast, we investigate whether incoming connections to a network server, which include both human-initiated connections and automated network traffic, also exhibit evidence of locality. We propose that the incoming connections to a network server can be subjected to locality-based analysis to create a normal profile.

In this chapter, we describe a locality-based method for detecting anomalous incoming connections to a network server. Analysis of the normal incoming connections to the server yields a profile. We demonstrate that connections to a production server do in fact fit a profile; hence, we can use the profile to detect intrusions.

2.2 Connection Analysis

Our connection analysis algorithm watches each packet as it arrives at the server and determines to what connection it belongs and which pair of (source, destination) ports it uses. A connection is defined as a stream of packets between two IP addresses. Each connection may have more than one associated pair of ports. The algorithm keeps a working-set of incoming connections.

This working-set has variable size with removal of the least recently used connection at regular intervals of time, as described in [2]. The connection working-set stores the most recent access time for each connection.

The algorithm also associates each connection with a working-set containing its ports. Like the overarching connection working-set, the port working-sets use variable size with removal of the least recently used pair of ports at fixed removal intervals. See Figure 1 below for a diagram of the working-set structure.

Over time, the connection working-set will grow or shrink based on the behavior of the incoming connections. If new connections are always made from IP addresses already in the working-set, the size will decrease as connections are automatically removed. If new connections are always made from IP addresses not in the working-set, the size will increase if the rate of new connections exceeds the rate at which connections are removed from the working-set. Each port working-set will exhibit similar behavior based on the ports used by its associated connection.

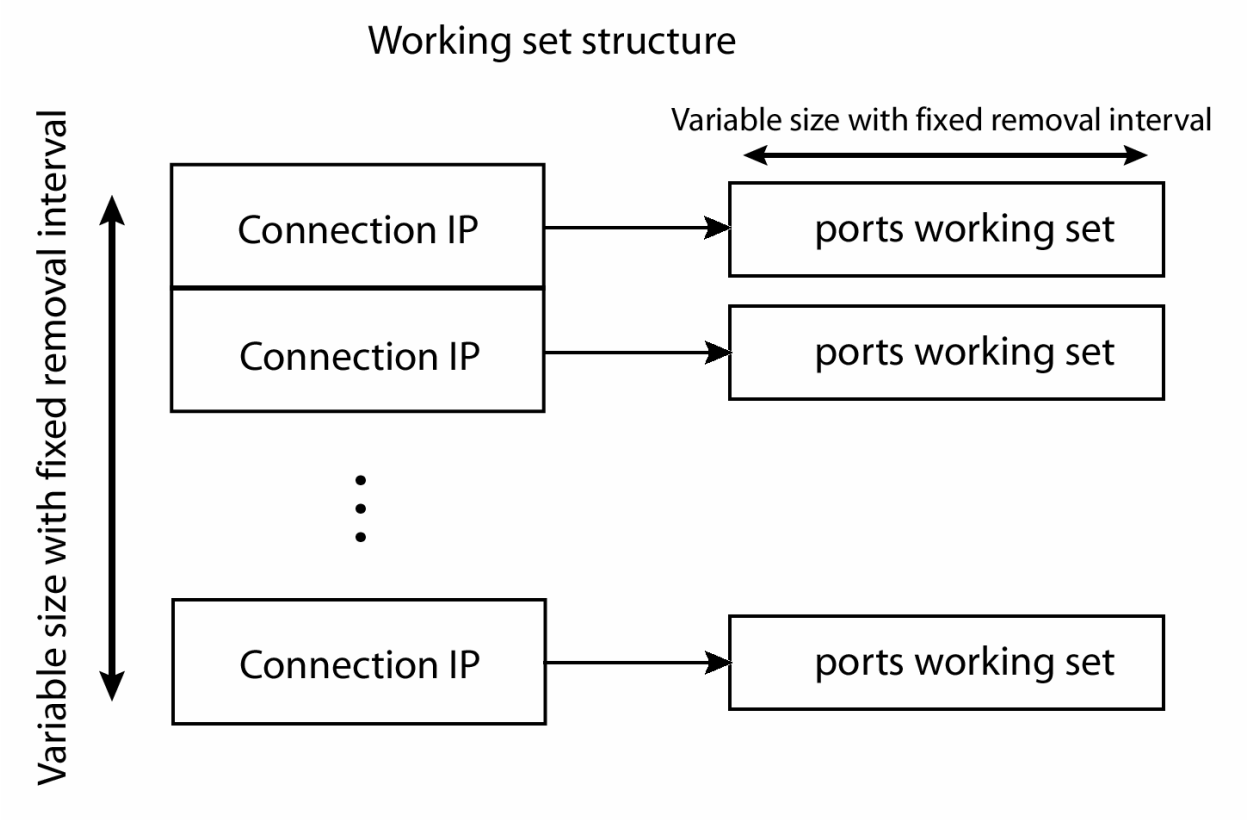

Figure 1: The structure of the working-set for incoming internal connections.

If the incoming packet belongs to a connection that already exists in the working-set, the algorithm updates the most recent usage time of the connection. If no connection exists in the working-set for the packet, a new connection is created. Likewise, the algorithm updates the most recent usage time in the port working-set if the packet uses existing ports; otherwise, it adds a new pair of ports for the packet. At regular intervals of time, the algorithm scans through the connection working-set and removes the least recently used connection. The interval is a constant parameter called the connection removal interval. The algorithm also scans through each port working-set at regular intervals and removes the least recently used pair of ports from each. The port removal interval can be adjusted independently from the connection removal

interval. Over time, the working-sets will vary in size depending on the behavior of the incoming connections and their ports.

A working-set exhibits locality if its size does not grow past a certain level, or threshold, over time. The locality arises in the idea that connections are likely to be made from IP addresses already in the connection working-set and incoming packets are likely to use ports already in a port working-set. Once a normal size threshold for a given server has been determined, the growth of the working-set past that size threshold would indicate violation of locality, and hence would detect anomalous network behavior that could correspond to a malicious attack. The two types of working-sets function concurrently to detect different kinds of attacks. Violation of locality in the connection working-set would indicate an attack originating from multiple IP addresses. Violation of locality in a port working-set would indicate an attack that connects to multiple ports. In our experiments, we will use working-set analysis to show that connections to a typical network server and their associated ports do in fact exhibit locality.

2.3 Experimental Results

We used the Wireshark network protocol analyzer [9] to capture the headers of all network packets incoming to and outgoing from a production network server for a class at our university over a period of four workdays, Tuesday through Friday, from December 5 through 8, 2006. This server, which ran the Microsoft Windows Server 2003 operating system, was used by students to

access class-related information. During the four-day period, 29666 connections were made to the server, with 2520223 incoming packets.

2.3.1 Connection Working-set Behavior

We chose a one-hour window from 13:00 to 14:00 during each day to focus our study. See Table I below for the breakdown of incoming connections and packets during the one-hour window each day. Note that there is strong consistency in the connection counts among the four days; in fact, the connection count never varies by more than 2.1% from the mean.

Table 1: Number of incoming connections and packets by day during the 13:00 to 14:00 window.

Day	Incoming Connections	Incoming Packets
Tuesday	336	31698
Wednesday	338	27436
Thursday	349	30041
Friday	344	37416

These four sets of data, one for each day from Tuesday through Friday, serve as four separate observations of connection behavior. We analyzed each of the four sets individually. Figures 2, 3, 4, and 5 below show connection working-set behavior for Tuesday, Wednesday, Thursday, and Friday, respectively. Each figure shows the size of the working-set during the one-hour period, given a certain fixed removal interval. The eight lines in each figure represent removal intervals of 2, 4, 6, 8, 10, 12, 14, and 16 seconds.

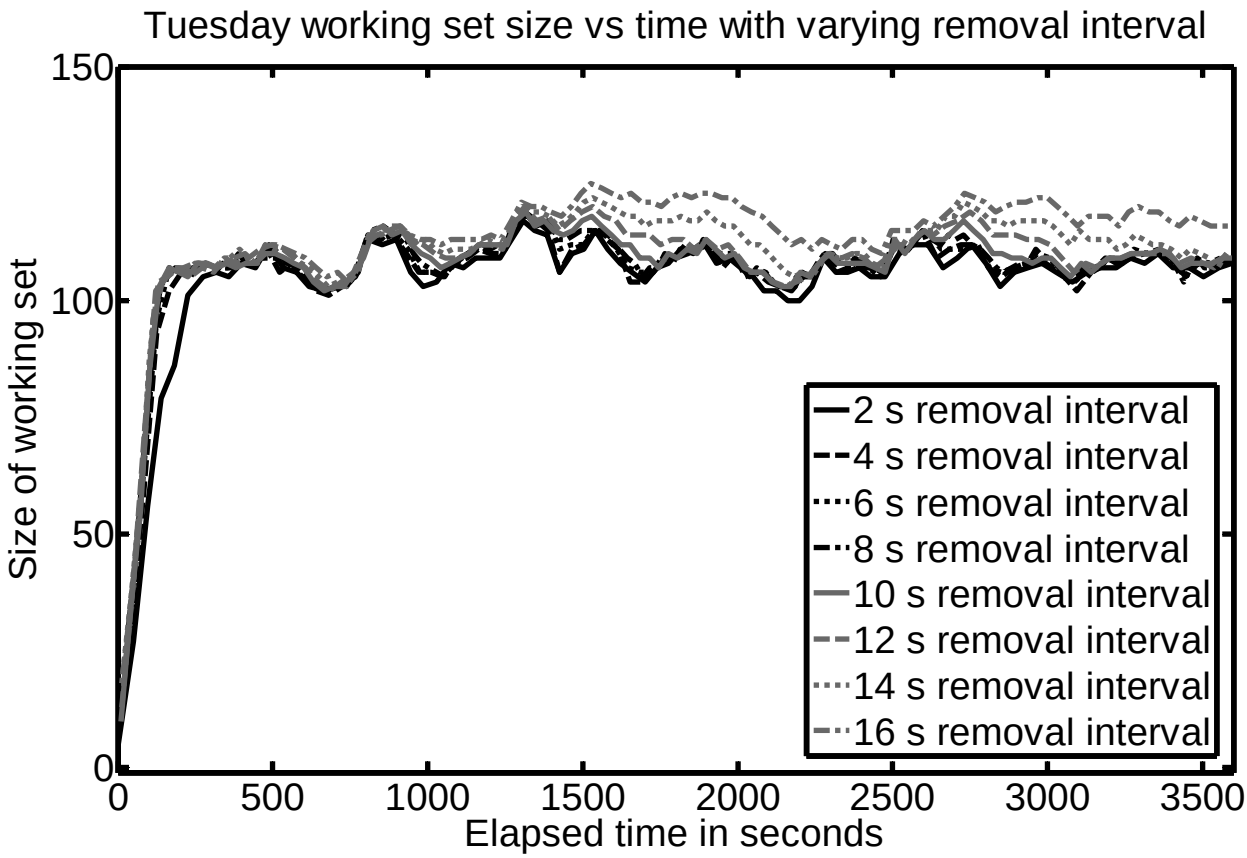

Figure 2: The size of the Tuesday connection working-set over time, given a fixed removal interval. The size does not exceed 125.

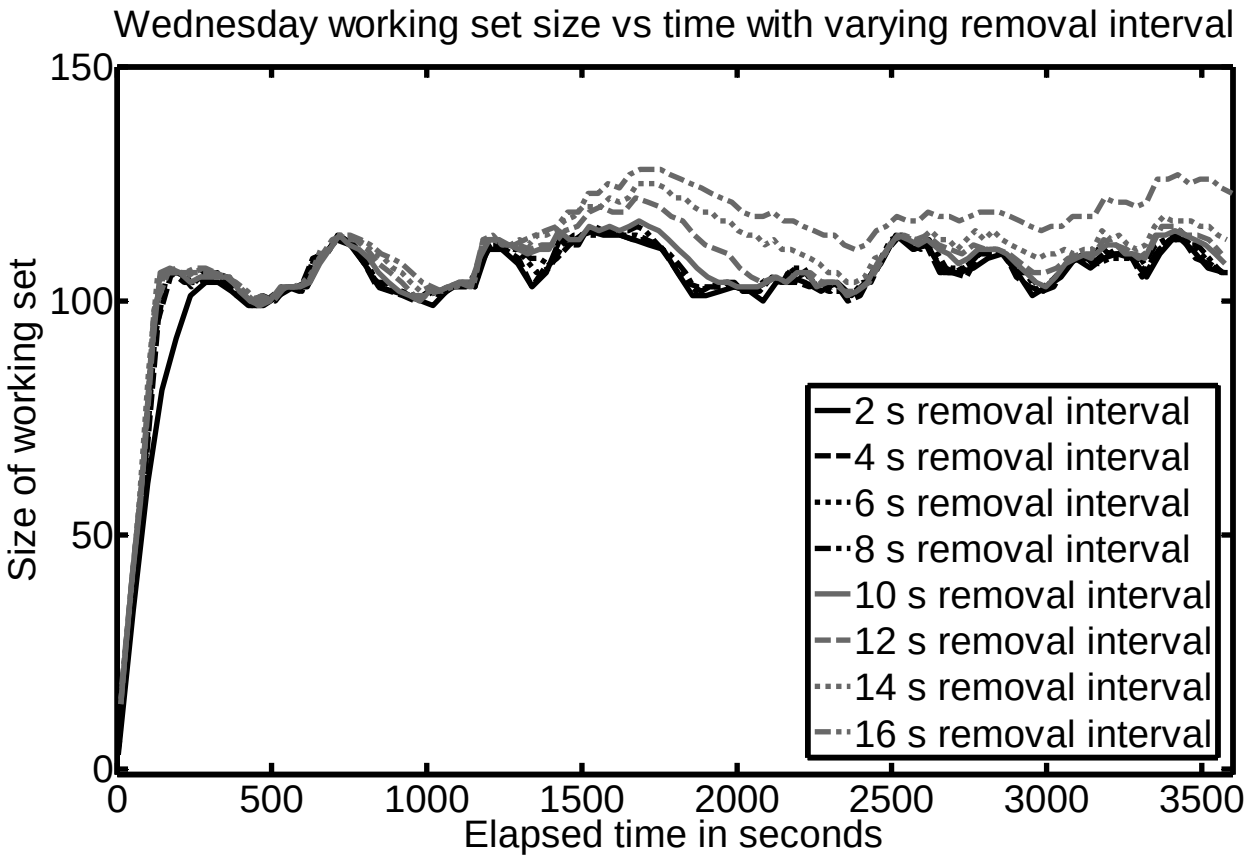

Figure 3: The size of the Wednesday connection working-set over time, given a fixed removal interval. The size does not exceed 129.

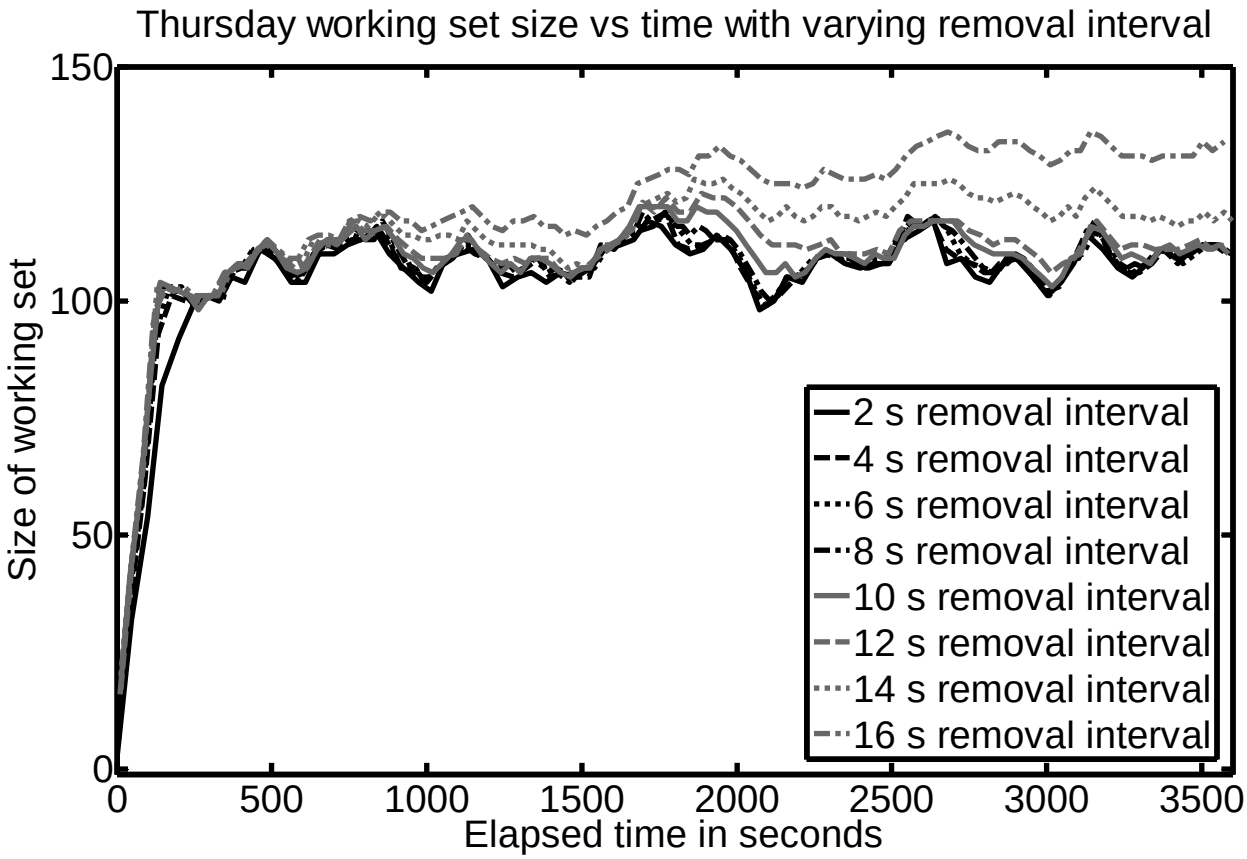

Figure 4: The size of the Thursday connection working-set over time, given a fixed removal interval. The size does not exceed 136.

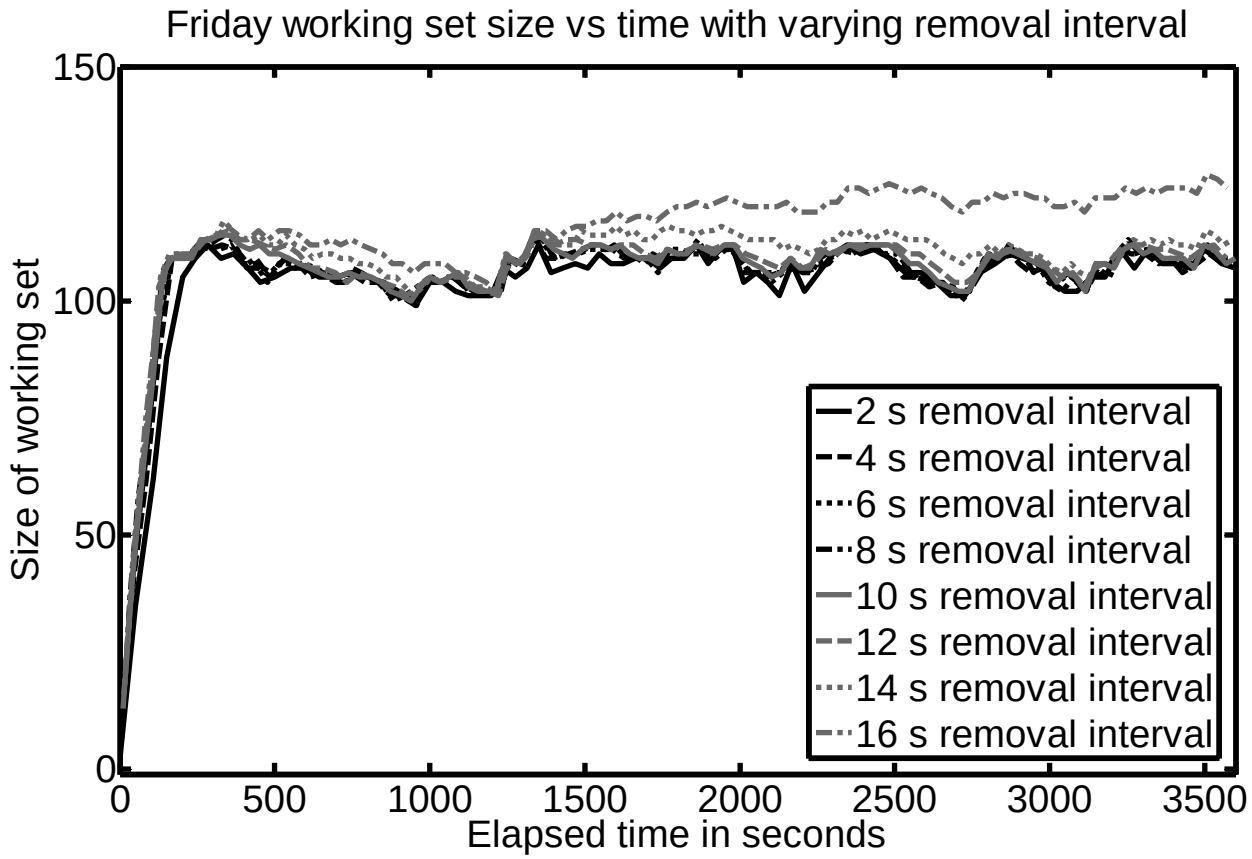

Figure 5: The size of the Friday connection working-set over time, given a fixed removal interval. The size does not exceed 127.

Several observations stand out from these data. On each of the four days, the working-set size reaches a near-constant level within about 300 seconds. There is clear consistency in the behavior of the working-sets across the four days. The working-set sizes do not exceed 125, 129, 136, and 127 on Tuesday, Wednesday, Thursday, and Friday, respectively. As the removal interval increases, the maximum size of the working-set also increases. This is expected, because connections are being removed less often from the working-set. The data support the idea that incoming connections to a network server behave in accordance with the principle of locality.

For this network server, we can choose a connection removal interval of between 2 and 16 seconds and a threshold of about 150 for violation of locality. The experimental data also indicate that the working-set size remains constant for a wide range of removal intervals (between 2 and 10 seconds), but starts to deviate when the removal interval is greater than 10 seconds. Thus, the working-set need not be updated very frequently during real-time monitoring.

These working-sets include only incoming connections to a network server. The working-sets in [1] that consisted of outgoing connections from a personal computer differed in the size threshold for violation of locality. Under normal behavior, which included mostly human-initiated activity, the working-set did not grow past a size of 6 connections when using a removal interval of 2 seconds. Here, our server working-sets do not grow past a size of about 120 connections when using a removal interval of 2 seconds. Note that we do not yet have experimental data for attacks on the server from multiple computers to test whether violation of locality would occur, although we plan to carry out such experiments in the future. However, we expect that an attack originating from multiple IP addresses, such as a distributed denial-of-service attack, would cause the size of the working-set to exceed the threshold, and we test this hypothesis in Chapter 9. The consistency of the working-set size over time assures us that normal incoming connections to the server follow a predictable pattern.

2.3.2 Port Working-set Behavior

We carried out a TCP/IP fingerprinting attack against the server using nmap [10], a network security scanner. This attack connects to more than 1000 ports on the target machine in an attempt to determine the operating system running on the machine, based on the characteristic pattern of open and closed ports exhibited by various operating systems. The attacker can then design an attack against the particular operating system that was detected. Figures 6, 7, and 8 below are semilog graphs of port working-set behavior under the fingerprinting attack, using a port removal interval of 2, 4, and 6 seconds, respectively. In addition, we also conducted a port scan attack using nmap. This attack connects to more than 1700 ports on the target machine to determine which ports are open and possibly vulnerable to attack. Figures 9, 10, and 11 below are semilog graphs of port working-set behavior under the port scan attack, using a port removal interval of 2, 4, and 6 seconds, respectively.

The figures also include the size of the connection working-set as a separate line. Recall that each connection in the connection working-set has an associated port working-set. Each figure shows the size of the largest port working-set among all the connections. The figures include a one-hour period surrounding the attack.

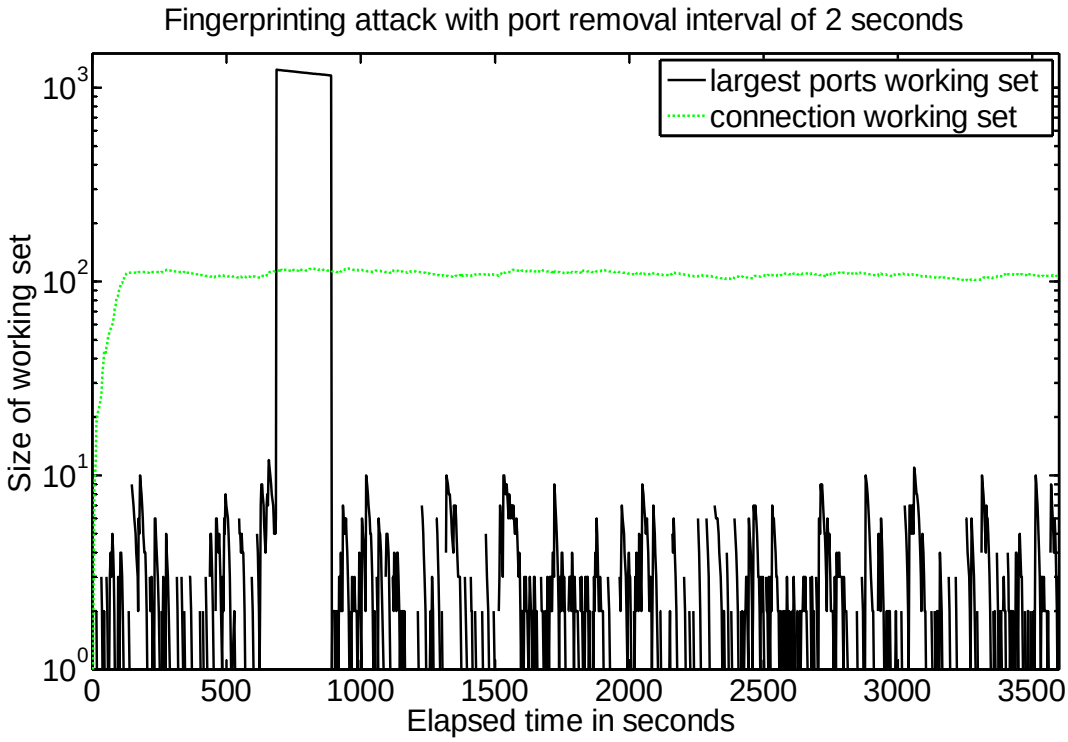

Figure 6: The size of the largest port working-set over time, given a removal interval of 2 seconds, under a fingerprinting attack. Maximum size of the largest port working-set is 12 when not under attack.

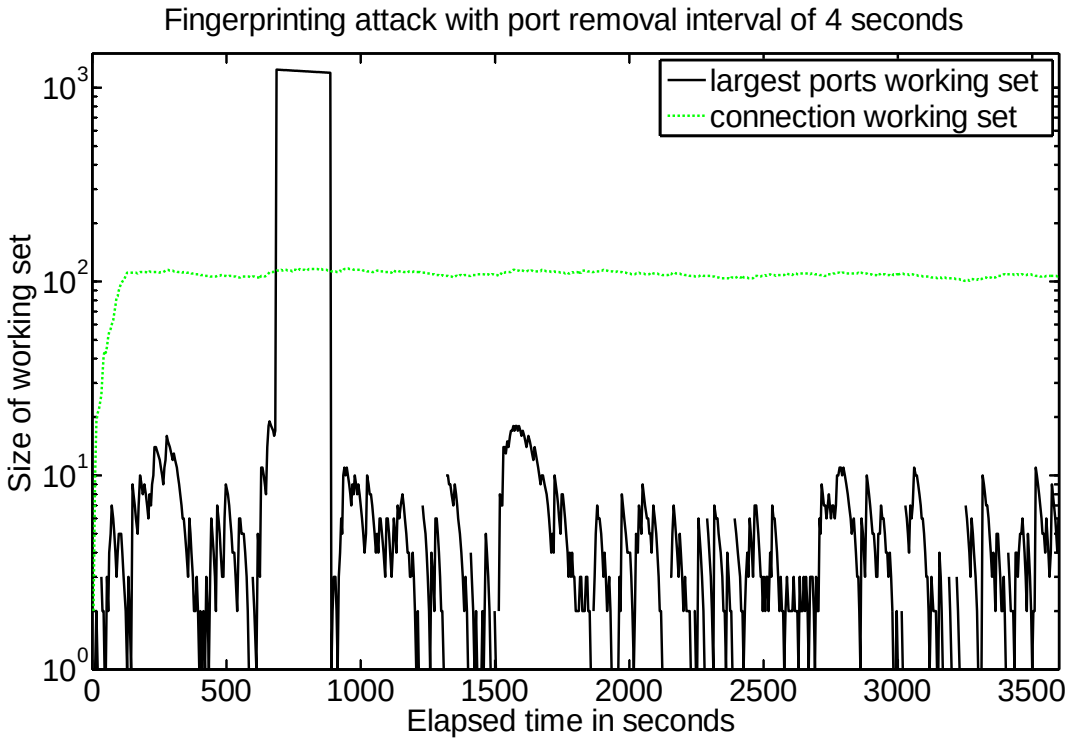

Figure 7: The size of the largest port working-set over time, given a removal interval of 4 seconds, under a fingerprinting attack. Maximum size of the largest port working-set is 19 when not under attack.

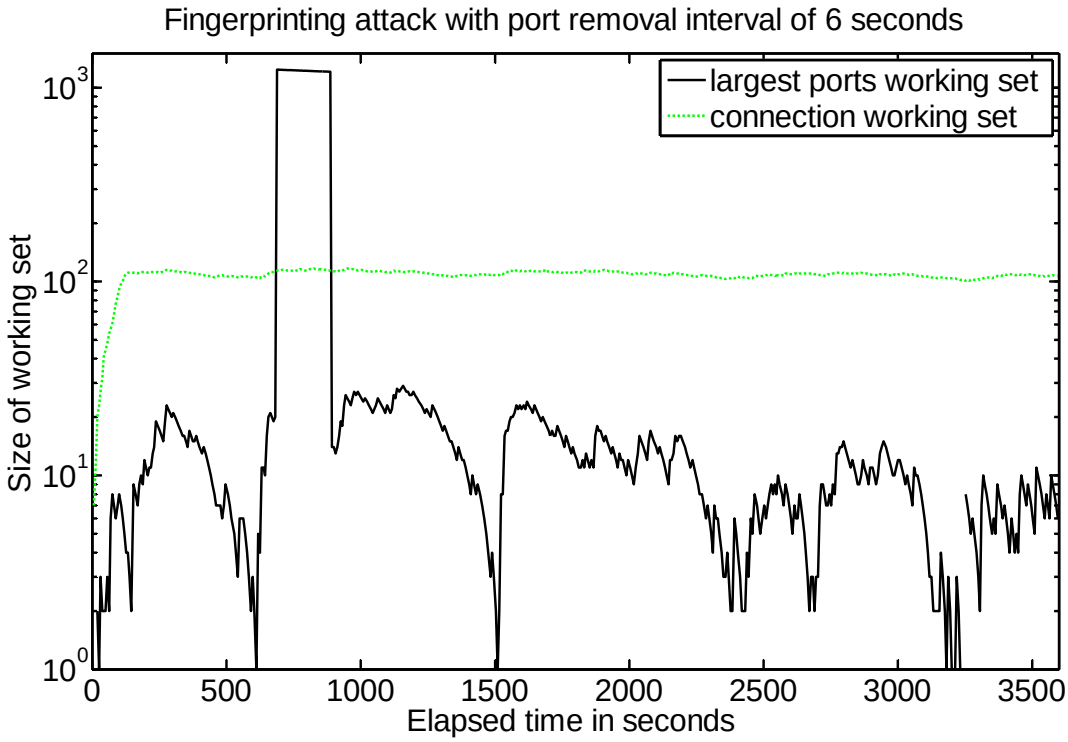

Figure 8: The size of the largest port working-set over time, given a removal interval of 6 seconds, under a fingerprinting attack. Maximum size of the largest port working-set is 29 when not under attack.

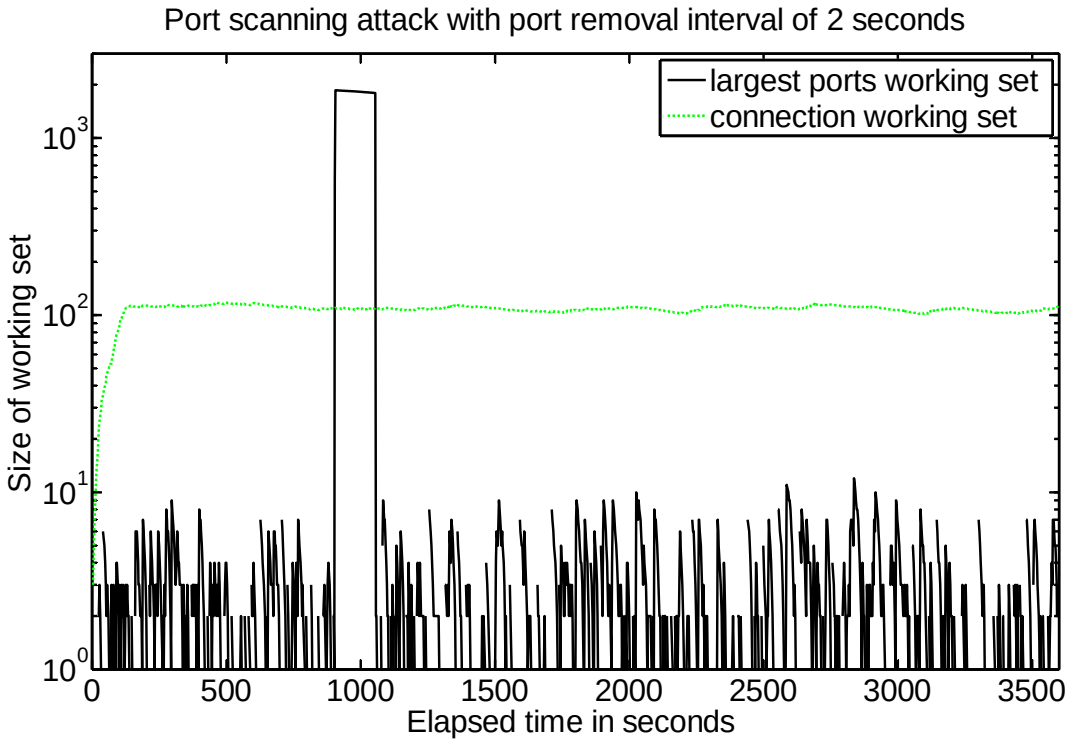

Figure 9: The size of the largest port working-set over time, given a removal interval of 2 seconds, under a port scanning attack. Maximum size of the largest port working-set is 12 when not under attack.

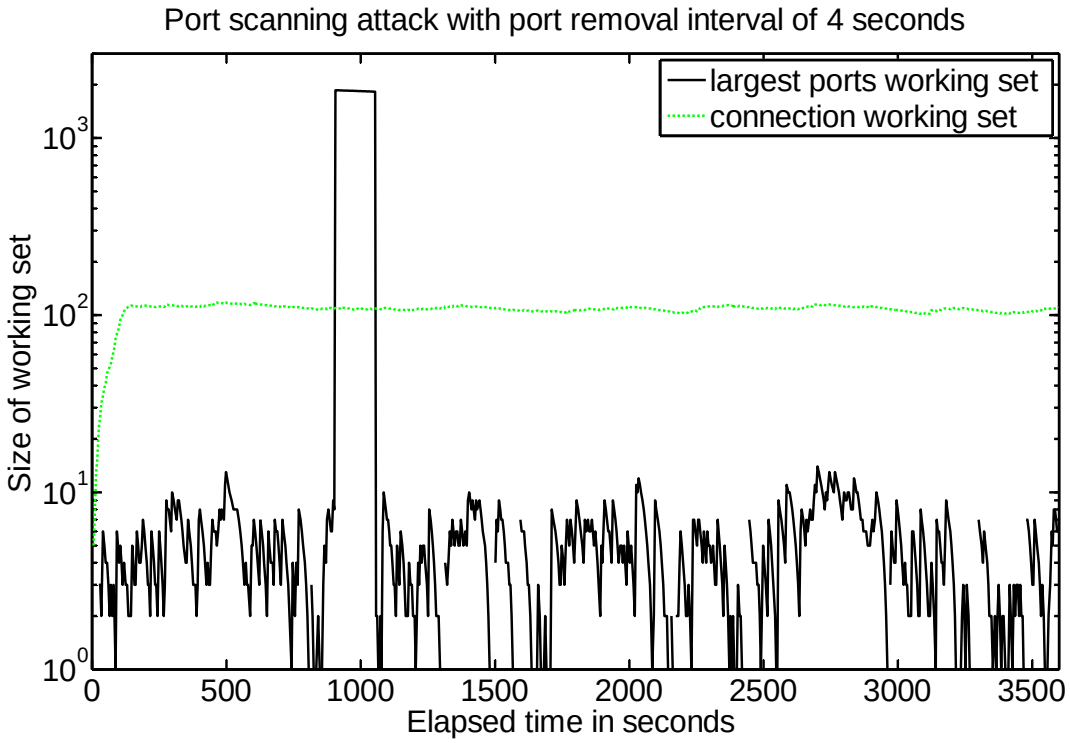

Figure 10: The size of the largest port working-set over time, given a removal interval of 2 seconds, under a port scanning attack. Maximum size of the largest port working-set is 14 when not under attack.

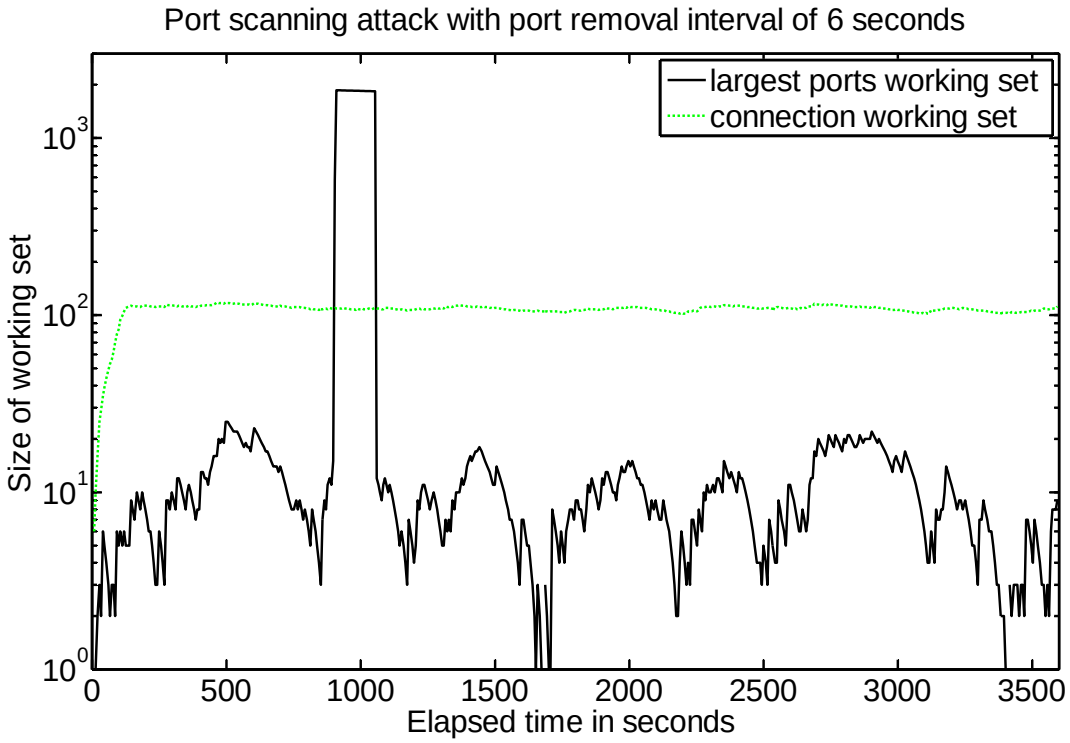

Figure 11: The size of the largest port working-set over time, given a removal interval of 2 seconds, under a port scanning attack. Maximum size of the largest port working-set is 25 when not under attack.

In Figures 6 through 8 above, the fingerprinting attack appears as a steep rise in the size of the largest port working-set to over 1000. In Figures 9 through 11 above, the port scan attack appears as a steep rise in the size of the largest port working-set to nearly 2000. As pairs of ports are removed from that largest working-set, the size gradually decreases. The sudden drop in the size of that set occurs when the connection that performed the fingerprinting attack is removed from the connection working-set. When there is no attack present, the size of the largest port working-set never exceeds a certain level. For the fingerprinting attack, with a removal interval of 2, 4, and 6 seconds, the maximum size is 12, 19 and 29, respectively. For the port scan attack, with a removal interval of 2, 4, and 6 seconds, the maximum size is 12, 14 and 25, respectively. The

data support the thesis that the ports used by incoming connections behave in accordance with the principle of locality. For this network server, we can choose a port removal interval of between 2 and 6 seconds and a threshold of about 35 for violation of locality.

Note that the connection working-set size, represented by a dotted line in the six attack data figures above, remains nearly constant at about 100. This is consistent with our earlier observations regarding connection working-set behavior.

2.4 Conclusion

Our experimental results showed that connections to a typical network server and their associated ports exhibit locality. We demonstrated that this allows us to generate a normal profile for the network server. By conducting attacks on the server, we confirmed that violation of locality occurred in a port working-set, allowing us to detect the attacks.

A strength of our locality-based intrusion detection method is its simplicity of implementation in real-time. One type of attack for which signature-based detection complements our method is "sneaky" scans, that is, scans that connect at a rate too slow to affect the working-set in a detectable manner.

CHAPTER 3: FIFO REMOVAL METHOD

3.1 Introduction

In Chapter 2, the replacement method used to detect locality was LRU (least recently used). When the LRU method is utilized, at each removal interval of 2 to 16 seconds, the connection with the least recent access time is removed from the working-set. Note that the most recent access time for each connection in the working-set is stored. If a connection is already in the working-set and another packet is received from that connection, the most recent access time for that connection is updated. Thus, the connection with the least recent access time is not necessarily the connection that was first placed into the working-set. On the other hand, when the FIFO (first in, first out) removal method is utilized, the connection that was first placed into the working-set is the one removed.

In Section 3.2, we will compare the execution time of the LRU method with the FIFO method given the same four sets of input data: the packets collected from 1300-1400 on Tuesday, Wednesday, Thursday, and Friday (December 6-9, 2006). In Section 3.3, we will examine how well the FIFO method preserves locality, as compared with LRU.

3.2 Execution Time Using LRU Versus FIFO

We processed the traffic data for each of the four days using the LRU removal method and the FIFO removal method, with 10 runs for each day. All runs were performed on an Intel Core 2 processor operating at 2.33 GHz. Table 2 below shows the execution times with mean and standard deviation.

Table 2. LRU versus FIFO execution times on 1300-1400 data.

FIFO	Tue	Wed	Thu	Fri
	1.453	1.312	1.421	1.64
	1.437	1.296	1.406	1.671
	1.484	1.281	1.359	1.64
	1.515	1.281	1.406	1.64
	1.453	1.312	1.406	1.64
	1.421	1.265	1.437	1.64
	1.468	1.281	1.39	1.671
	1.453	1.312	1.39	1.625
	1.453	1.265	1.359	1.64
	1.421	1.265	1.421	1.64
Mean	1.4558	1.287	1.3995	1.6447
Stdev	0.026936	0.018751	0.024328	0.013878
LRU	Tue	Wed	Thu	Fri
1	1.593	1.359	1.5	1.75
2	1.531	1.343	1.453	1.734
3	1.578	1.375	1.453	1.703
4	1.625	1.343	1.453	1.734
5	1.546	1.359	1.468	1.734
6	1.5	1.375	1.453	1.703
7	1.546	1.343	1.453	1.734
8	1.515	1.343	1.453	1.718
9	1.546	1.375	1.484	1.734
10	1.515	1.343	1.453	1.703
Mean	1.5495	1.3558	1.4623	1.7247
Stdev	0.036898	0.013948	0.015906	0.015906
Packets:	58988	52501	55805	69829
Difference between LRU and FIFO mean execution times:				
	Tue	Wed	Thu	Fri
	0.0937	0.0688	0.0628	0.08
	6.44%	5.35%	4.49%	4.86%

Note that the standard deviation for the runs was low, so we are confident that the execution times were precisely measured. The LRU removal method requires only about 5.3% more time (on average) to execute than the FIFO removal method. Both of these algorithms used a binary min-heap. The only difference in the FIFO algorithm is that, when a connection already exists in the working-set, its most recent access time is not updated, so a sift-down is not required. We expected FIFO to be faster, because it is a slightly simpler method.

3.3 Locality Preservation Using LRU Versus FIFO

Figures 12 and 13 below show the Tuesday 1300-1400 data processed using LRU and FIFO.

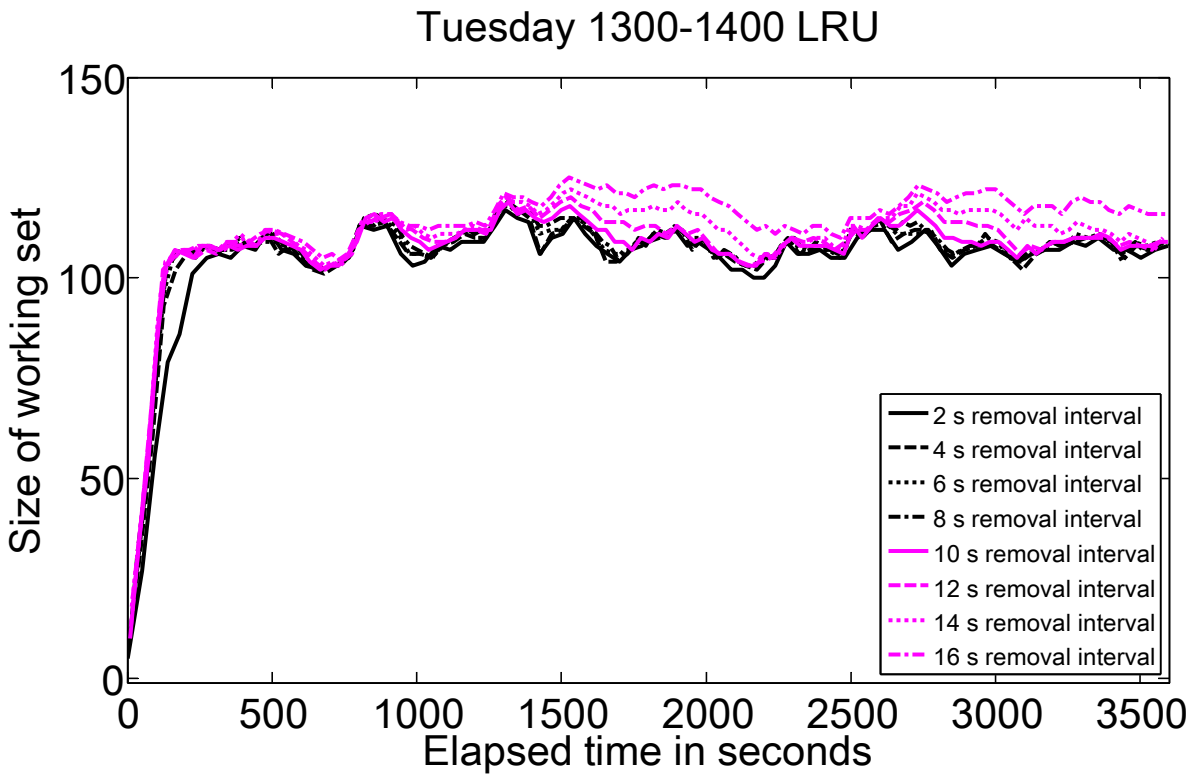

Figure 12: Tuesday 1300-1400 data processed using LRU.

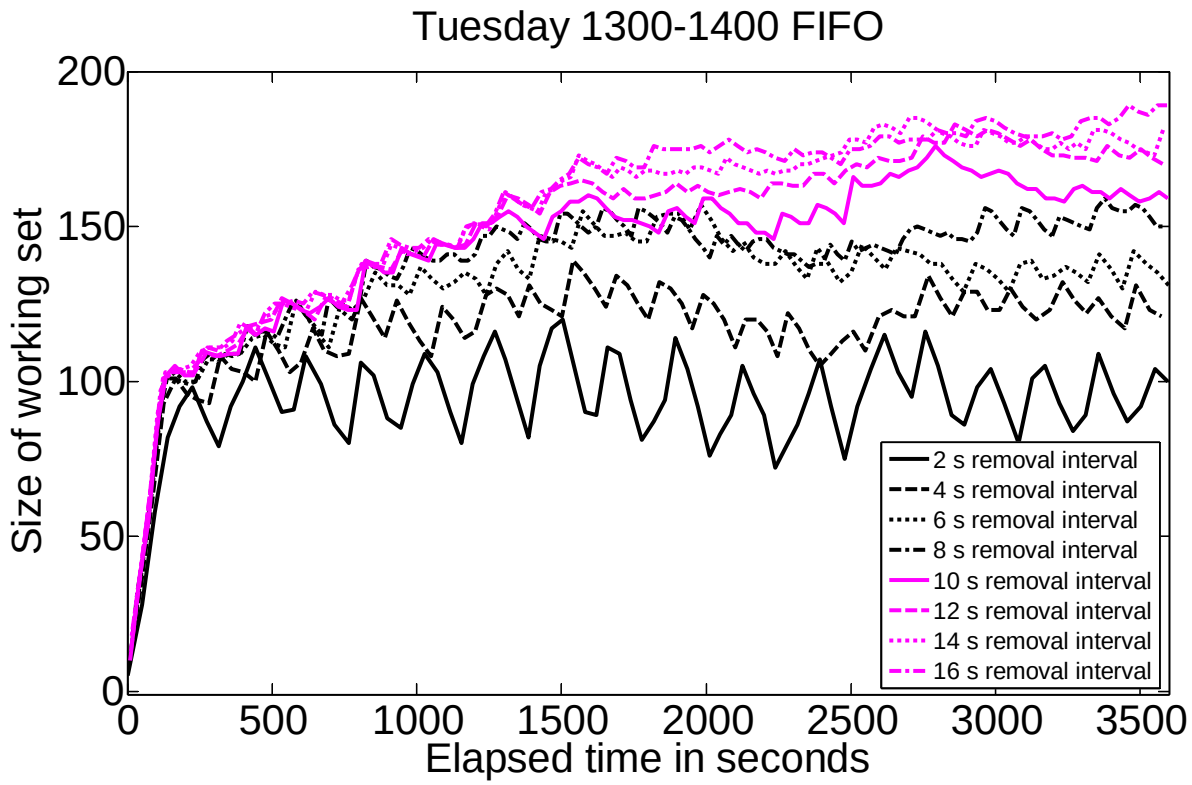

Figure 13: Tuesday 1300-1400 data processed using FIFO.

The shape of the FIFO curves is very different from the shape of the LRU curves. While the LRU curves all stabilize within about 300 seconds to a small range of working-set sizes (about 100 to 125), the FIFO curves do not stabilize quickly, if at all. It appears that FIFO is not as suitable as LRU for locality preservation.

Figures 14 and 15 below show the Wednesday 1300-1400 data processed using LRU and FIFO.

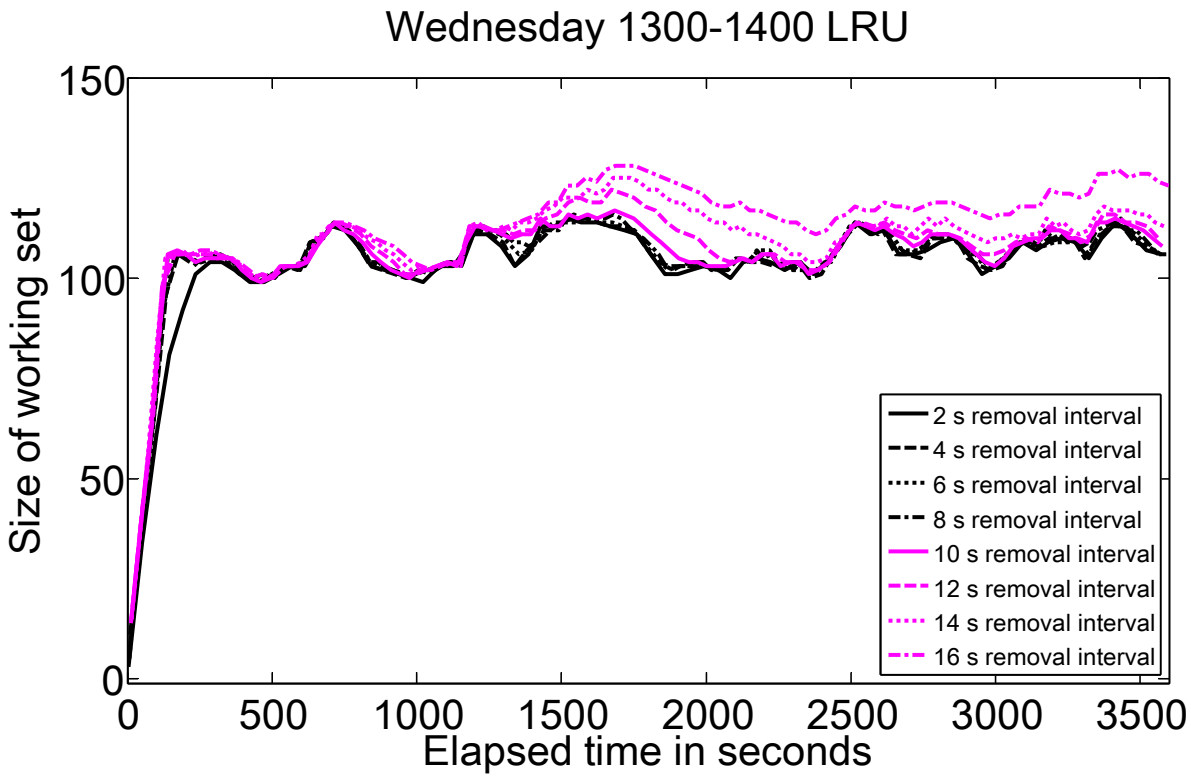

Figure 14: Wednesday 1300-1400 data processed using LRU.

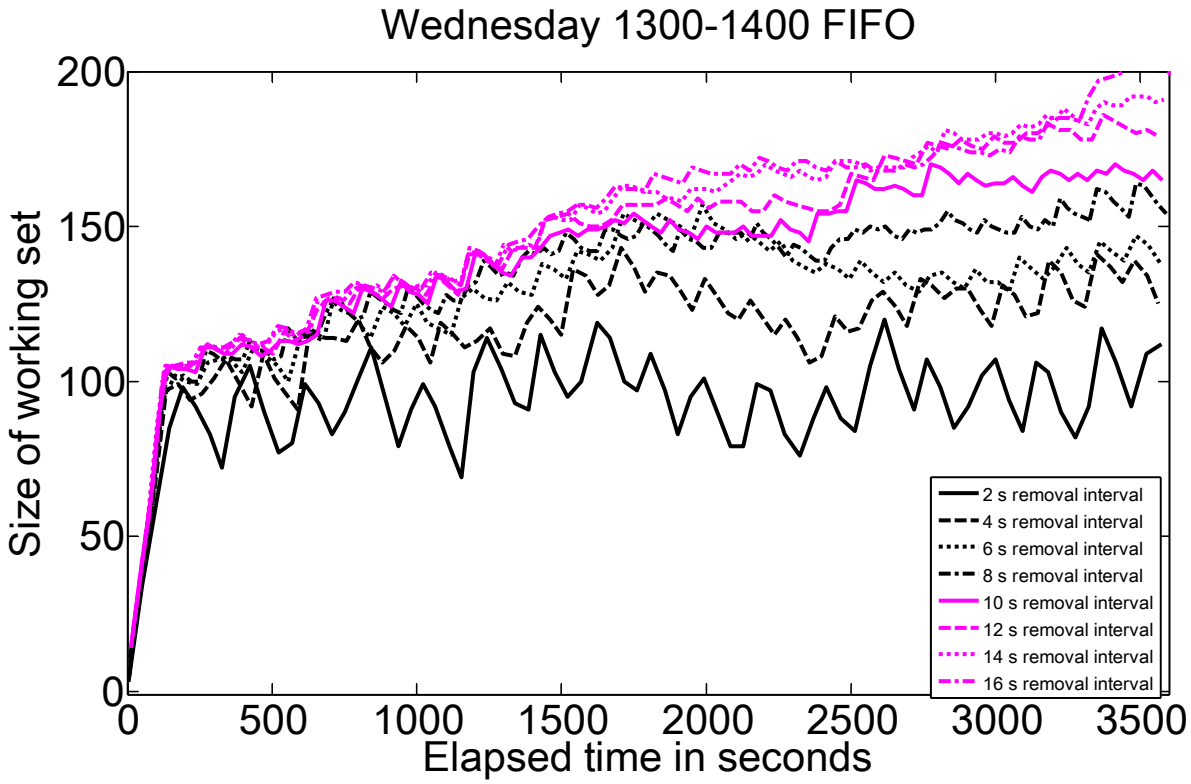

Figure 15: Wednesday 1300-1400 data processed using FIFO.

Figures 16 and 17 below show the Thursday 1300-1400 data processed using LRU and FIFO.

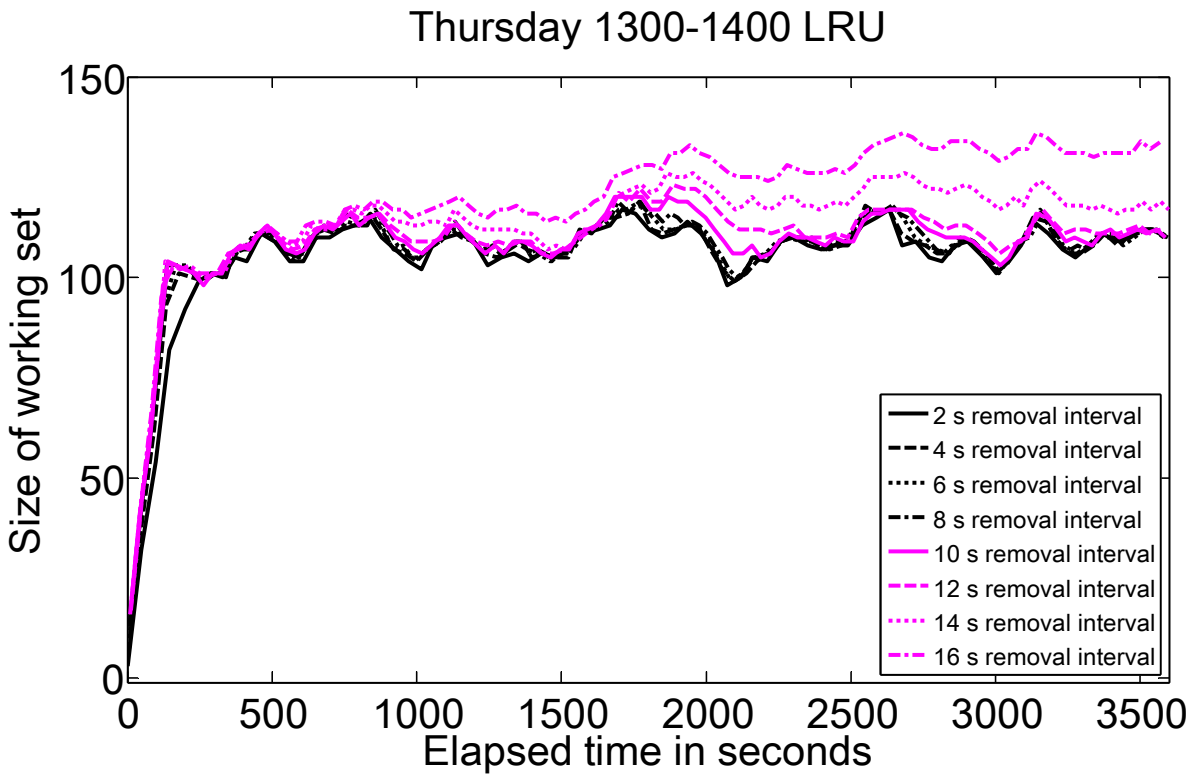

Figure 16: Thursday 1300-1400 data processed using LRU.

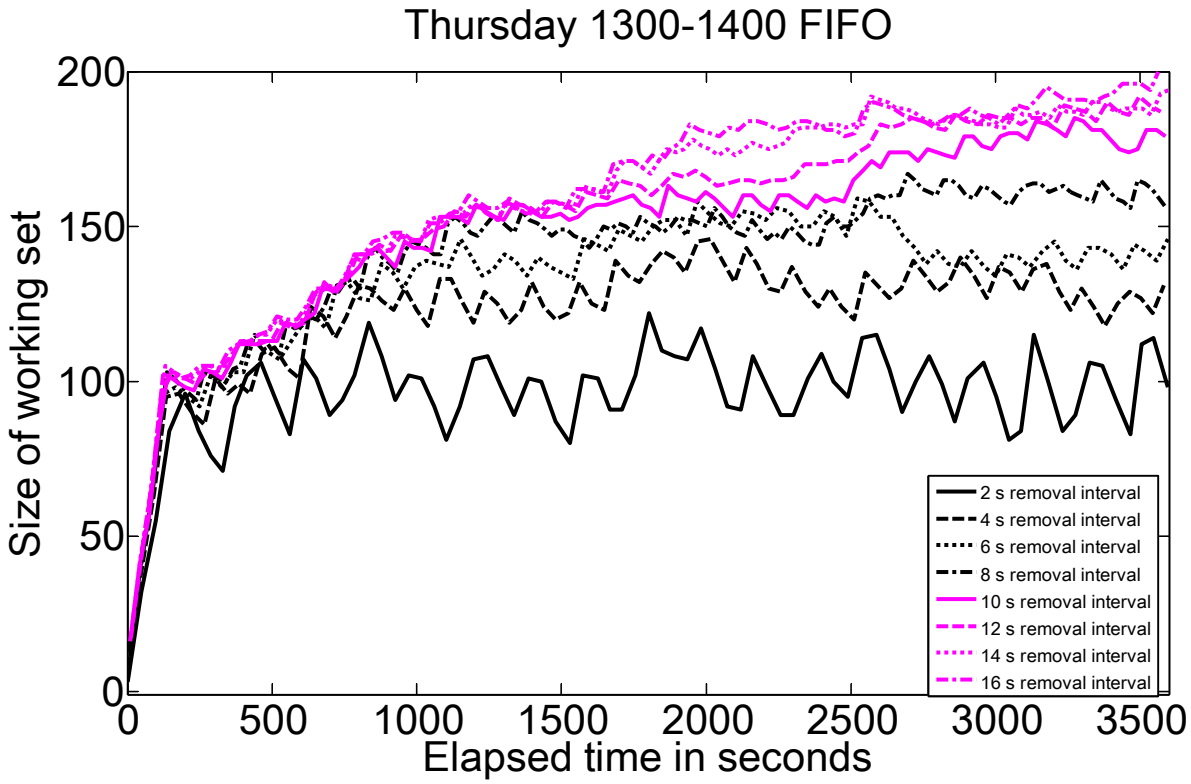

Figure 17: Thursday 1300-1400 data processed using FIFO.

Figures 18 and 19 below show the Friday 1300-1400 data processed using LRU and FIFO.

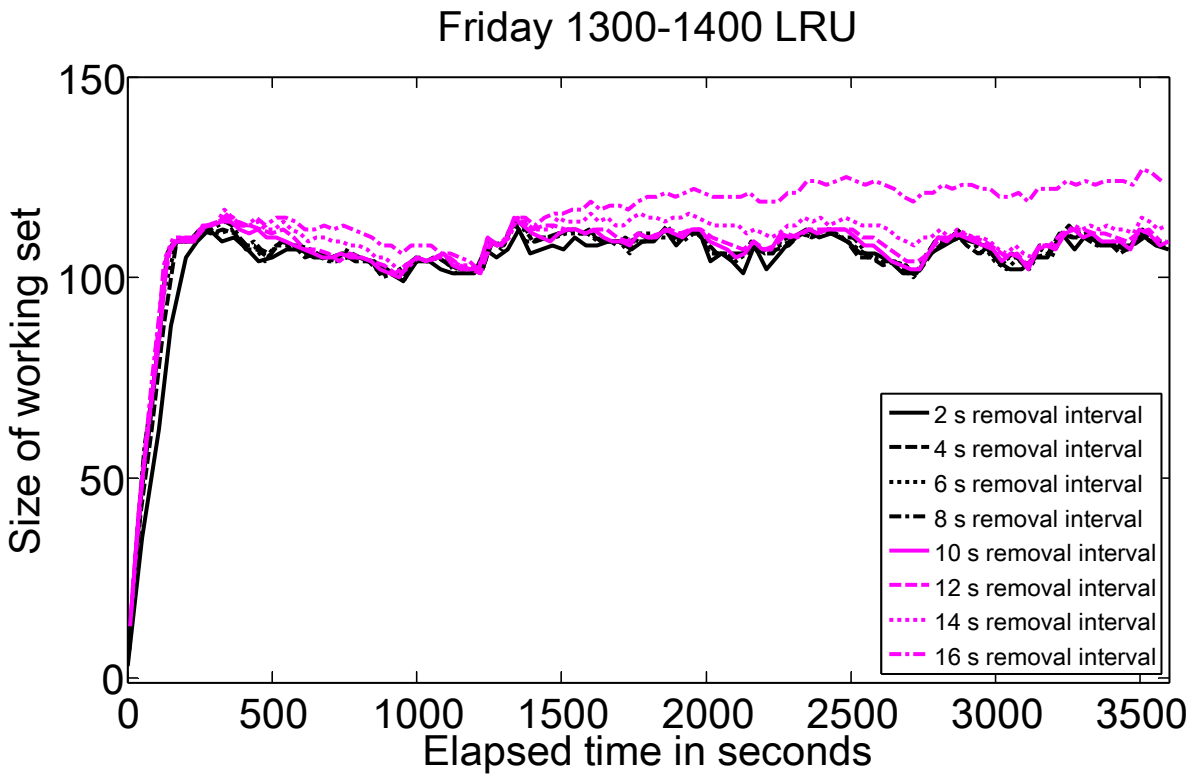

Figure 18: Friday 1300-1400 data processed using LRU.

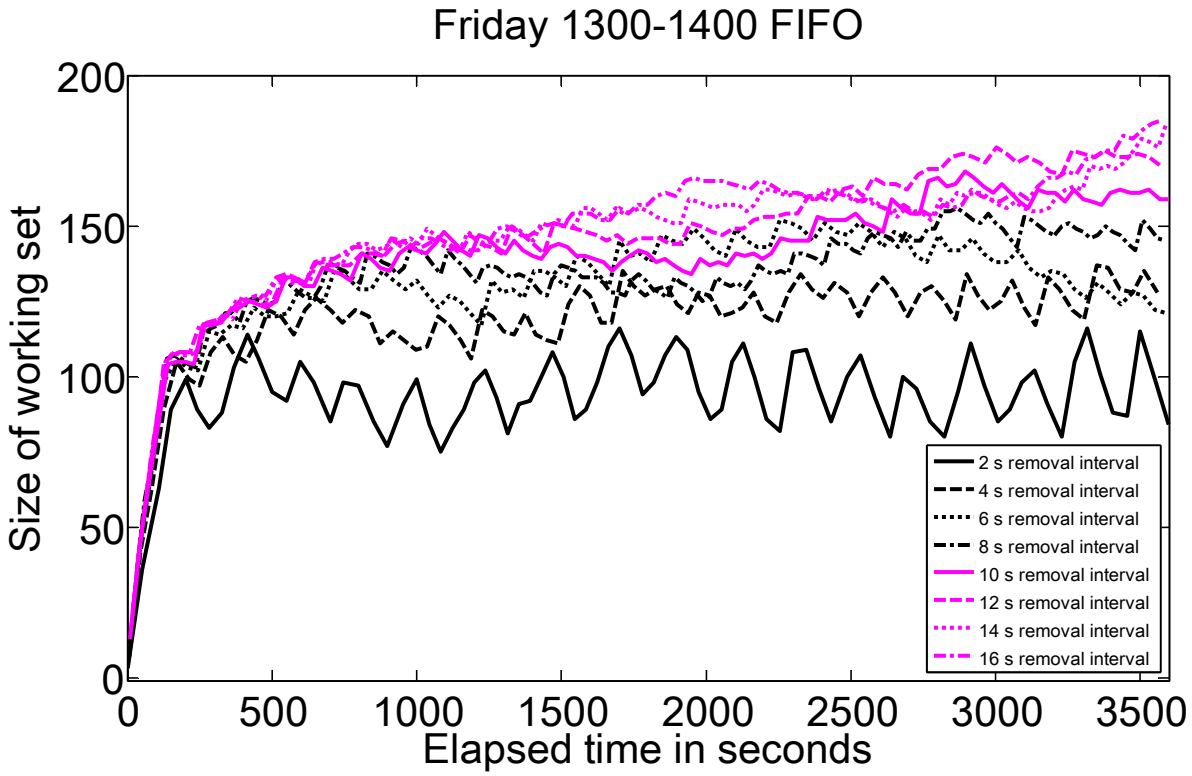

Figure 19: Friday 1300-1400 data processed using FIFO.

Figures 14-15 (Wednesday), 16-17 (Thursday), and 18-19 (Friday) show a difference between the LRU and FIFO results similar to the difference observed in Figures 12-13 (Tuesday). They further support the idea that FIFO is not as suitable as LRU in preserving locality.

CHAPTER 4: LFU AND PURE LFU REMOVAL METHODS

4.1 Introduction

We have already discussed the LRU (least recently used) and FIFO (first in, first out) removal methods. In this chapter, we consider the LFU (least frequently used) removal method and the closely related Pure LFU removal method. In LFU, the connection removed from the working-set is the one with the smallest number of accesses since it entered the working-set. In Pure LFU, the connection removed from the working-set is the one with the smallest number of accesses since the period of observation began, and the access count is not reset to zero when a connection is removed from the working-set. Here is an example to help make the distinction clear. Suppose that, during the past 30 minutes, connections 1, 2, 3, 4, and 5 have been present in the working-set at one time or another. At the moment, however, connections 2 and 5 are the only connections present in the working-set. If we are to decide using LFU which connection to remove from the working-set, we compare the number of accesses of connections 2 and 5 since their most recent entry into the working-set. On the other hand, if we are to decide using Pure LFU which connection to remove from the working-set, we compare the number of accesses of connections 2 and 5 since the period of observation began 30 minutes ago.

4.2 LFU Working-set Performance

In Figures 20, 21, 22, and 23 below are graphs of LFU working-set size over time for Tuesday, Wednesday, Thursday, and Friday, respectively, all for the data from 1300-1400.

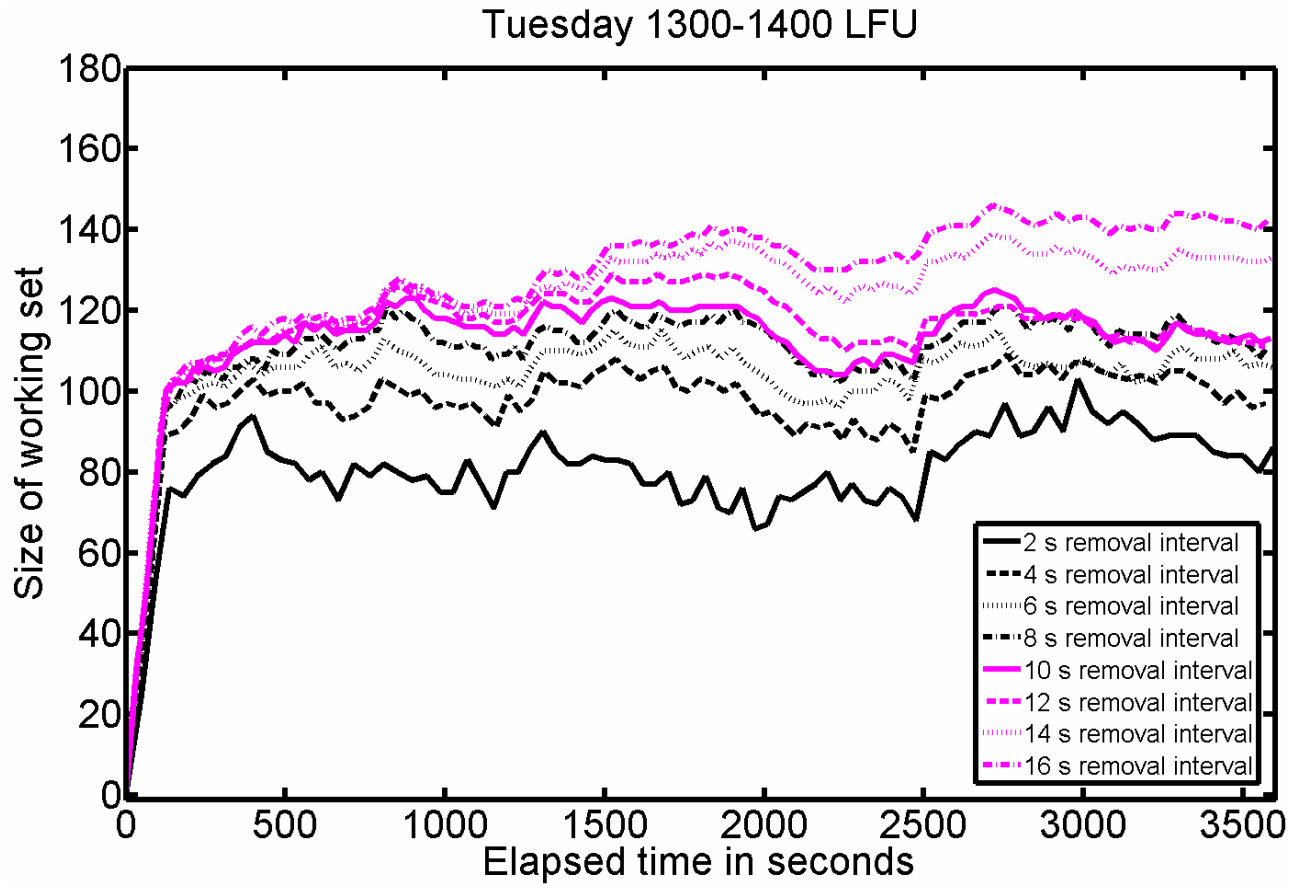

Figure 20: LFU working-set size over time for Tuesday 1300-1400.

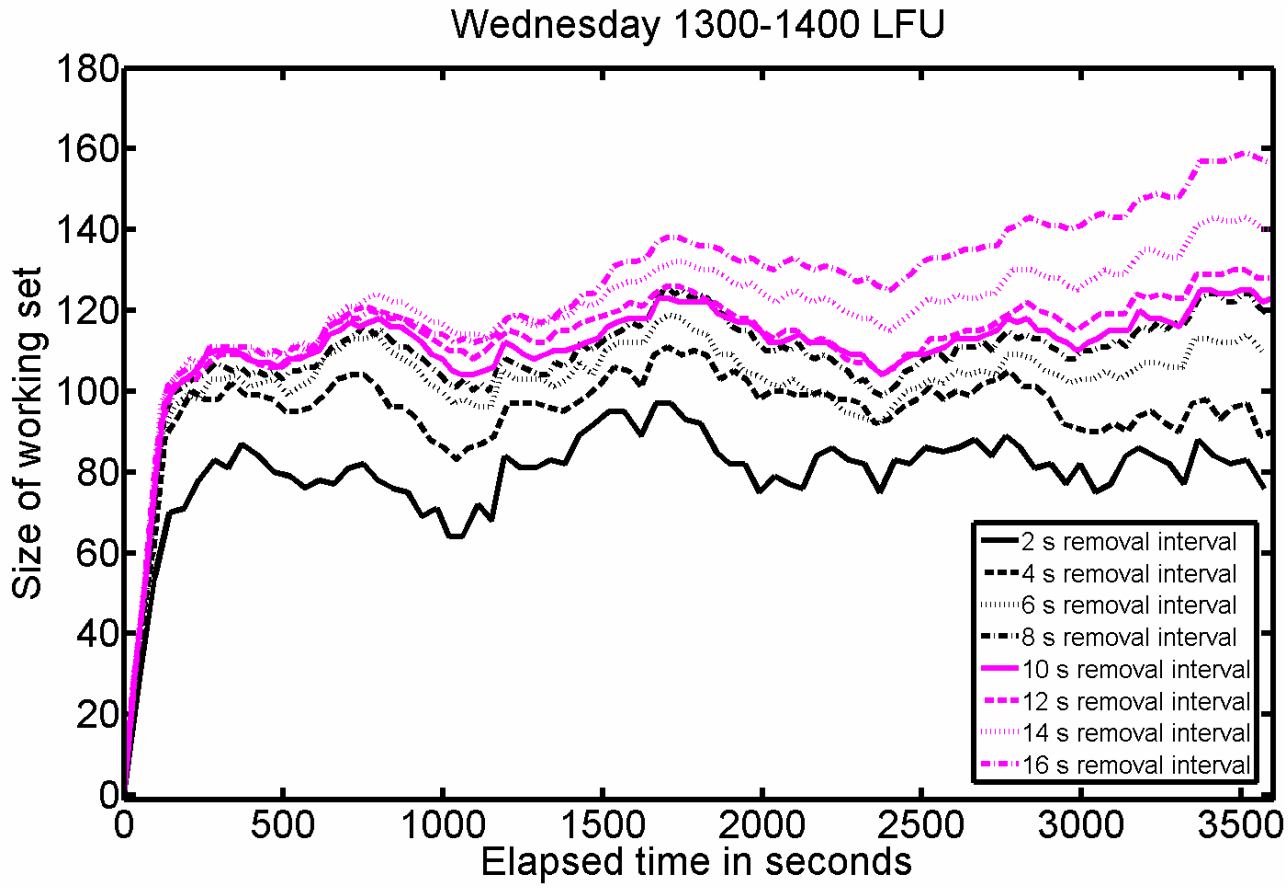

Figure 21: LFU working-set size over time for Wednesday 1300-1400.

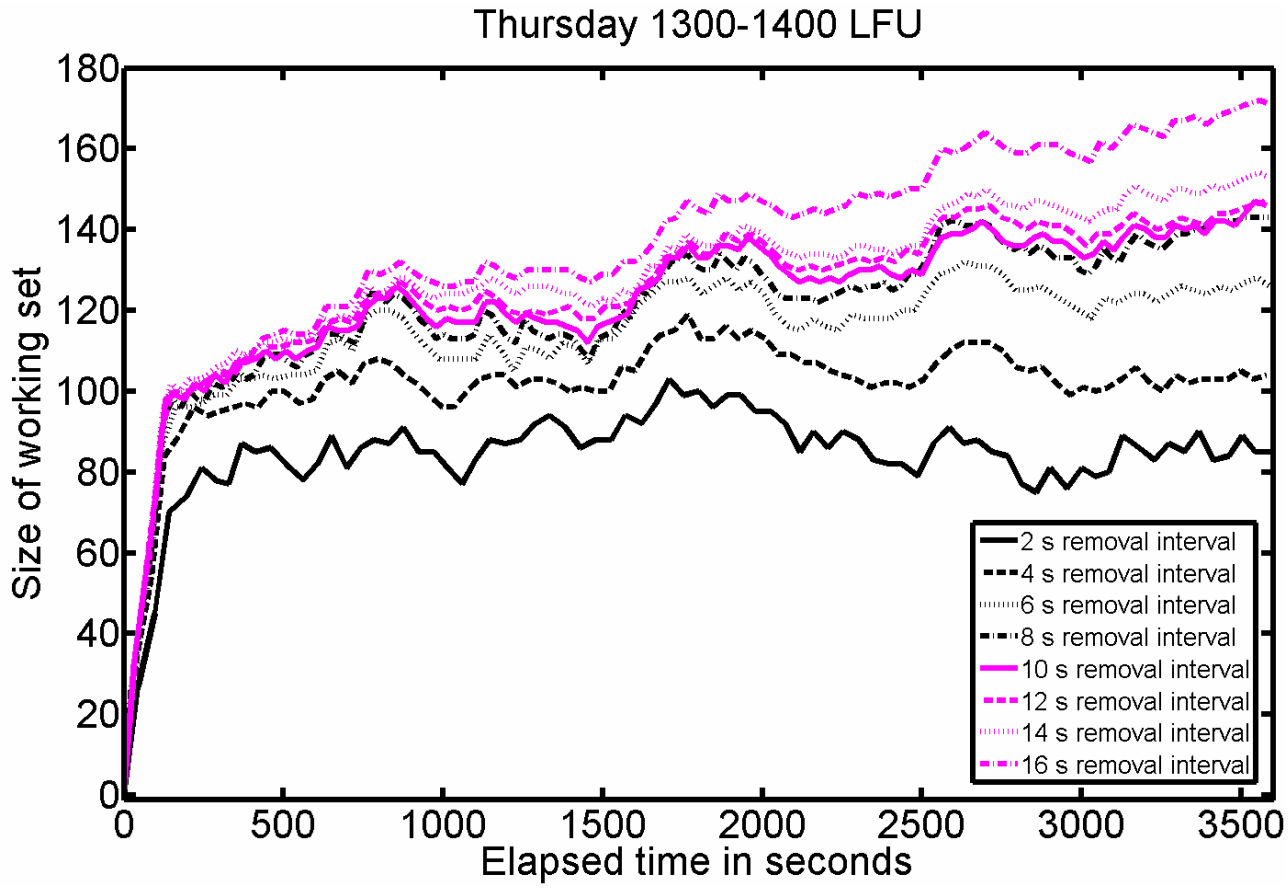

Figure 22: LFU working-set size over time for Thursday 1300-1400.

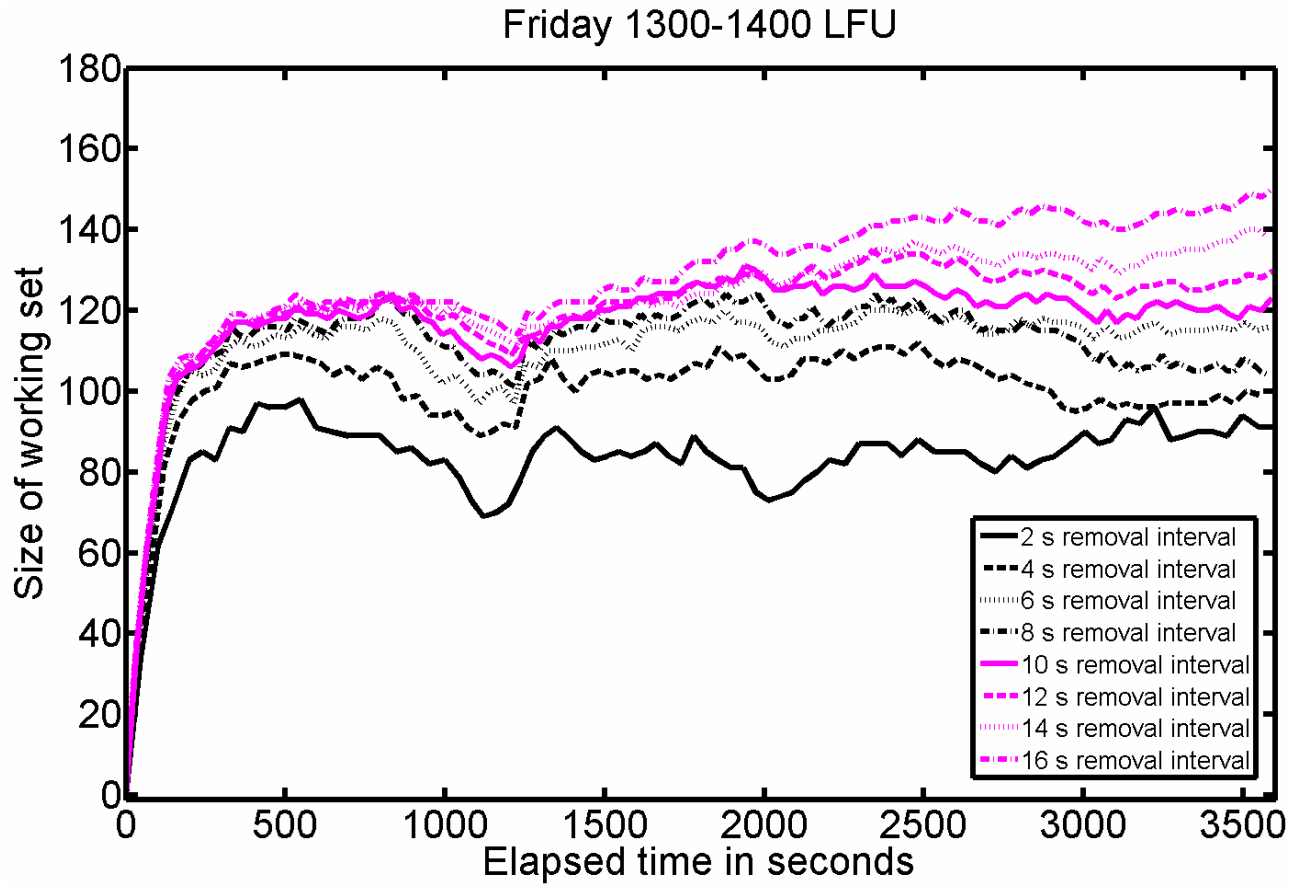

Figure 23: LFU working-set size over time for Friday 1300-1400.

From the graphs, it is apparent that LFU does not perform as well as LRU. For a quantitative comparison, in Table 3 below are the average and standard deviation of the working-set sizes for LFU (in the middle row), along with the corresponding values for LRU and FIFO.

Table 3: Average and standard deviation of working-set sizes for FIFO, LFU, and LRU.

TUE 1300-1400 FIFO			WED 1300-1400 FIFO			THU 1300-1400 FIFO			FRI 1300-1400 FIFO		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	94.1354	15.1837	2	93.40972	15.26625	2	95.97724	15.33164	2	94.08328	13.90973
4	116.6965	16.7281	4	117.2631	17.74021	4	122.9416	19.11664	4	119.0222	16.13868
6	131.2619	20.0632	6	128.189	21.27813	6	134.2066	22.3873	6	131.1357	18.77506
8	137.4696	21.7033	8	135.1211	22.89263	8	141.9123	24.90416	8	133.7687	18.49262
10	144.6532	25.1202	10	141.2679	25.58505	10	150.173	29.00525	10	140.7464	21.18444
12	151.1276	28.1948	12	147.2702	29.4063	12	155.0521	31.48083	12	147.3576	23.87658
14	154.784	29.0748	14	150.8496	30.97163	14	158.5224	32.47499	14	147.1365	22.21539
16	156.8721	30.4883	16	152.2791	31.46465	16	160.4306	33.47076	16	149.0917	23.09873

TUE 1300-1400 LFU			WED 1300-1400 LFU			THU 1300-1400 LFU			FRI 1300-1400 LFU		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	80.03828	11.5594	2	79.92029	11.14617	2	84.5231	12.18657	2	83.50947	10.52625
4	97.15159	12.236	4	95.73469	11.54653	4	101.8289	13.13805	4	100.2901	12.44935
6	104.1555	12.6409	6	102.4579	13.47794	6	114.6442	16.64626	6	110.8538	13.71508
8	110.1655	13.809	8	108.4493	14.60779	8	121.2482	19.3787	8	111.986	13.93708
10	112.853	14.6971	10	111.0178	14.2978	10	123.4503	19.31719	10	117.5291	14.86584
12	115.3058	15.2925	12	113.3462	15.42484	12	126.0174	20.13101	12	120.7612	15.74368
14	123.6853	17.431	14	120.2186	17.47435	14	129.1057	21.35351	14	122.976	16.83019
16	128.1773	19.7097	16	125.9861	21.37095	16	137.424	26.33469	16	128.484	19.29472

TUE 1300-1400 LRU			WED 1300-1400 LRU			THU 1300-1400 LRU			FRI 1300-1400 LRU		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	104.8296	13.6099	2	103.285	13.61165	2	105.3009	14.12482	2	104.2106	12.45154
4	106.2313	12.4991	4	104.6833	11.73404	4	106.514	12.57479	4	105.2126	11.76718
6	106.7788	11.9583	6	104.8936	12.46381	6	106.9543	12.29957	6	105.7036	11.22774
8	107.3388	11.6113	8	105.4237	12.08093	8	107.5095	12.01021	8	106.0678	10.7572
10	107.841	11.8663	10	106.3185	11.75404	10	108.5777	11.70998	10	106.6297	10.73969
12	109.0034	11.9261	12	107.6035	12.30293	12	109.9931	12.07931	12	107.3854	10.58042
14	110.92	11.8055	14	109.687	12.55118	14	114.151	12.98813	14	109.2731	10.74708
16	113.7945	12.6595	16	113.4419	13.75382	16	120.588	15.49437	16	115.445	12.34627

The LFU standard deviations are mostly higher than those for LRU, and the average values are not as consistent as those for LRU. However, LFU does perform better than FIFO.

4.3 Pure LFU Working-set Performance

In Figures 24, 25, 26, and 27 below are graphs of Pure LFU working-set size over time for Tuesday, Wednesday, Thursday, and Friday, respectively, all for the data from 1300-1400.

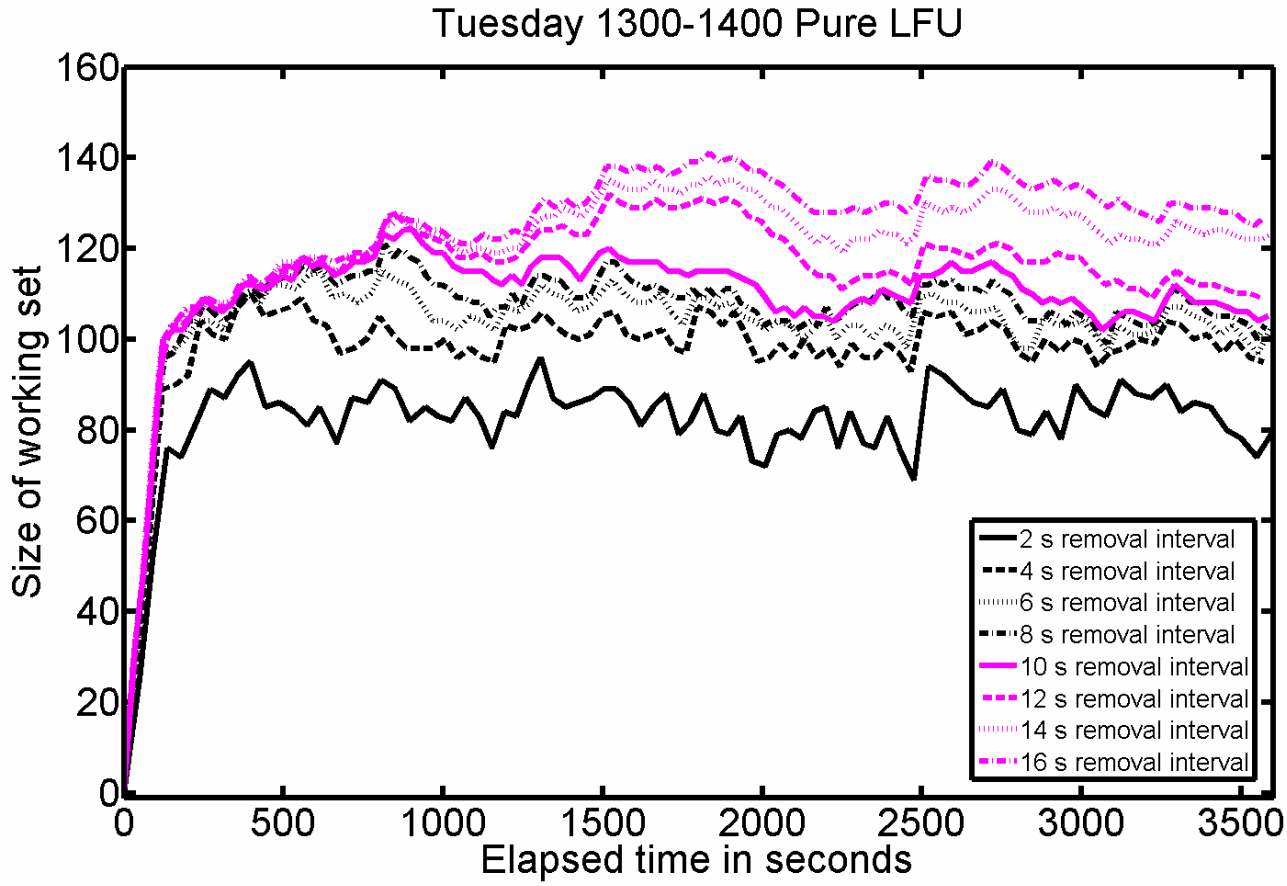

Figure 24: Pure LFU working-set size over time for Tuesday 1300-1400.

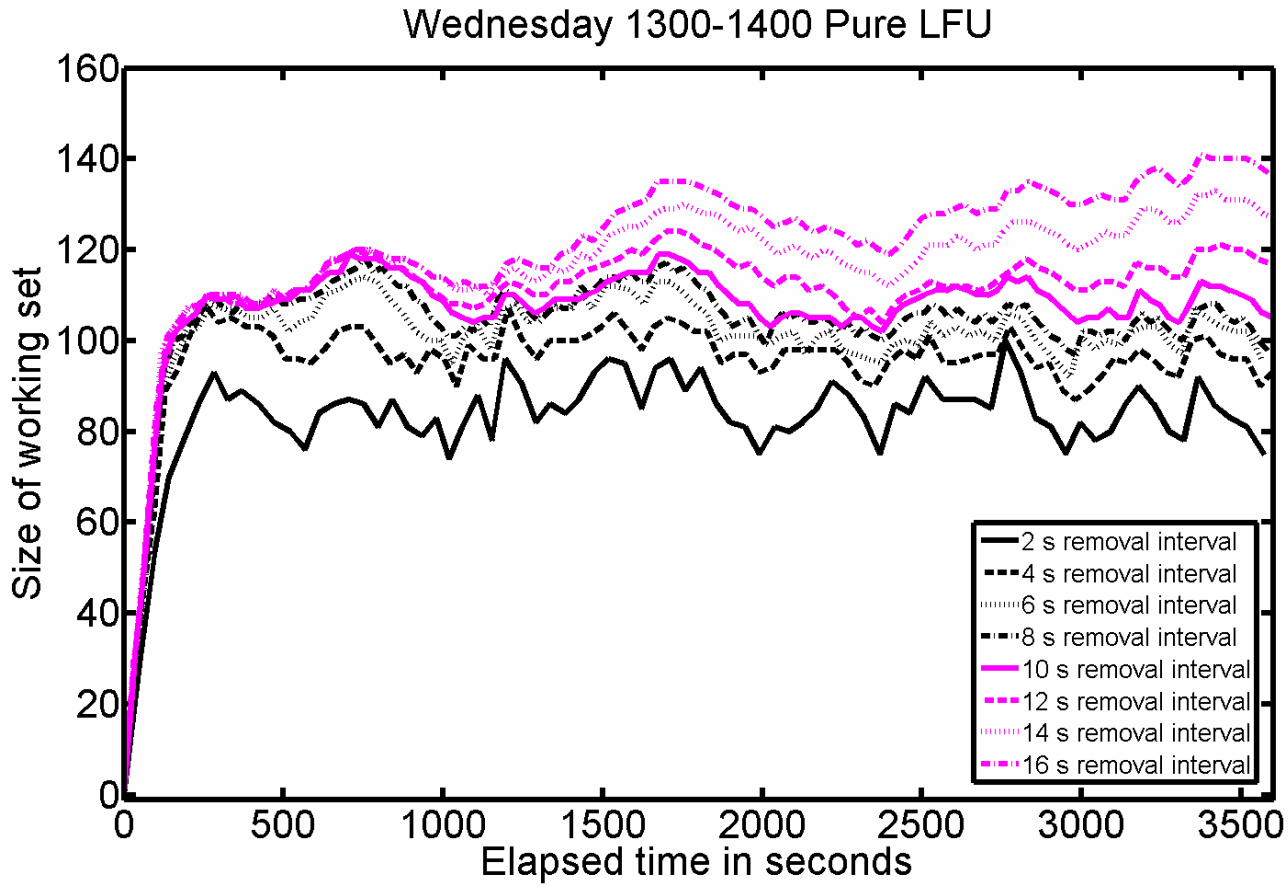

Figure 25: Pure LFU working-set size over time for Wednesday 1300-1400.

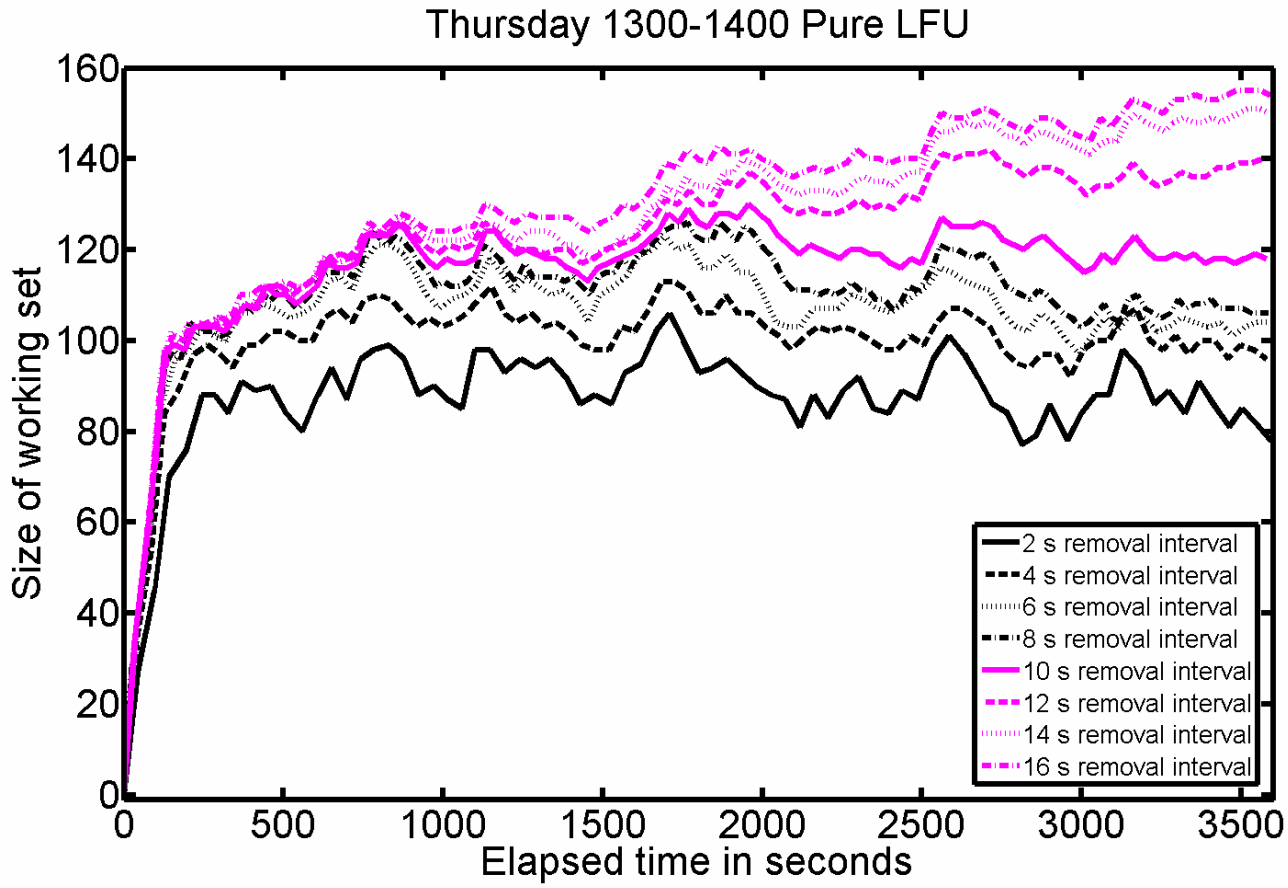

Figure 26: Pure LFU working-set size over time for Thursday 1300-1400.

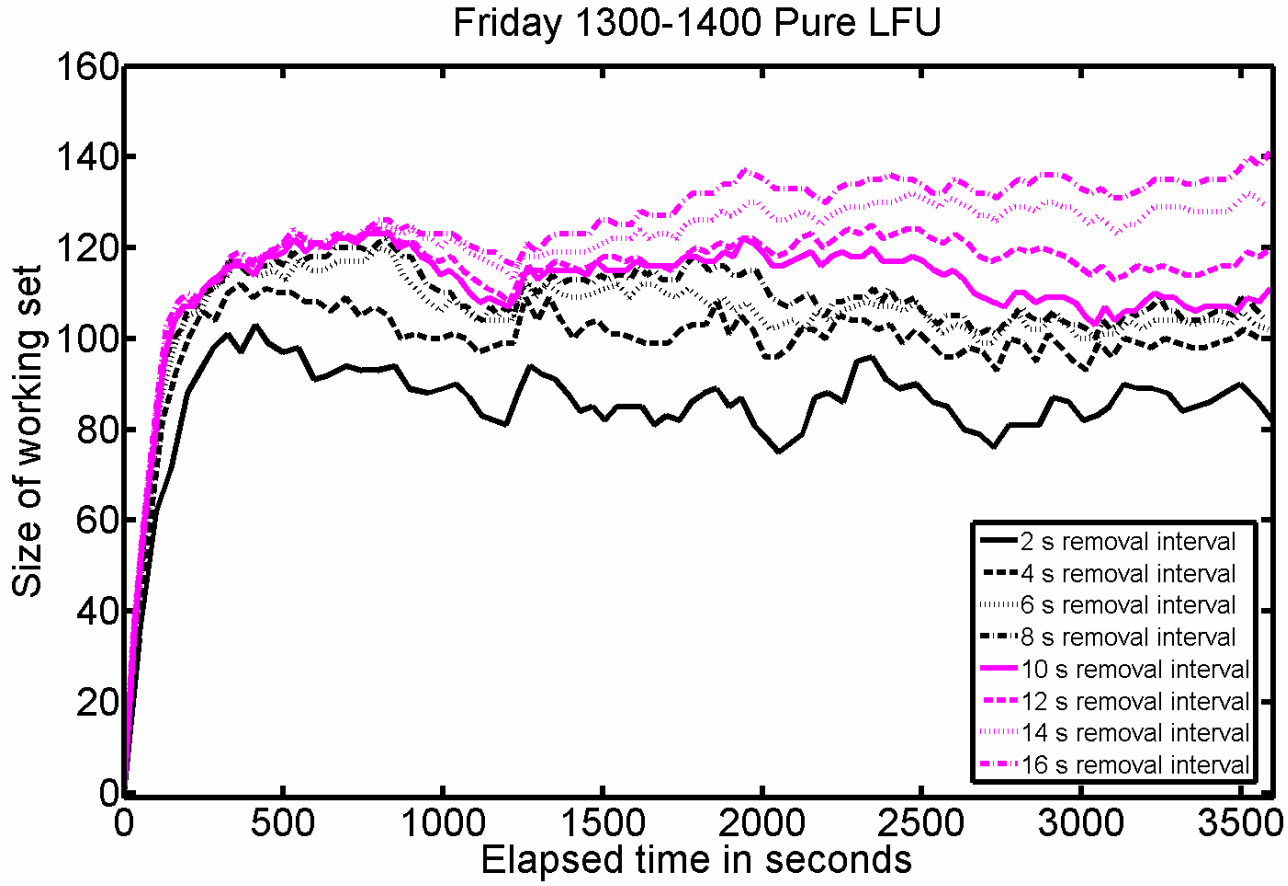

Figure 27: Pure LFU working-set size over time for Friday 1300-1400.

In Table 4 below are the average and standard deviation of the working-set sizes for Pure LFU (in the second row), along with the corresponding values for FIFO, LFU, and LRU.

Table 4: Average and standard deviation of working-set sizes for FIFO, Pure LFU, LFU, and LRU.

TUE 1300-1400 FIFO			WED 1300-1400 FIFO			THU 1300-1400 FIFO			FRI 1300-1400 FIFO		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	94.1354	15.18371	2	93.40972	15.26625	2	95.97724	15.33164	2	94.08328	13.90973
4	116.6965	16.72811	4	117.2631	17.74021	4	122.9416	19.11664	4	119.0222	16.13868
6	131.2619	20.06323	6	128.189	21.27813	6	134.2066	22.3873	6	131.1357	18.77506
8	137.4696	21.70332	8	135.1211	22.89263	8	141.9123	24.90416	8	133.7687	18.49262
10	144.6532	25.1202	10	141.2679	25.58505	10	150.173	29.00525	10	140.7464	21.18444
12	151.1276	28.19478	12	147.2702	29.4063	12	155.0521	31.48083	12	147.3576	23.87658
14	154.784	29.07482	14	150.8496	30.97163	14	158.5224	32.47499	14	147.1365	22.21539
16	156.8721	30.4883	16	152.2791	31.46465	16	160.4306	33.47076	16	149.0917	23.09873

TUE 1300-1400 Pure LFU			WED 1300-1400 Pure LFU			THU 1300-1400 Pure LFU			FRI 1300-1400 Pure LFU		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	82.19076	10.60655	2	83.17174	11.20948	2	87.48898	12.68518	2	85.60893	10.77479
4	98.14181	11.86659	4	95.77806	11.06286	4	99.96324	12.51139	4	99.35185	11.87802
6	103.5512	12.47159	6	100.9121	12.85947	6	106.7847	13.90089	6	105.1961	12.70721
8	106.6457	13.02554	8	104.3382	13.19811	8	110.2719	14.48091	8	107.8858	12.99914
10	109.6974	14.06666	10	107.3027	13.17077	10	115.9035	15.21655	10	111.5756	13.65139
12	115.6323	15.62002	12	111.0524	14.27554	12	123.941	18.65345	12	115.4602	13.95963
14	121.1116	16.43227	14	117	15.98177	14	127.9146	21.10034	14	121.32	15.51598
16	124.6818	18.11475	16	121.338	18.01958	16	131.0415	22.29263	16	125.1781	16.89398

TUE 1300-1400 LFU			WED 1300-1400 LFU			THU 1300-1400 LFU			FRI 1300-1400 LFU		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	80.03828	11.5594	2	79.92029	11.14617	2	84.5231	12.18657	2	83.50947	10.52625
4	97.15159	12.23602	4	95.73469	11.54653	4	101.8289	13.13805	4	100.2901	12.44935
6	104.1555	12.64089	6	102.4579	13.47794	6	114.6442	16.64626	6	110.8538	13.71508
8	110.1655	13.80899	8	108.4493	14.60779	8	121.2482	19.3787	8	111.986	13.93708
10	112.853	14.69709	10	111.0178	14.2978	10	123.4503	19.31719	10	117.5291	14.86584
12	115.3058	15.29254	12	113.3462	15.42484	12	126.0174	20.13101	12	120.7612	15.74368
14	123.6853	17.43104	14	120.2186	17.47435	14	129.1057	21.35351	14	122.976	16.83019
16	128.1773	19.70972	16	125.9861	21.37095	16	137.424	26.33469	16	128.484	19.29472

TUE 1300-1400 LRU			WED 1300-1400 LRU			THU 1300-1400 LRU			FRI 1300-1400 LRU		
Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev	Int	Avg	Stdev
2	104.8296	13.60995	2	103.285	13.61165	2	105.3009	14.12482	2	104.2106	12.45154
4	106.2313	12.49907	4	104.6833	11.73404	4	106.514	12.57479	4	105.2126	11.76718
6	106.7788	11.95829	6	104.8936	12.46381	6	106.9543	12.29957	6	105.7036	11.22774
8	107.3388	11.61129	8	105.4237	12.08093	8	107.5095	12.01021	8	106.0678	10.7572
10	107.841	11.86634	10	106.3185	11.75404	10	108.5777	11.70998	10	106.6297	10.73969
12	109.0034	11.92606	12	107.6035	12.30293	12	109.9931	12.07931	12	107.3854	10.58042
14	110.92	11.80549	14	109.687	12.55118	14	114.151	12.98813	14	109.2731	10.74708
16	113.7945	12.65953	16	113.4419	13.75382	16	120.588	15.49437	16	115.445	12.34627

Comparing Pure LFU with LFU, the Pure LFU averages and standard deviations are almost always lower than those for LFU. However, as with LFU, the Pure LFU values are not as consistent (with varying removal interval) as those for LRU.

CHAPTER 5: PROPERTIES OF THE REAL DATA

5.1 Introduction

Why should the network server traffic data analyzed in the preceding chapters exhibit properties of locality? We examined the frequency distributions of incoming access counts, sorted by connection (i.e., the incoming IP address), along with the rate of arrival of connections not already in the working-set.

We hypothesized that the access counts would conform to a Zipf distribution, in which the frequency of an occurrence is inversely proportional to its rank raised to a power of at least 1. The Zipf distribution appears in many Internet-related statistics [17], such as the number of users accessing a website within a given timeframe and the number of links to and from webpages. One of its important applications is in caching webpages locally for faster access [18].

5.2 Access Counts by Connection

Figures 28, 29, 30, and 31 below graph the access-count data, along with a best-fit curve of the following form:

$$f(i) = \frac{c}{i^{\alpha}} \tag{1}$$

where i is the rank of the IP address in terms of access count (1 being the highest) and c and α are constants. The curve minimizes the sum of the square of the distances between the curve and each data point.

Note that, on Friday, the IP addresses ranked 1 and 2 both had very high access counts, skewing the best-fit curve upward when compared to the curves for Tuesday, Wednesday, and Thursday.

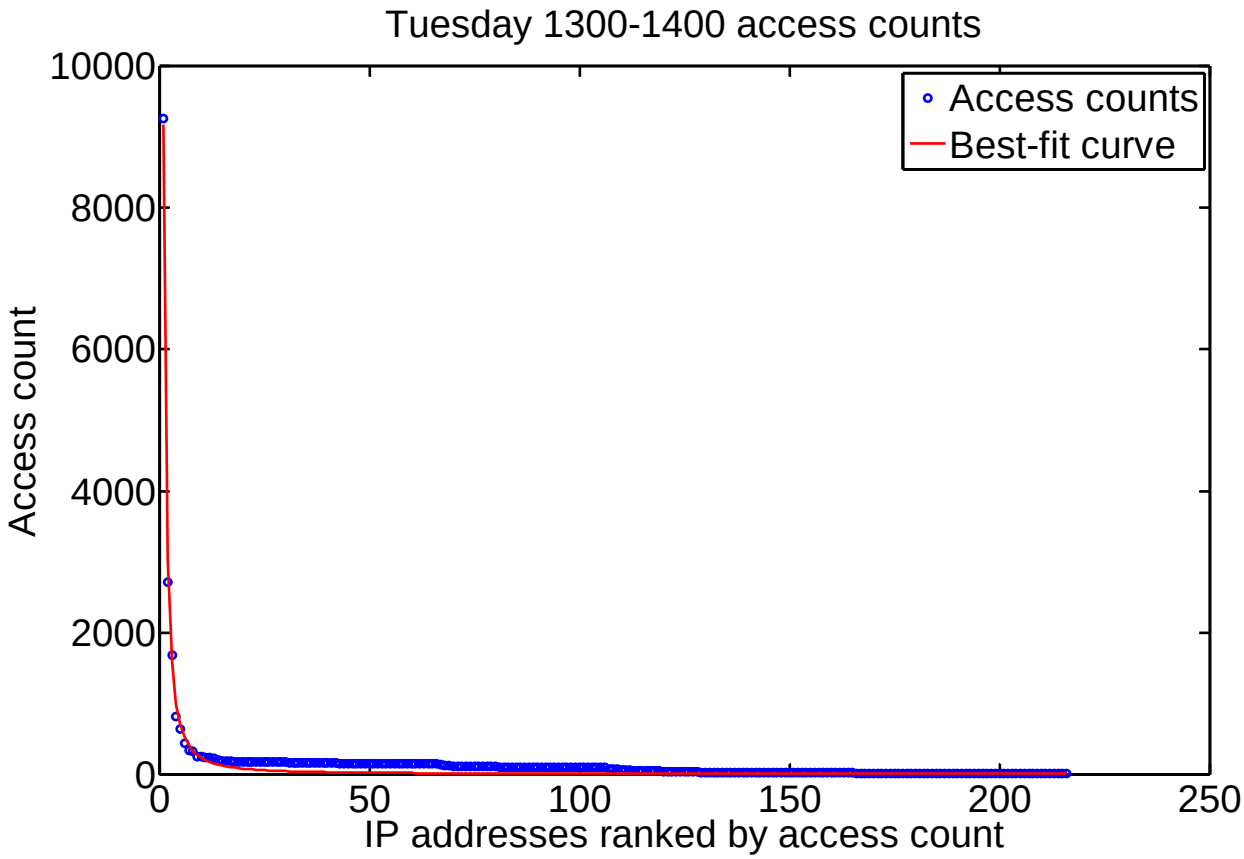

Figure 28: Tuesday 1300-1400 access counts with best-fit curve.

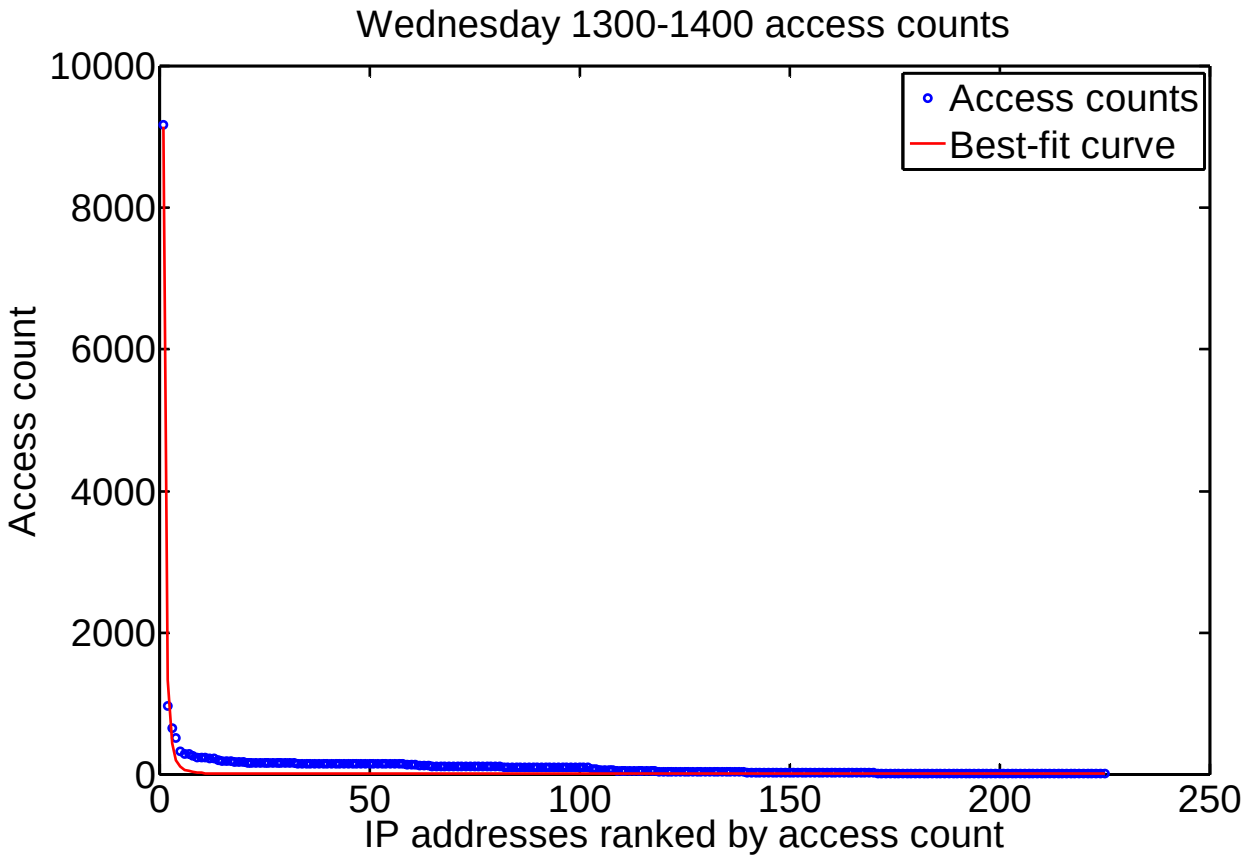

Figure 29: Wednesday 1300-1400 access counts with best-fit curve.

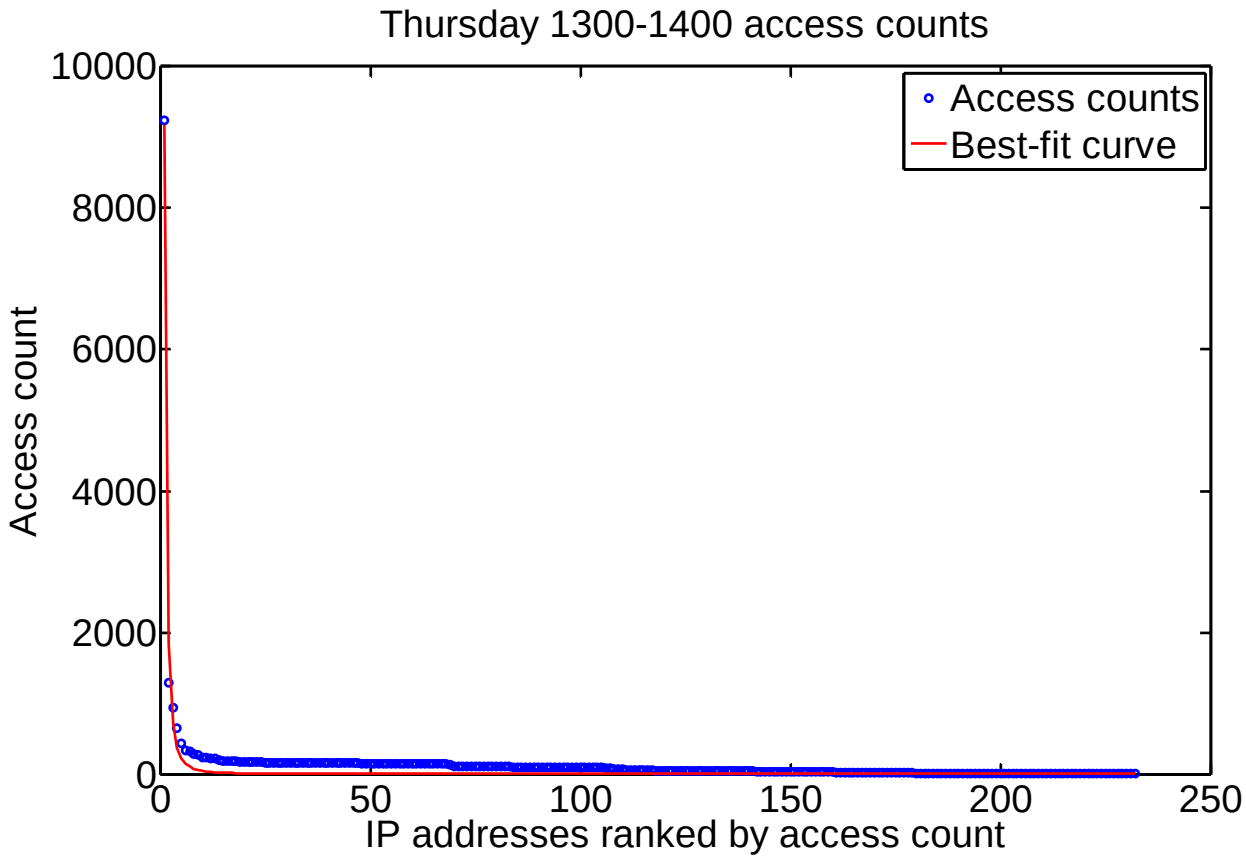

Figure 30: Thursday 1300-1400 access counts with best-fit curve.

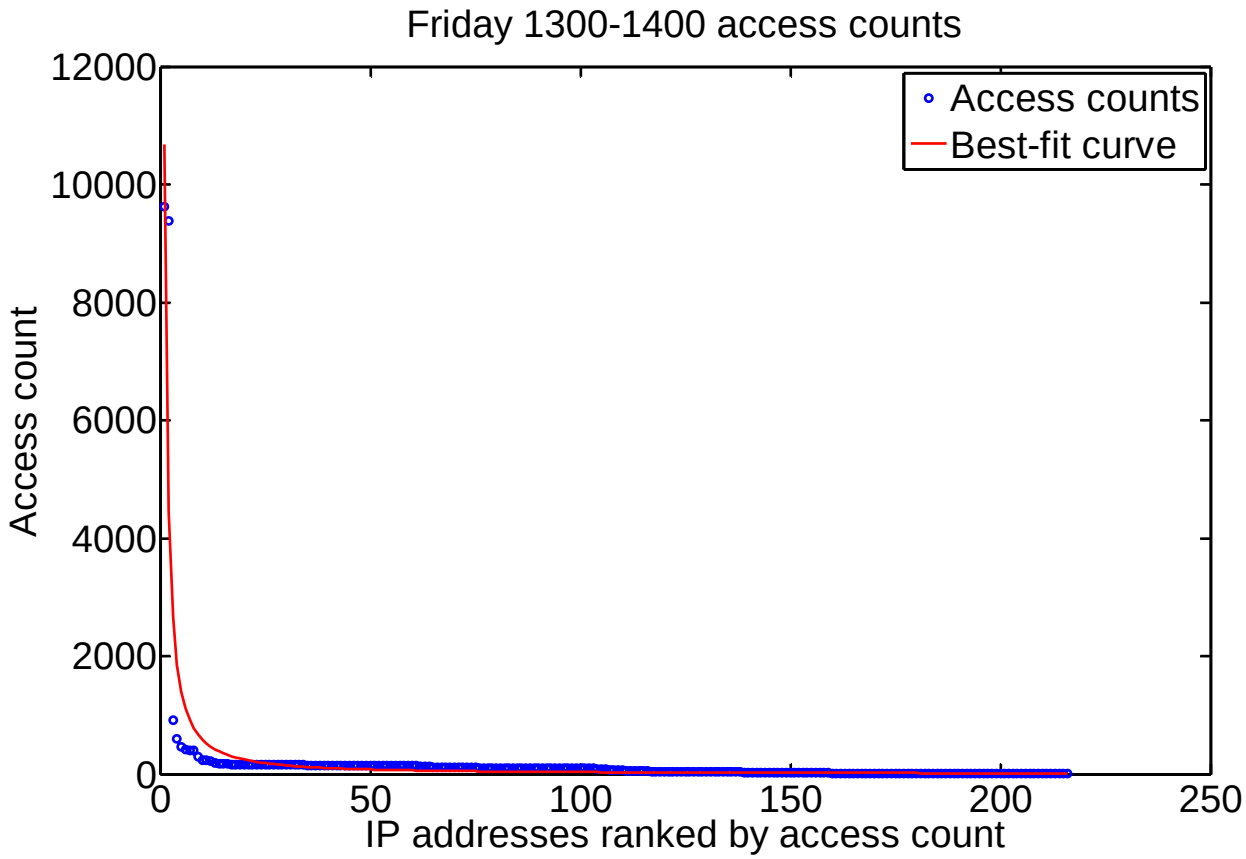

Figure 31: Friday 1300-1400 access counts with best-fit curve.

5.3 Curve Parameters

Table 5 below shows the parameters of the function $f(i) = \frac{c}{i^{\alpha}}$ for each of the best-fit curves shown in Figures 1-4.

Table 5: Parameters for the best-fit curves.

Period	c	α
Tuesday 1300-1400 (Figure 28)	9179.17443	1.60422
Wednesday 1300-1400 (Figure 29)	9149.89145	2.78208
Thursday 1300-1400 (Figure 30)	9240.98300	2.32757
Friday 1300-1400 (Figure 31)	10675.87359	1.26270

5.4 New Arrival Probability

Another characteristic of the data that could influence the behavior of the working-sets is the probability that an incoming packet is new, that is, it does not belong to a connection already in the working-set. We calculated this probability for each minute during the 1300-1400 hour for the Tuesday, Wednesday, Thursday, and Friday data. The results are shown in Figures 32, 33, 34, and 35 below.

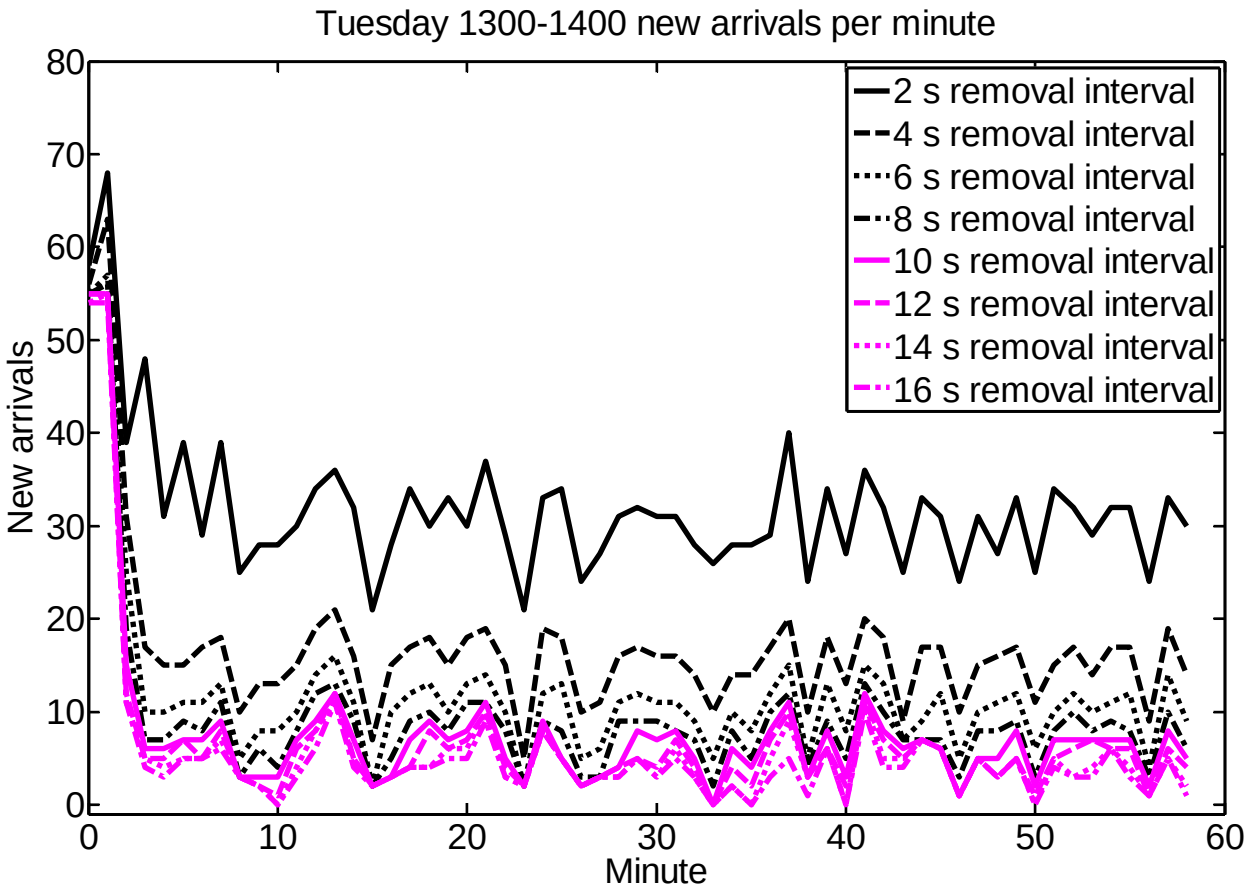

Figure 32: Tuesday 1300-1400 new arrivals per minute.

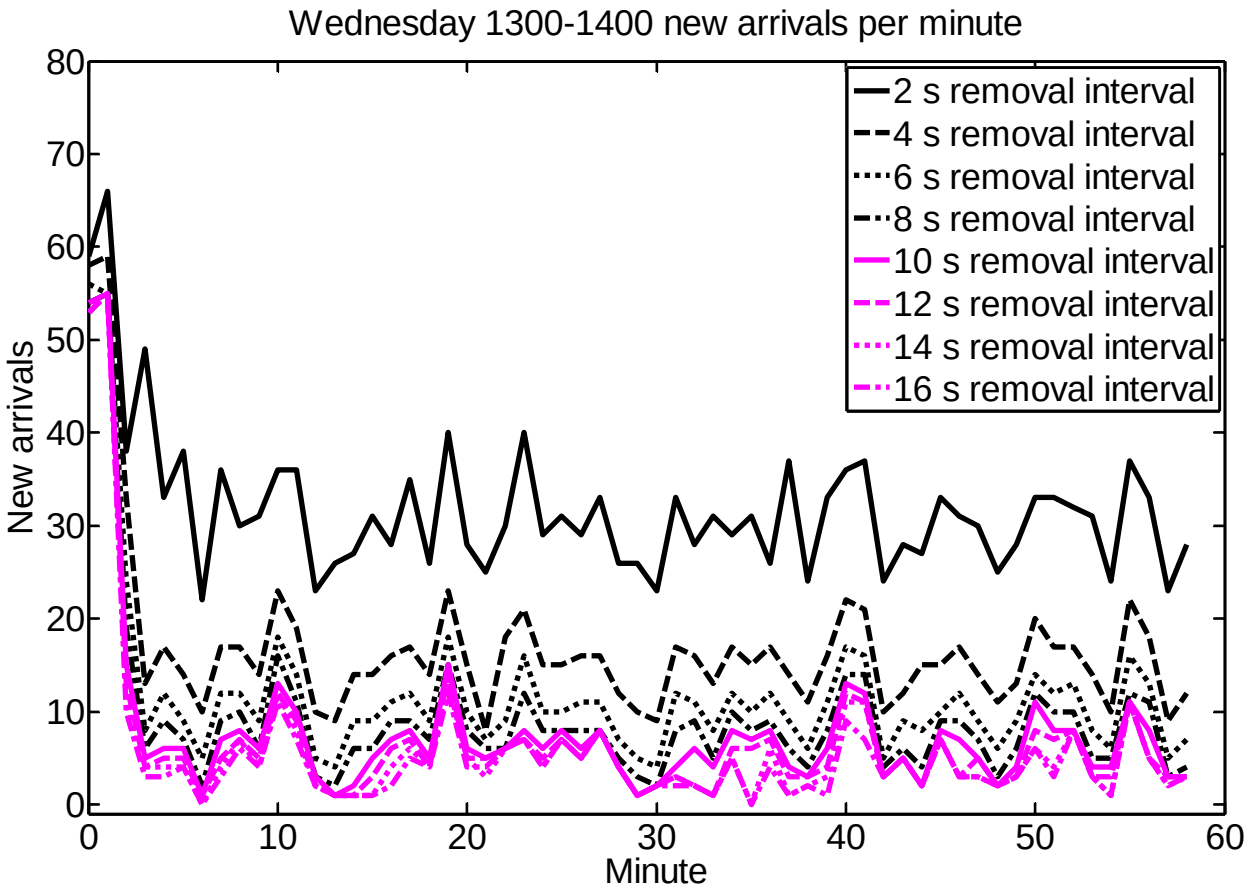

Figure 33: Wednesday 1300-1400 new arrivals per minute.

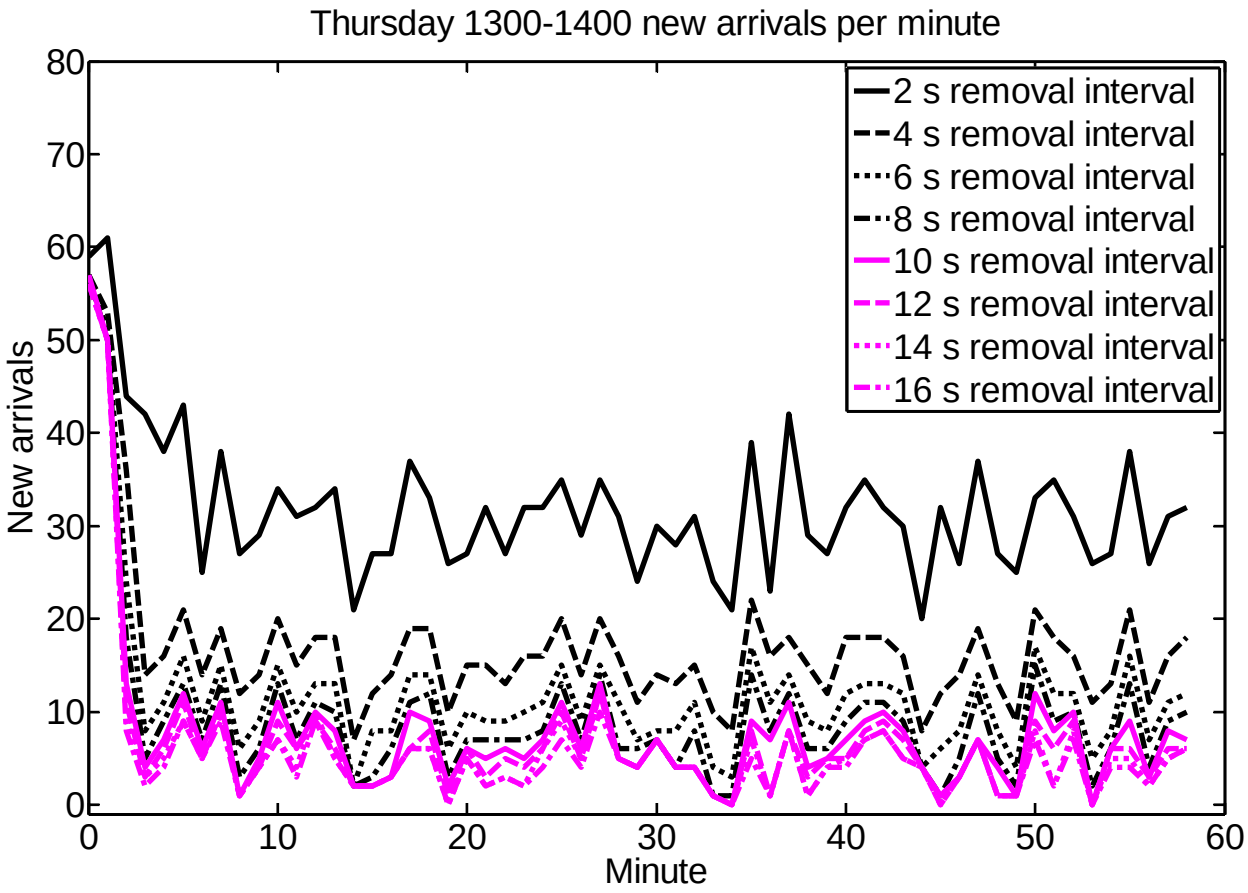

Figure 34: Thursday 1300-1400 new arrivals per minute.

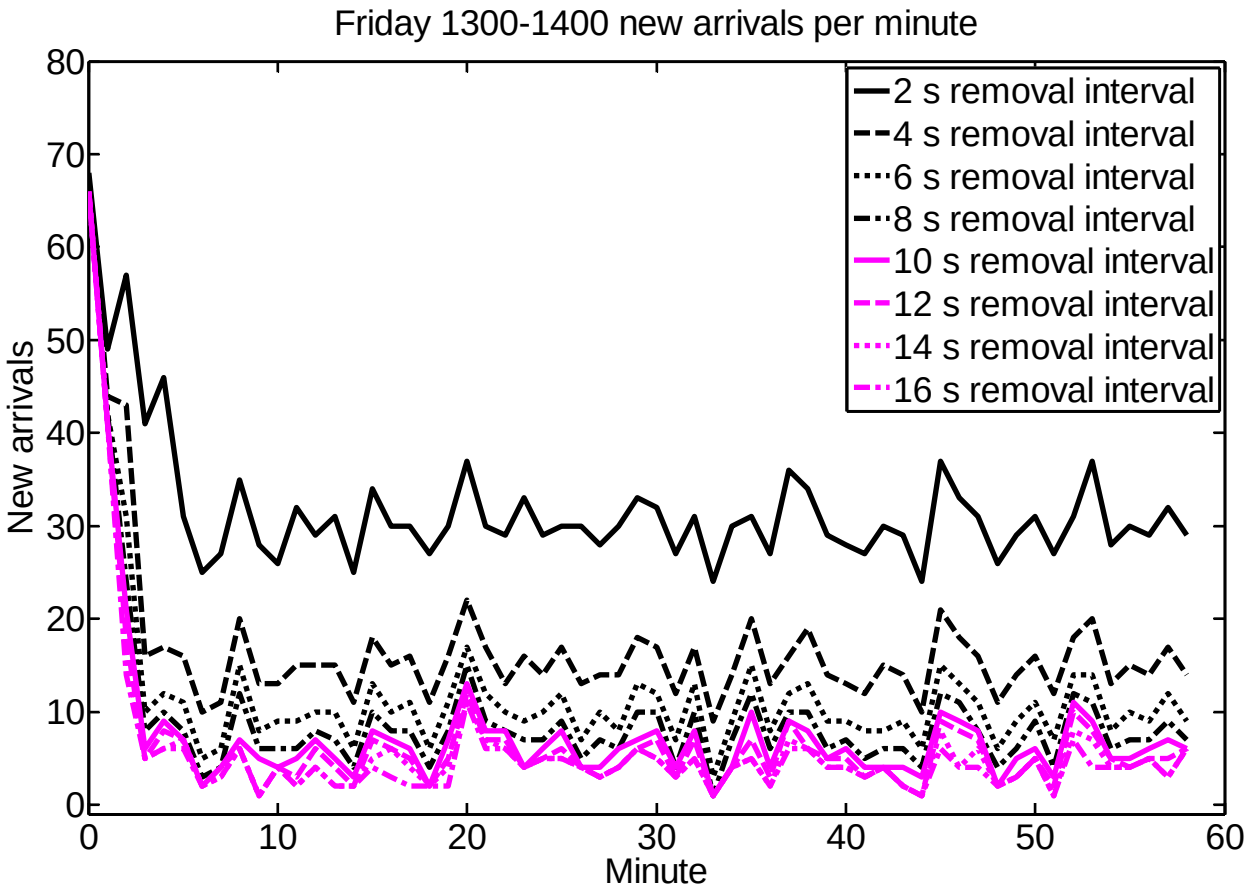

Figure 35: Friday 1300-1400 new arrivals per minute.

From these four figures, we can see that the number of new arrivals per minute is relatively constant for each of the removal intervals.

We expanded this investigation by looking at the probability that a packet was a new arrival over an entire hour of data. Figure 36 below shows the results for all hours for which data was collected.

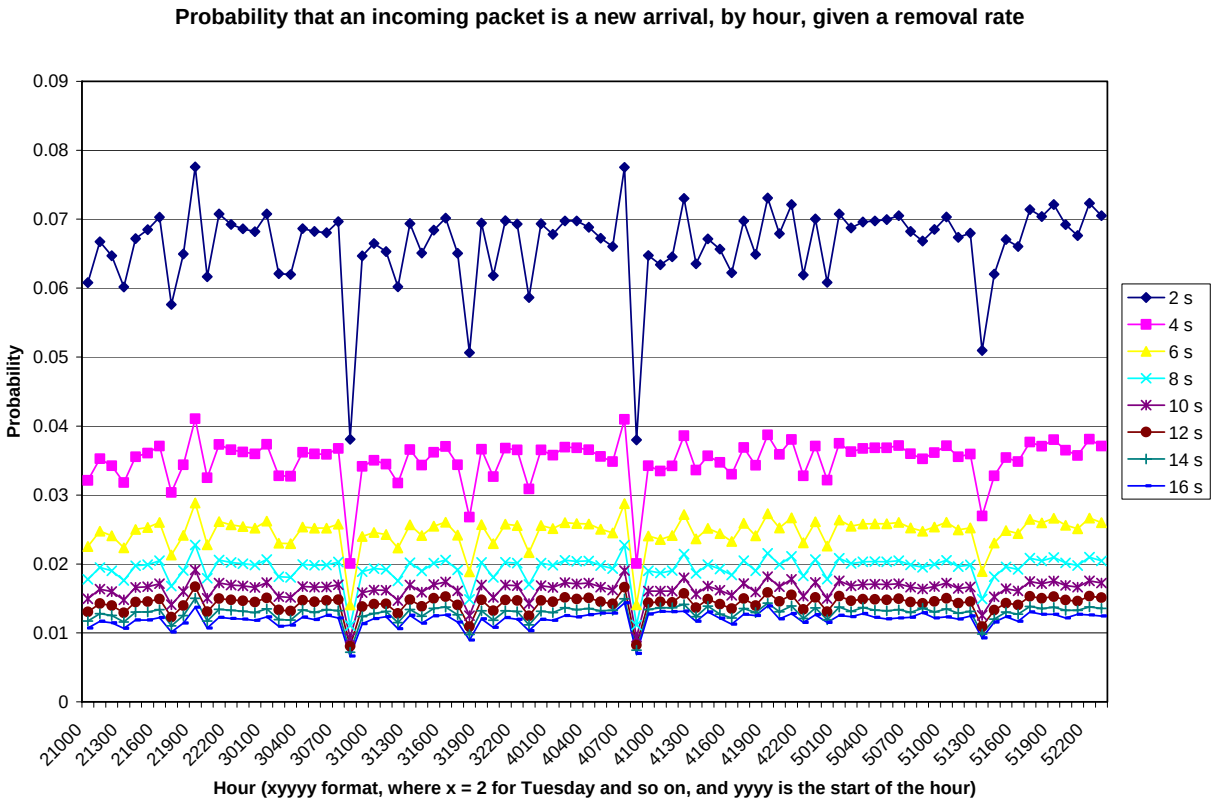

Figure 36: Probability that an incoming packet is a new arrival, by hour. The removal interval varies from 2 s to 16 s.

Given a certain removal interval, the probability that an incoming packet is a new arrival is close to constant across the hours of data collected.

5.5 Bias of In-Packets

Having looked at the probability of new arrivals, which we will also refer to as "out-packets" because they are packets corresponding to connections not in the working-set, we proceeded to

look at “in-packets”, which naturally are packets corresponding to connections already in the working-set. We investigated the hypothesis that in-packets would more likely correspond to connections that had been used more recently. In other words, the more recently a connection had received a packet, the greater the probability that it would receive another packet. We processed the 21300 data (Tuesday 1300-1400) using our working-set algorithm with removal intervals from 2 s to 16 s. We waited 1000 seconds for the working-set sizes to stabilize, then we started classifying each in-packet based on how recently the connection to which it belonged had been accessed. There were 10 categories, from the 10% least recent to the 10% most recent connections in the working-set. The results are shown in Figure 37 below.

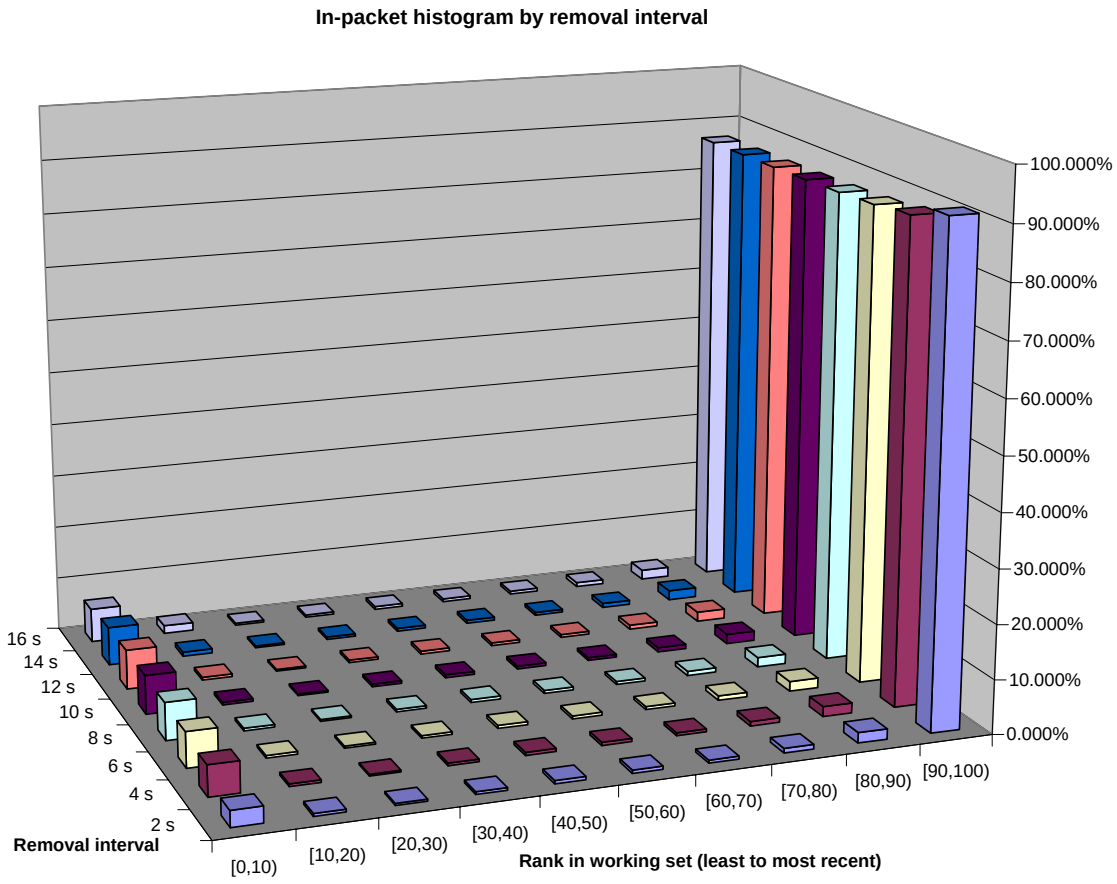

Figure 37: In-packet histogram by removal interval for 21300 data.

For all removal intervals, the probability that an in-packet belonged to the 10% most recent connections was close to 90%. The probabilities were very stable across removal intervals. This finding may help to explain why the LRU removal method is very effective at preserving a constant working-set size.

We will use the information gathered in the preceding sections to build a model for the network traffic behavior.

CHAPTER 6: SYNTHETIC PACKETS (LRU)

6.1 Introduction

To understand better the behavior of the actual network traffic, we decided to generate a synthetic model. Specifically, our goal was to generate a set of incoming packets that would demonstrate locality under our LRU working-set system. We needed to choose an appropriate distribution for the arrival times of the packets and a suitable method of generation.

6.2 Poisson Distribution

In queueing theory, a Poisson distribution is often used to model the behavior of arrivals to the queue [19]. In our case, the arrivals are individual packets, and the queue is the working-set. We decided to use a Poisson distribution for the interarrival times of our synthetic packets. The average interarrival time of 0.12 s would produce approximately 30000 packets in an hour, which is close to the number of packets in an hour of the actual data.

The interarrival time generation method used the following formula:

$$t = \frac{-1}{\frac{1}{\lambda} \cdot \log(r)},$$

where λ is the desired average interarrival time and r is a random number greater than 0 and less than 1.

6.3 Pseudorandom Number Generator

For the pseudorandom number generator used in generating interarrival times and source IP addresses, we chose the Mersenne Twister algorithm designed by Matsumoto and Nishimura [20]. The period of a pseudorandom number generator is the quantity of random numbers it will generate before a non-unique number is generated. This algorithm has a period of $2^{19937}-1$, which is approximately 4.3×10^{6001}. This is suitably large, because we will not need to generate nearly as many random numbers. For an hour of network data, our program will generate fewer than 10^7 random numbers.

6.4 LRU Working-set Generation Method

Based on our prior observation that the probability was constant that an incoming packet would belong to a connection already in the working-set, we decided to build a simple probabilistic model for the synthetic packets using a working-set. Furthermore, the working-set used during generation would be based on an actual working-set taken from processing real data.

We started processing the 21300 hour of data (Tuesday 1300-1400) using the LRU working-set algorithm with a removal interval of 6 s. We stopped the processing at 1000 seconds into the hour, at which point the size of the working-set had already stabilized, and extracted the set of IP addresses in the working-set at that moment. This initial working-set was the seed for generation of synthetic data. Each synthetic packet was generated as follows.

First, generate a random interarrival time conforming to a Poisson distribution, as described in Section 6.2 above. This time is added to the current time, which starts at 0, to determine the timestamp for the packet.

Second, we need to choose a source IP address for the packet. Based on the results shown in Figure 36 above and corresponding to the removal interval of 6 s, the probability that this packet is not already in the working-set is chosen to be 0.02. Generate a random number x at least 0 and less than 1. If x is less than 0.02, this packet should be a new arrival; thus, we generate a random source IP address not already in the working-set. If x is not less than 0.02, we choose a source IP address already in the working-set. The choice of IP address from the working-set is biased towards the most recently used IP addresses based on the results of Figure 37 above: with probability 0.9, the IP address is chosen randomly from the 10% most recent connections in the working-set, and with probability 0.1, the IP address is chosen randomly from the remaining 90% of the connections in the working-set.

As each packet is generated, the timestamp increases, hence time progresses. During the generation, our program removes the least recently used connection from the working-set at each removal interval, in this case 6 s. This removal mimics the behavior of the LRU working-set algorithm. The generator continues until the timestamps of the packets encompass one hour.

6.5 LRU Synthetic Packet Results

After the synthetic packets have been generated, we process them using the LRU working-set algorithm, just as we processed the real packets. The results of this processing are shown below in Figure 38.

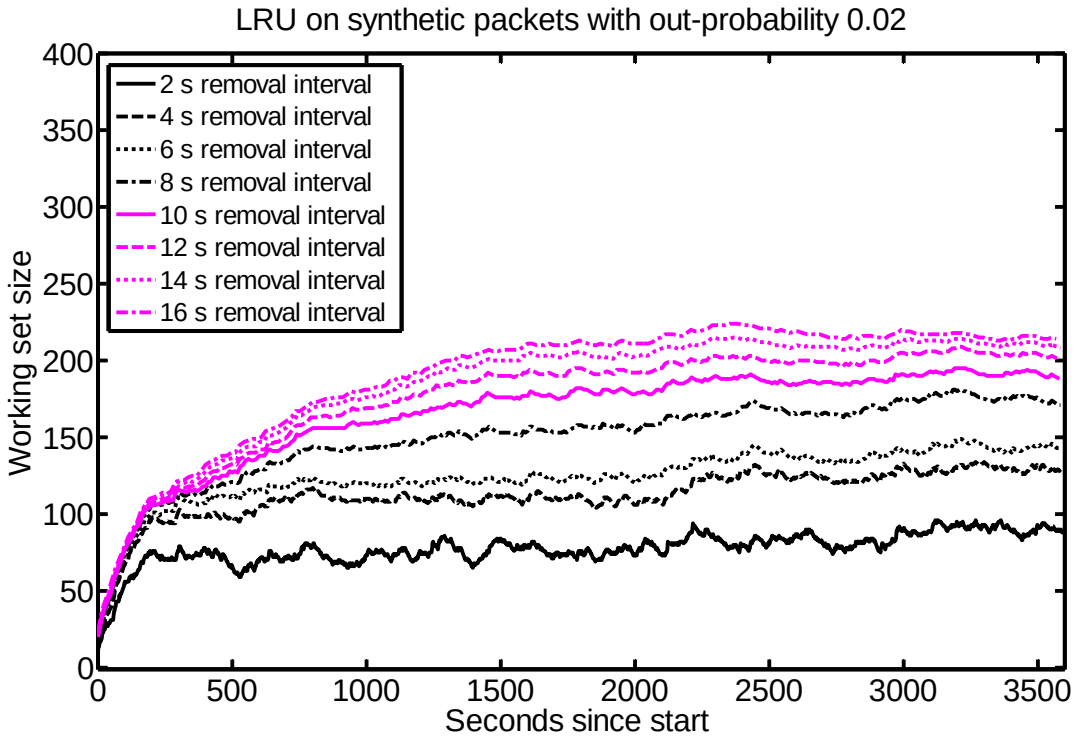

Figure 38: LRU analysis of LRU synthetic packets.

By inspection, the working-set sizes do stabilize to a constant level, but the curves for the different removal intervals are not close to each other. Compare these with the curves for the actual 21300 data in Figure 2 above. By quantitative measures, these results are also markedly different from those for the real data. Table 6 below shows the average and standard deviation of the working-set sizes for the processing of the synthetic data versus the same for the real 21300 data. Also included is the "time to range", which is the time elapsed before the working-set size first exceeds the average minus the standard deviation.

Table 6: Average, standard deviation, and time to range for LRU synthetic packets, 21300 seed.

Average and Standard Deviation of Working-set Sizes, with Time to Range			
LRU Analysis of LRU Synthetic Packets, 21300 seed			
Removal Interval	Avg	Stdev	TimeToRange
2	58.06837	7.384786	92
4	83.13237	10.04801	116
6	91.95659	10.88465	132
8	120.2339	18.53213	184
10	137.9304	24.11574	290
12	154.1572	30.73719	408
14	166.2879	35.4793	420
16	173.4063	38.03488	448
LRU Analysis of TUE 1300-1400 Packets			
Removal Interval	Avg	Stdev	TimeToRange
2	104.8296	13.60995	216
4	106.2313	12.49907	134
6	106.7788	11.95829	126
8	107.3388	11.61129	119
10	107.841	11.86634	123
12	109.0034	11.92606	123
14	110.92	11.80549	132
16	113.7945	12.65953	130

The average working-set sizes for the synthetic data across the various removal intervals are not as close together as those for the real data. At the same time, the standard deviations of the working-set sizes for the synthetic data increased as the removal interval increased, while those for the real data remained small. Finally, the time to range for the synthetic data not only increased as the removal interval increased, but was also generally greater in magnitude compared to the same for the real data, indicating that the working-sets for the synthetic data required more time to stabilize.

Recall that the packet counts by IP address for the real data conformed to a Zipf distribution. We investigated whether the same was true for the synthetic data. Figure 39 below shows the access count distribution for the synthetic data, and Figure 40 shows the same for the real data.

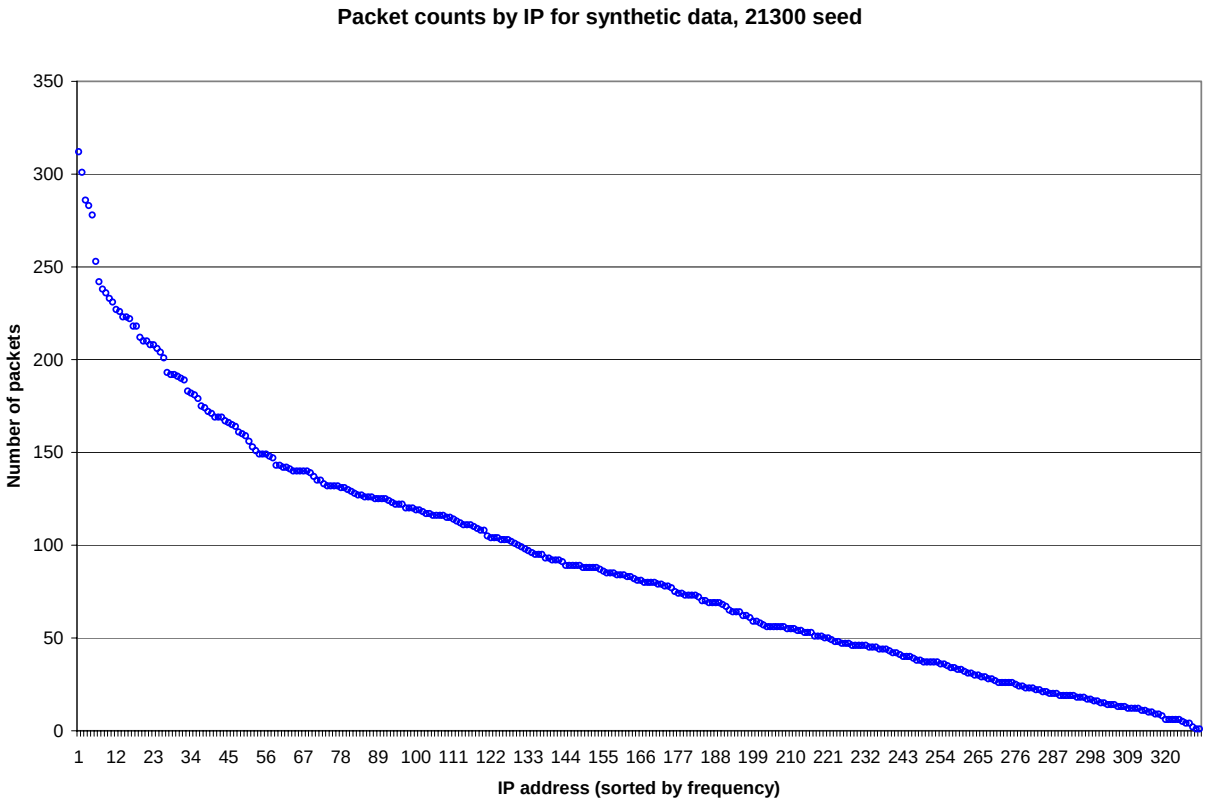

Figure 39: Packet counts by IP for LRU synthetic data, 21300 seed.

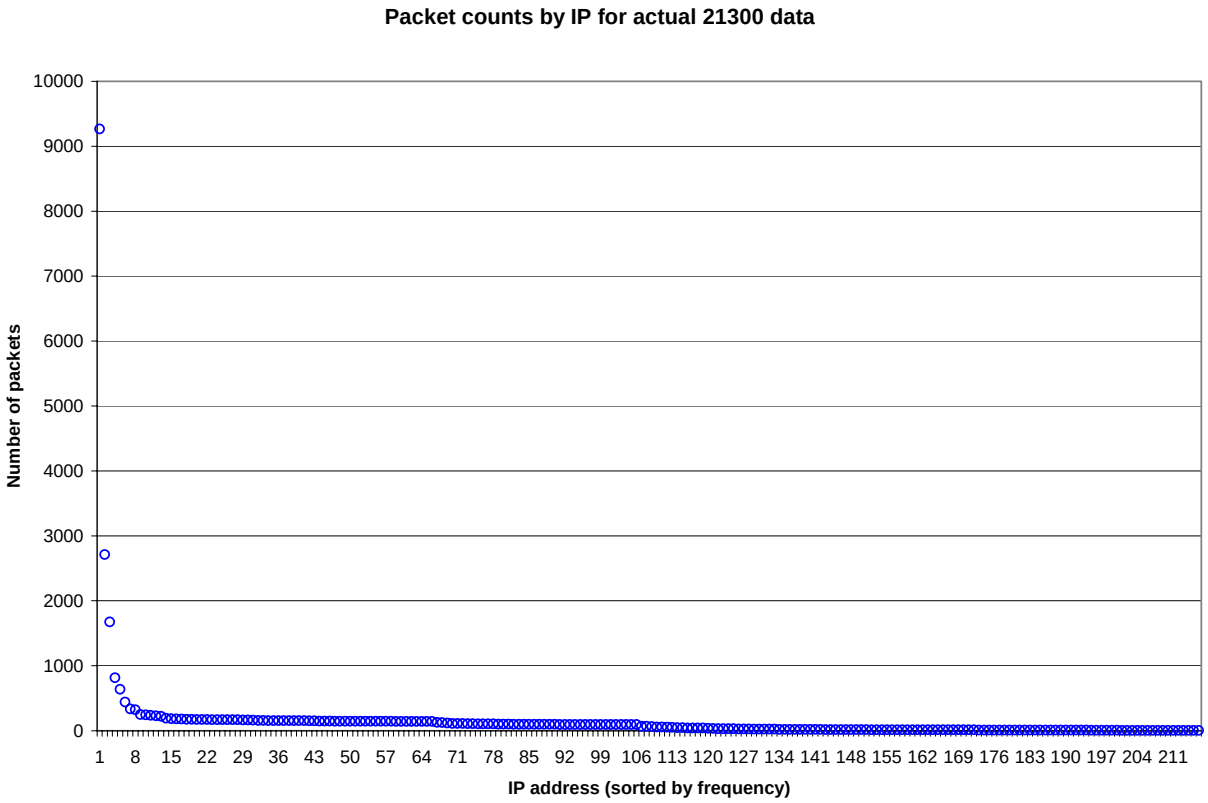

Figure 40: Packet counts by IP for real 21300 data.

Note that the scale of the vertical axis for Figures 39 and 40 is very different. The packet counts for the synthetic data in Figure 39 are all less than 350. For the real data in Figure 40, there are about five connections with relatively high packet counts compared to the rest of the connections. Unlike the real data, the synthetic data do not conform to a Zipf distribution.

What can be done to make the synthetic data more like the real data? Could the Zipf distribution of the packet counts in itself be causing the real data to have properties of locality? We investigate this question in the next chapter.

CHAPTER 7: SYNTHETIC PACKETS (ZIPF)

7.1 Introduction

We have already seen that the packet counts by IP for the real data conform to a Zipf distribution. Can we generate synthetic data that behaves like the real data simply by forcing the packet counts to conform to a Zipf distribution?

7.2 Zipf Synthetic Packet Generation Method

In a graph of packet counts with a Zipf distribution such as that in Figure 40 above, the data points fit a curve of the following form:

$$f(i) = \frac{c}{i^{\alpha}}.$$

The two parameters in this formula are c and α. For the real data, as seen in Table 5 above, c ranged from about 9000 to 11000 and α ranged from about 1.2 to 2.8. We chose 10000 for c and 1.6 for α to correspond roughly with the figures for the real 21300 data.

There is another parameter involved that is less obvious: the total number of different IP addresses that connect to the server. Because there are approximately 200 different incoming IP

addresses in the real data, as can be observed on the horizontal axes of Figures 28, 29, 30, and 31.

As did the LRU synthetic packet generator in Chapter 6, our generator uses the Mersenne Twister pseudorandom number generator and a Poisson distribution for the interarrival times. The mean interarrival time was again 0.12 s, so the total number of packets generated would be about 30000, to correspond with the real data. We begin by generating 200 different random IP addresses. These represent all the source IP addresses that connect to the server during one hour, and they are distinguishable by their order: the first IP address is the one that will have the highest packet count, and the two-hundredth IP address will have the lowest packet count. Next, for each IP address, we calculate the desired probability that a packet will be sent from that IP address. This probability is given by the formula

$$\frac{f(i)}{\sum_{j=1}^{200} f(j)}$$

where i is the order of the IP address, from 1 to 200, and $f(i) = \frac{c}{i^{\alpha}}$. The denominator is the sum of all the packet counts; in other words, the total number of incoming packets. Each packet is generated as follows.

First, generate a random interarrival time conforming to a Poisson distribution, as described in Section 6.2 above. This time is added to the current time, which starts at 0, to determine the timestamp for the packet.

Second, we need to choose a source IP address for the packet. Pick a random number *x*, at least 0 and less than 1. Starting with $i = 1$, take the sum of the probabilities for each IP address from $j = 1$ to *i*. If this sum is greater than *x*, then let *i* be the source IP address for this packet; otherwise, increment *i* and continue.

Packets are generated until the packet timestamps span one hour. Note that, in this generation method, there is no working-set and no removal interval.

7.3 Zipf Synthetic Packet Results

After generating the Zipf synthetic packets, we processed them using our LRU working-set algorithm with removal intervals from 2 s to 16 s. The results are shown in Figure 41 below.

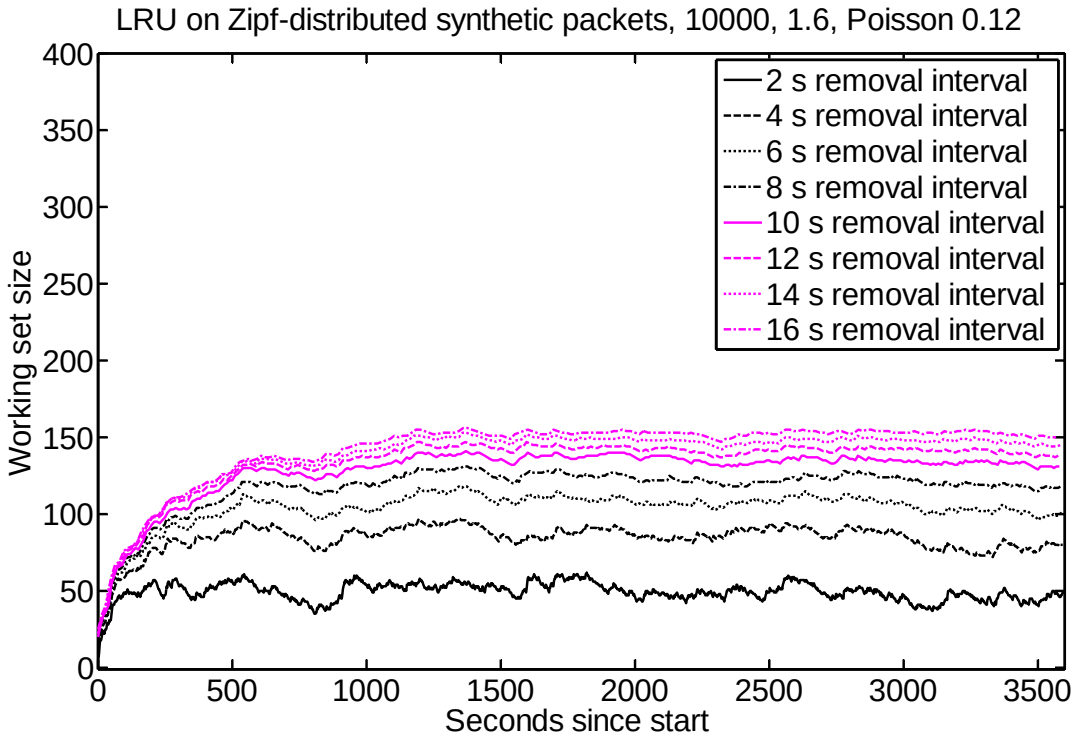

Figure 41: LRU analysis of Zipf synthetic packets.

These curves are similar to those in Figure 38 above, for the synthetic packets generated using a working-set and the LRU removal method. However, the packet counts sorted by IP address for the Zipf synthetic packets do in fact conform to a Zipf distribution, as shown in Figure 42 below.

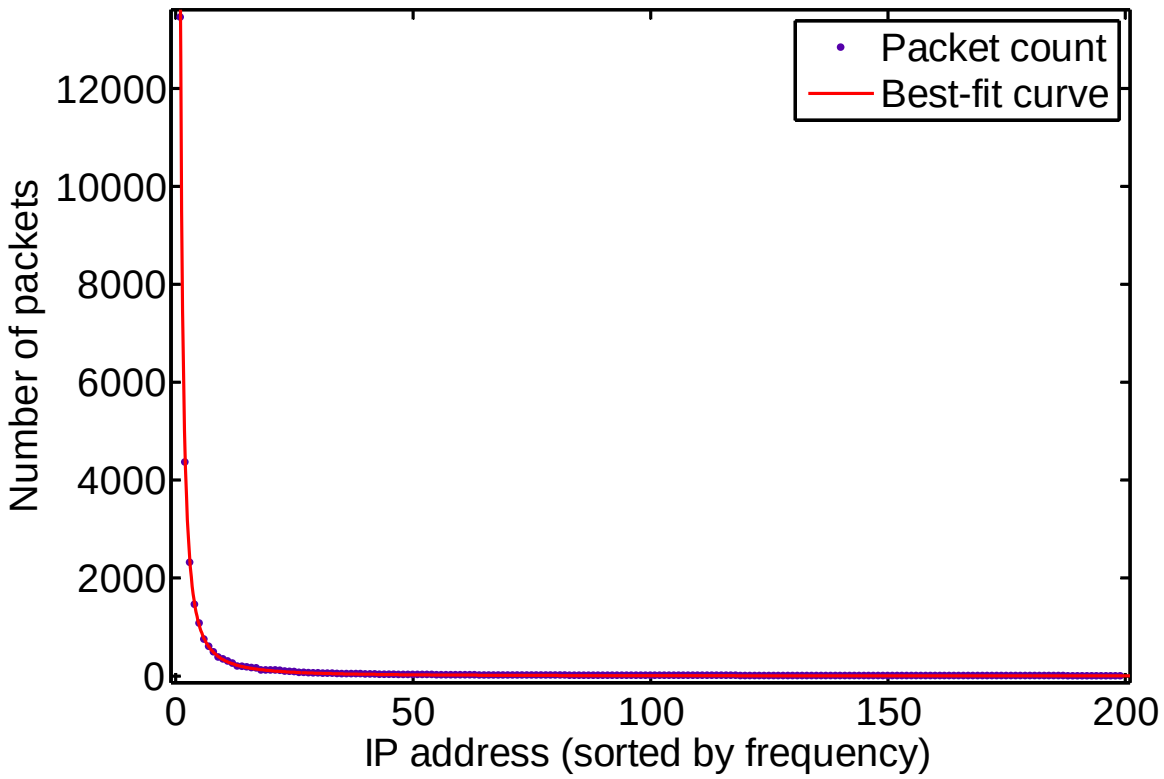

Figure 41: Packet counts by IP for Zipf synthetic packets, with best-fit curve.

The parameters for the best-fit curve were $c = 13447$ and $\alpha = 1.604$. Although packet counts for these synthetic packets do conform to a Zipf distribution, the results of the LRU working-set algorithm processing show problems similar to those for the LRU synthetic packets: the average working-set sizes are not consistent across removal intervals and the working-sets require a long time to reach equilibrium.

These results led us to consider whether there might be a way to create a hybrid: to achieve a Zipf distribution while using a working-set during the generation process. Because the Zipf

distribution is based on frequencies of accesses, we were inspired by the word “frequency”. The least-frequently-used removal method is one of the strategies used earlier as part of the working-set algorithm for detecting locality. In the next chapter, we use Pure LFU with a working-set to generate synthetic packets.

CHAPTER 8: SYNTHETIC PACKETS (PURE LFU)

8.1 Introduction

In Chapter 4, we studied the Pure LFU removal method as one way of removing packets from the working-set during processing. While Pure LFU did not perform as well as LRU, we can take advantage of some of its useful properties to generate synthetic packets such that their packet counts by IP address conform to a Zipf distribution. The Pure LFU removal method removes from the working-set the connection with the lowest number of accesses among all the connections currently in the working-set. In contrast, the Pure LFU removal method removes from the working-set the connection with the lowest number of accesses among all connections that have ever accessed the server, not just those currently in the working-set. Thus, to implement the Pure LFU method, we require two sets of connections: the working-set and the set of all connections that have ever accessed the server. Naturally, the working-set is a subset of the set of all connections. We will use both of these connection sets during the generation process.

8.2 Pure LFU Generation Method

In Chapter 6, we processed real data and "froze" the working-set at 1000 seconds into the hour, after the working-set size had stabilized, and then used the connections in the working-set at that time as the basis for packet generation. For Pure LFU, we are freezing two sets of connections:

the working-set and the set of all connections. These sets differ from the LRU working-set in that they keep an access count for each connection in the set. So that we could obtain a better representation of working-set behavior for the entire hour, we froze the working-sets after processing all the packets in the hour of actual data, using the Pure LFU removal method with a removal interval of 14 s. The seed working-set and the seed set of all connections now form the basis for packet generation. Each packet is generated as follows.

First, generate a random interarrival time conforming to a Poisson distribution, as described in Section 6.2 above. This time is added to the current time, which starts at 0, to determine the timestamp for the packet.

Second, we need to choose a source IP address for the packet. The probability that the packet is an out-packet, that is, it belongs to a connection not already in the working-set, is determined based on the results shown in Figure 42 below.

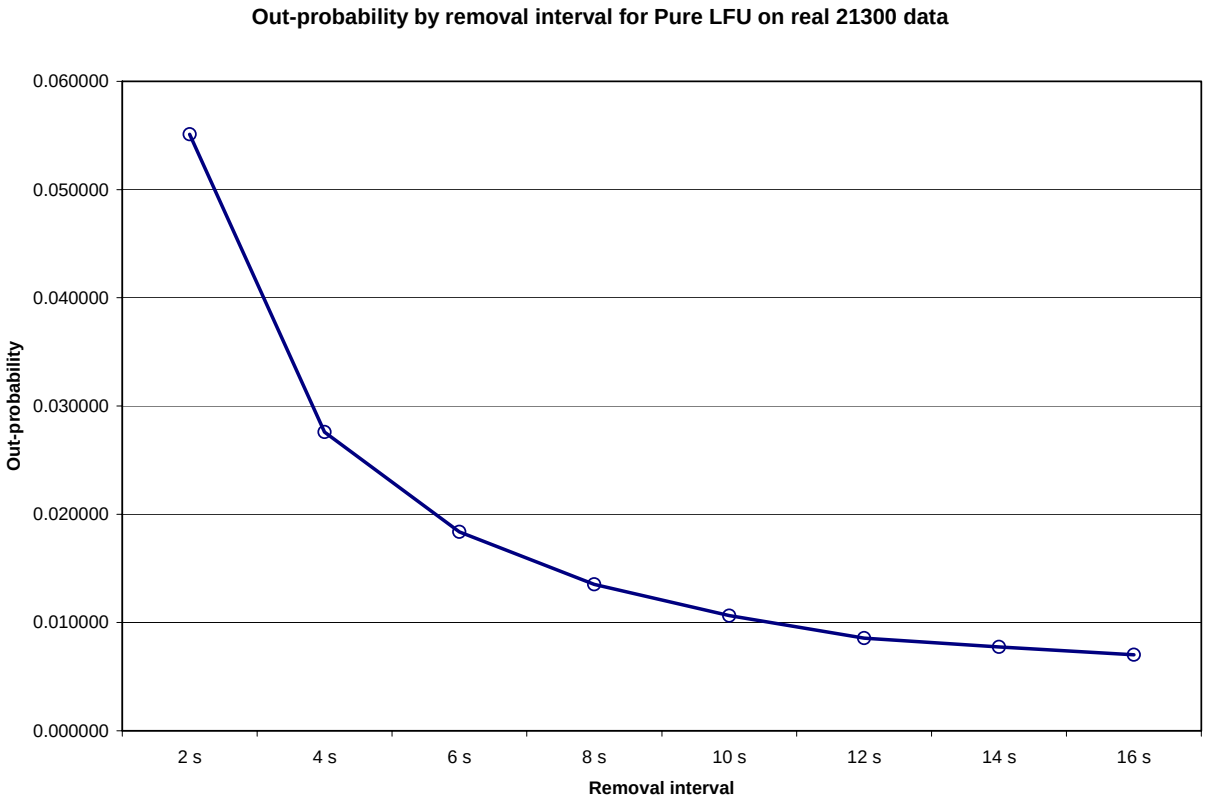

Figure 42: Out-probability for Pure LFU on real 21300 data.

Corresponding to the removal interval of 14 s, the probability that this packet is not already in the working-set is chosen to be 0.008. Generate a random number x at least 0 and less than 1. If x is less than 0.008, this packet should be an out-packet; thus, we randomly select a source IP address from the set of all connections that is not already in the working-set. If x is not less than 0.008, we choose a source IP address already in the working-set. The choice of IP address from the working-set is biased towards the most recently used IP addresses based on the results shown in Figure 43 below.

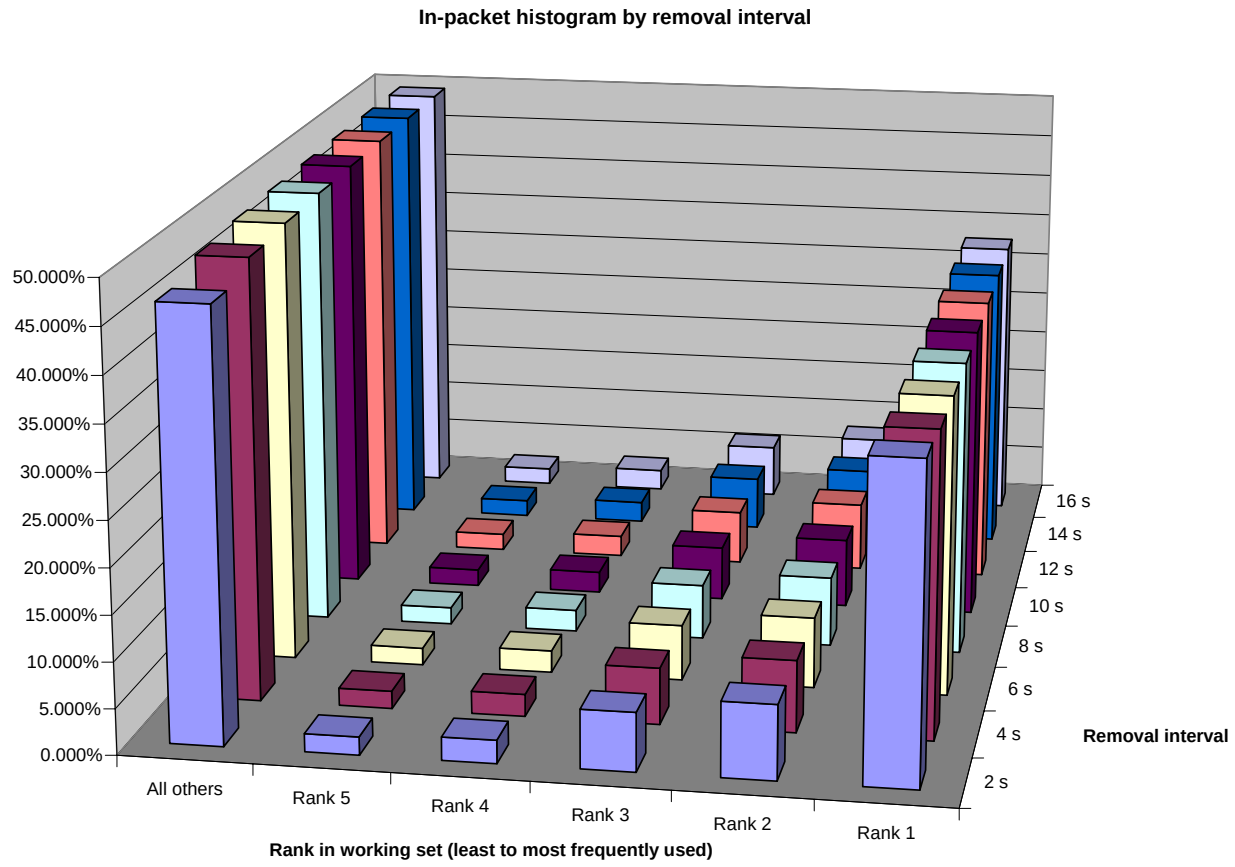

Figure 43: In-packet probability histogram for Pure LFU on real 21300 data.

In Figure 43, “Rank 1” refers to the single most frequently used connection in the working-set. With a probability of about 0.34, that connection was the one to which an in-packet belonged. Recall that an in-packet is a packet belonging to a connection already in the working-set. “Rank 2” refers to the second most frequently used connection in the working-set, and so on. “All others” refers to any connection in the working-set other than the top five individual most frequently used connections.

Based on these results from actual data, the probabilities we chose for the generation of in-packets are shown in Table 7 below.

Table 7: Probabilities for generating in-packets for Pure LFU synthetic data.

Connection	Probability
Rank 1	0.34302
Rank 2	0.08122
Rank 3	0.06385
Rank 4	0.02515
Rank 5	0.01975
All others	0.46699

Thus, with probability 0.34302, the most frequently used connection would be chosen for an in-packet, and so on.

As each packet is generated, the timestamp increases, hence time progresses. During the generation, our program removes the least recently used connection from the working-set at each removal interval, in this case 14 s. This removal mimics the behavior of the LFU working-set algorithm. The generator continues until the timestamps of the packets encompass one hour.

8.3 Pure LFU Synthetic Packet Results

Having generated synthetic packets using a working-set and the Pure LFU removal method, we process the packets with the same LRU working-set algorithm used in analyzing the other synthetic packets. The results are shown in Figure 44 below.

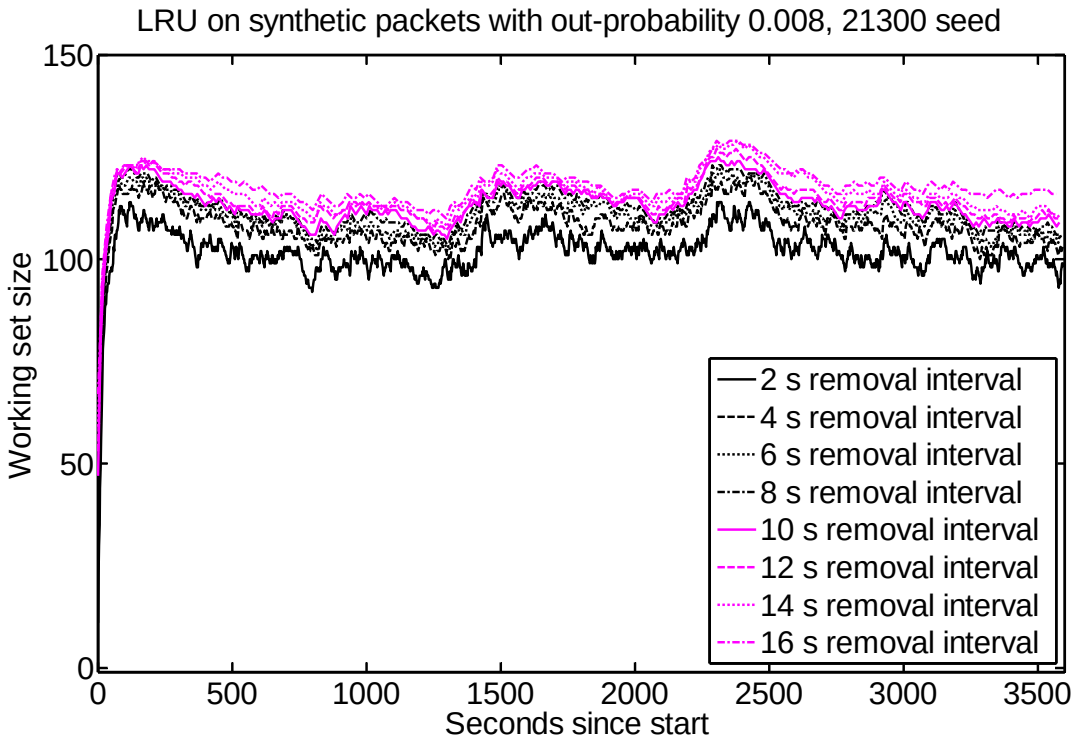

Figure 44: LRU analysis of Pure LFU synthetic packets, 21300 seed.

Compare Figure 44 to the synthetic packet results in Figures 38 and 41 above, and to the results for the real 21300 data, reproduced below in Figure 45.

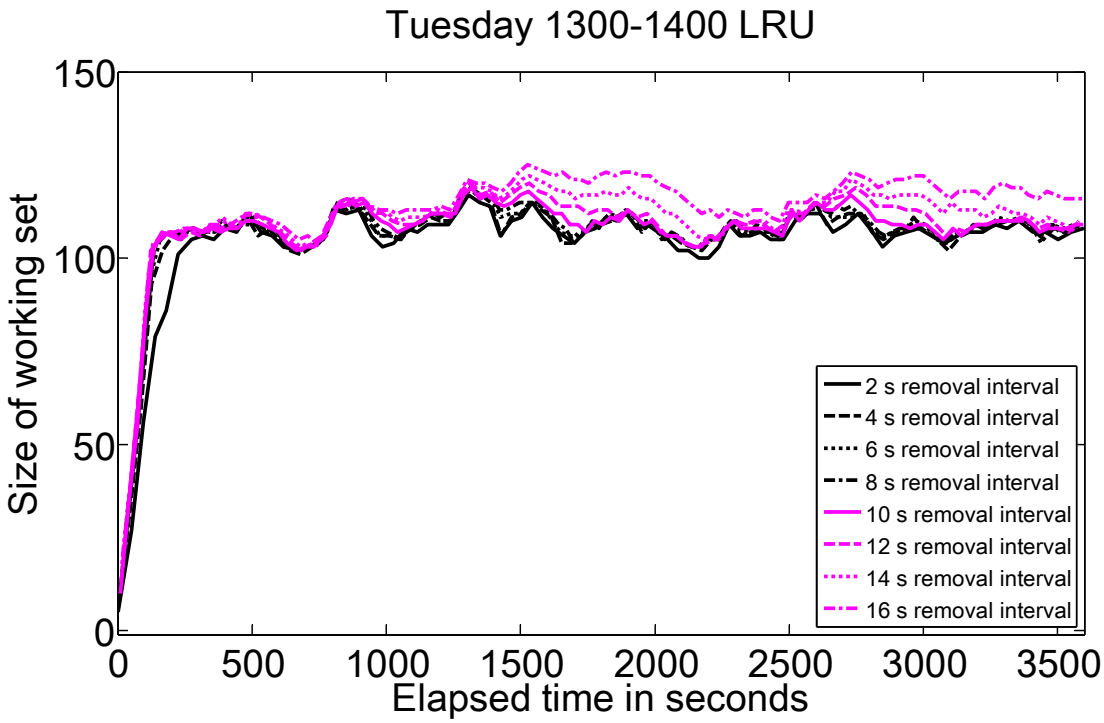

Figure 45: LRU analysis of real 21300 data.

By inspection, the curves in Figure 45 are much closer to those in Figure 44 than those in Figures 38 and 41. The quantitative comparisons are shown below in Table 8.

Table 8: Average, standard deviation, and time to range for Pure LFU synthetic packets, 21300 seed.

Average and Standard Deviation of Working-set Sizes, with Time to Range			
LRU Analysis of Pure LFU Synthetic Packets, 21300 seed			
Removal Interval	Avg	Stdev	TimeToRange
2	105.01	7.546772	44
4	113.6663	7.974507	48
6	116.6962	7.614973	48
8	118.7171	7.629079	48
10	120.1198	7.600735	50
12	121.6923	7.554859	48
14	124.0039	7.546741	56
16	127.1161	7.877121	64
LRU Analysis of TUE 1300-1400 Packets			
Removal Interval	Avg	Stdev	TimeToRange
2	104.8296	13.60995	216
4	106.2313	12.49907	134
6	106.7788	11.95829	126
8	107.3388	11.61129	119
10	107.841	11.86634	123
12	109.0034	11.92606	123
14	110.92	11.80549	132
16	113.7945	12.65953	130

For the synthetic packets, the averages of the working-set sizes fall within a narrow range, with the average for the 16 s removal interval about 21% greater than the average for the 2 s removal interval. In comparison, for the real data, the average for the 16 s removal interval is about 9% greater than the average for the 2 s removal interval. The standard deviations for the synthetic packet working-set sizes are even tighter than those for the real data, and the time to range is shorter; thus, the synthetic packets similarly show properties of locality.

One of our goals with Pure LFU generation was to create synthetic data with packet counts that conformed to a Zipf distribution. The packet count distribution for the synthetic data is shown in Figure 46 below.

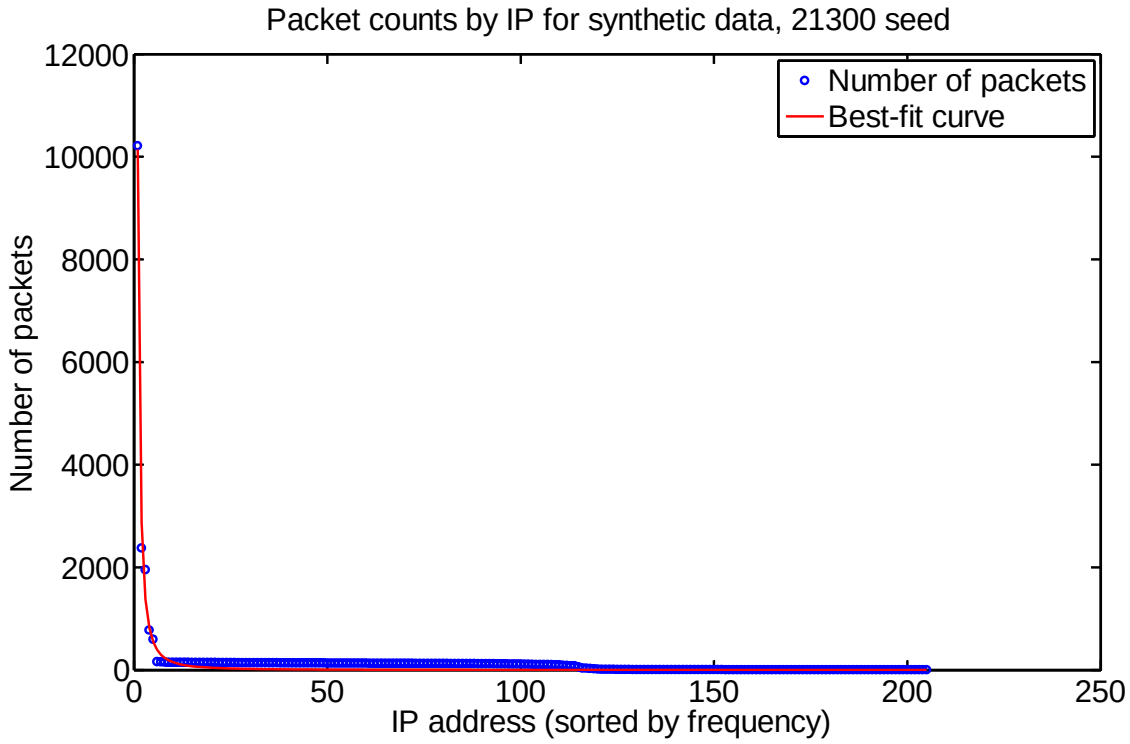

Figure 46: Packet counts by IP for Pure LFU synthetic data, 21300 seed.

The packet counts do conform to a Zipf distribution. To test the influence of different seed data, we repeated the generation process using the entire hour of real data from 31300 (Wednesday 1300-1400), 41300 (Thursday 1300-1400), and 51300 (Friday 1300-1400). The LRU working-set processing results and packet count distributions are shown in Figures 47-52 below.

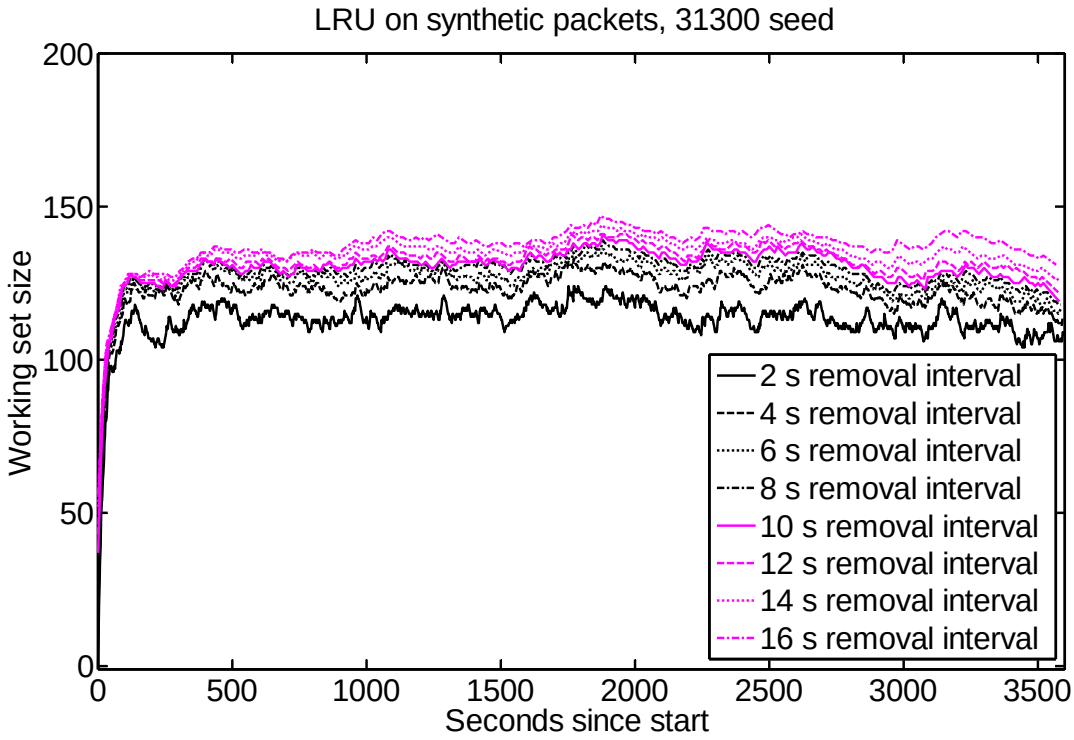

Figure 47: LRU analysis of Pure LFU synthetic packets, 31300 seed.

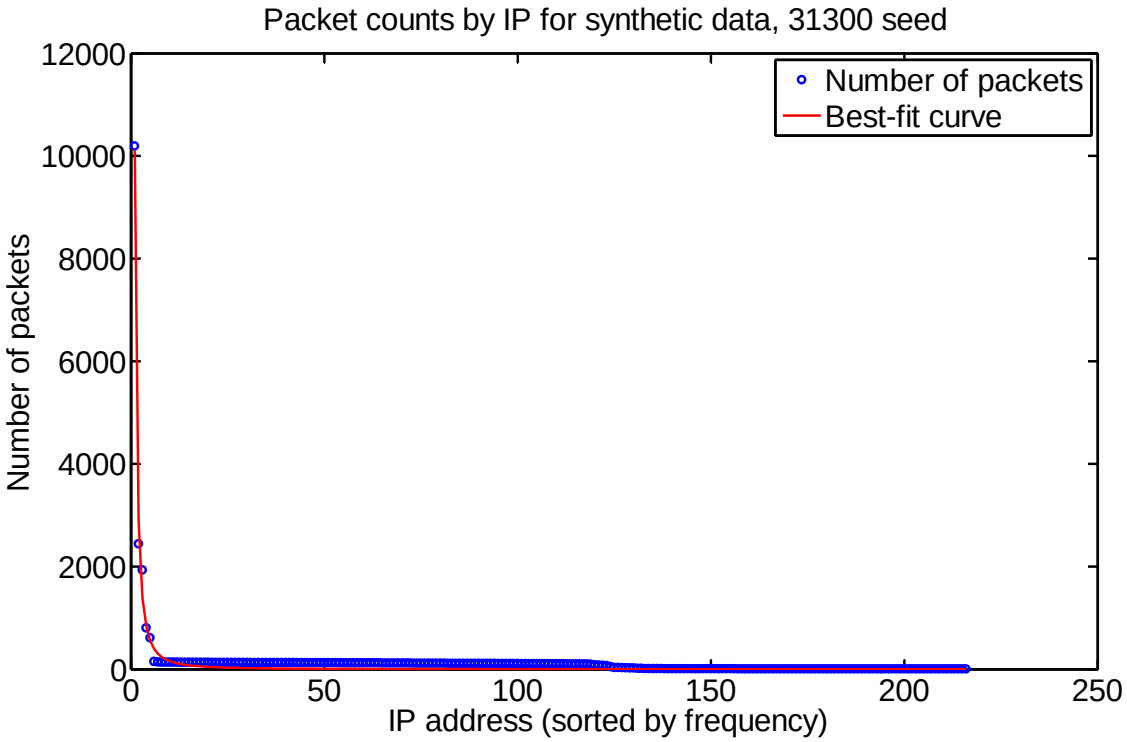

Figure 48: Packet counts by IP for Pure LFU synthetic data, 31300 seed.

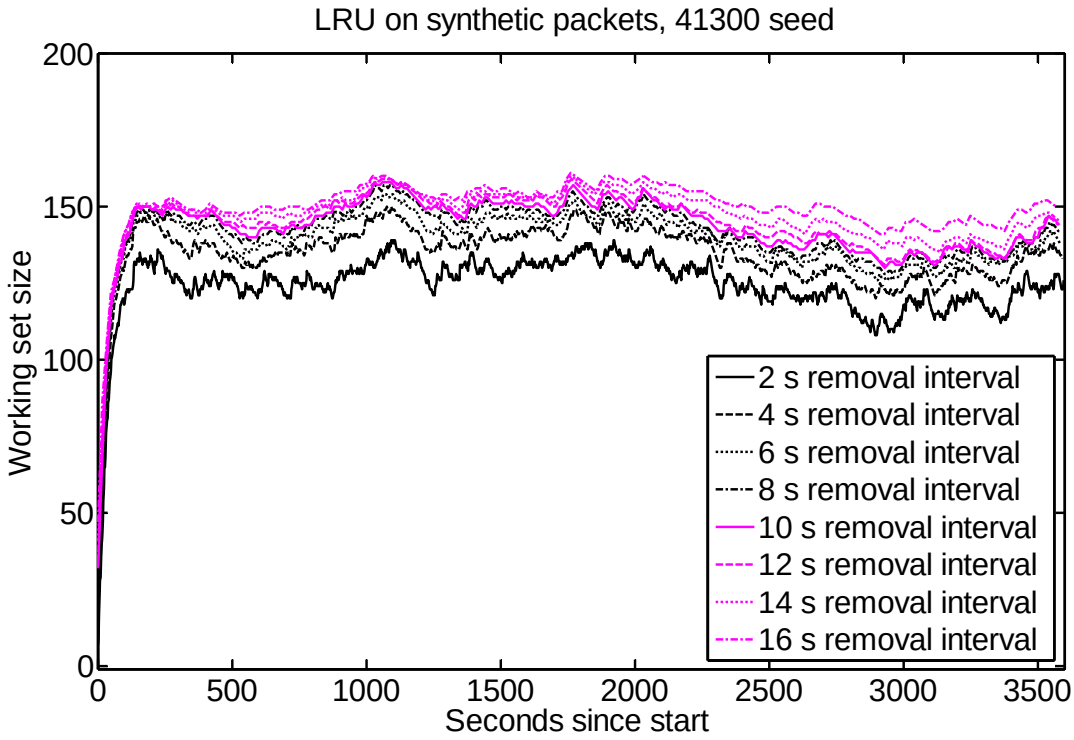

Figure 49: LRU analysis of Pure LFU synthetic packets, 41300 seed.

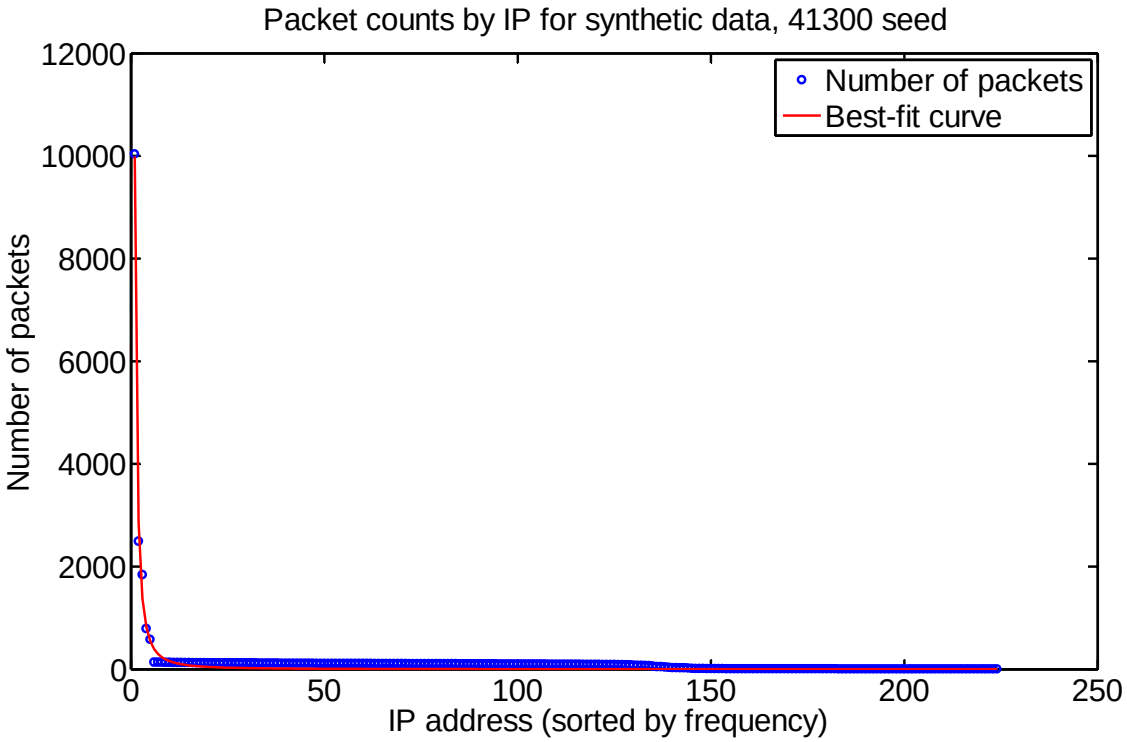

Figure 50: Packet counts by IP for Pure LFU synthetic data, 41300 seed.

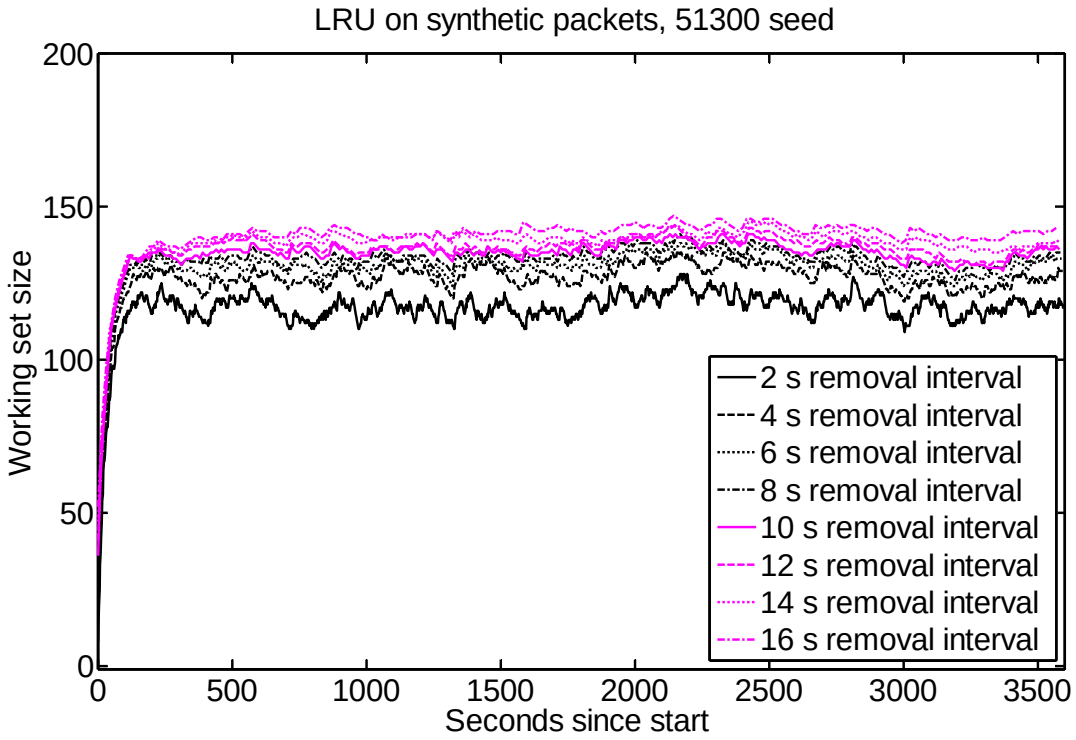

Figure 51: LRU analysis of Pure LFU synthetic packets, 51300 seed.

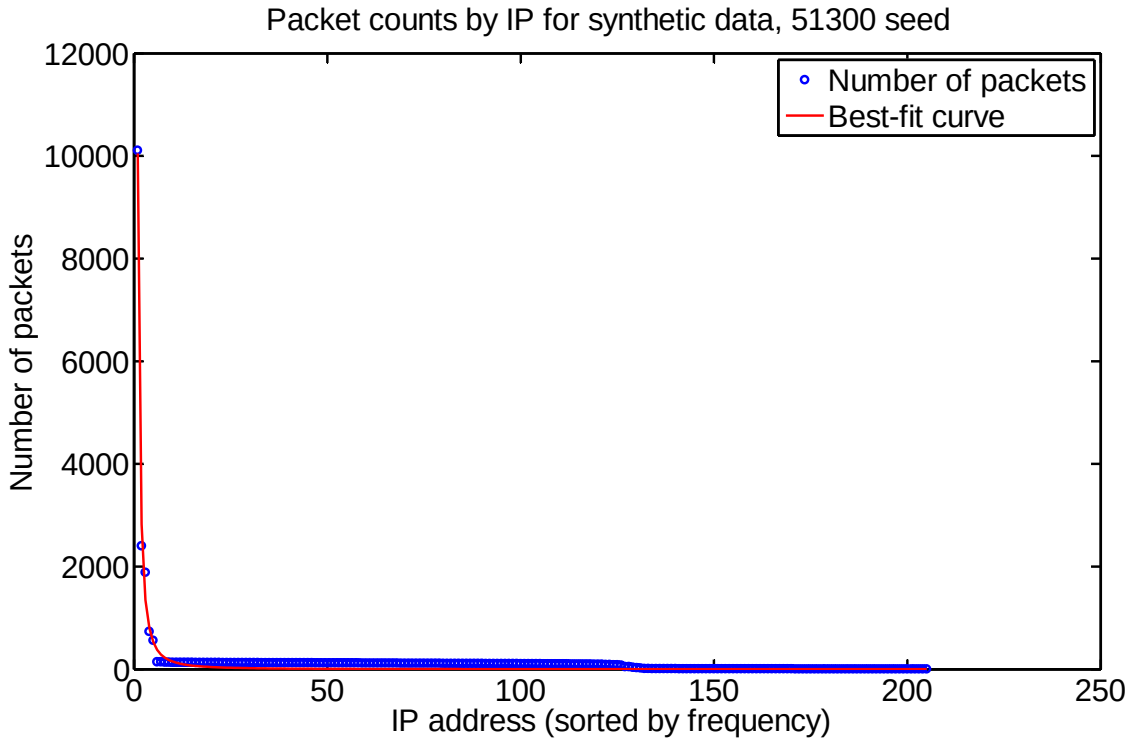

Figure 52: Packet counts by IP for Pure LFU synthetic data, 51300 seed.

The Pure LFU generation method seeded with the real 21300, 31300, 41300, and 51300 data produces synthetic packets that show locality and with packet counts that conform to a Zipf distribution. The parameters for the best-fit curves $f(i) = \frac{c}{i^{\alpha}}$ are given below in Table 9.

Table 9: Parameters for the best-fit curves.

Seed for Pure LFU Synthetic Data	c	α
21300 (Figure 46)	10142.35297	1.81514
31300 (Figure 48)	10128.63036	1.80736
41300 (Figure 50)	9988.57821	1.80542
51300 (Figure 52)	10049.71781	1.82969

The synthetic data shows similar properties when compared to the real data. Will it respond to an attack in the same way? We attempt to answer this question in the next chapter.

CHAPTER 9: DDOS ATTACK ON REPLAYED REAL AND SYNTHETIC TRAFFIC

9.1 Introduction

A distributed denial-of-service (DDoS) attack attempts to overwhelm certain resources of a network server, thus preventing legitimate users from accessing those resources. The "distributed" modifier refers to the use of many different attack machines in different locations. The advantages of using distributed attackers include greater difficulty for the defender in isolating the attack traffic, because it comes from many directions, and higher capacity to send attack packets, because more attackers are involved. We simulated a DDoS attack by inserting attack packets into both the real and synthetic traffic, then processed the resulting packets to see whether our LRU working-set algorithm could detect the attack.

9.2 Generation of DDoS Attack Packets

We chose to simulate a TCP SYN flood attack, which tries to overload a server with half-open TCP connections so that no legitimate TCP connections can be made with the server. The attack consisted of 1 packet sent from each of 1024 different machines. We used a Poisson distribution for the interarrival times of the packets with a mean of 0.1s. Thus, the entire attack would last for about 102.4 seconds. The destination for the attack packets was the IP address of the server from which we collected real data.

After the attack packets were generated, we shifted the timestamps of all the packets so that the first packet would arrive at 1200 seconds into the 21300 (Tuesday 1300-1400) time period. Next, we inserted the attack packets into the appropriate places among the real 21300 data packets based on their timestamps. For comparison, we also inserted the same attack packets among the Pure LFU synthetic packets with 21300 seed, which were discussed in Chapter 8. These two newly-formed sets of packets were now ready to be analyzed.

9.3 DDoS Attack Results

We analyzed the real 21300 data packets both without and with the inserted DDoS attack packets using our LRU working-set algorithm. The results for the dataset without the attack packets are shown in Figure 53 below, and the results for the dataset with the attack packets are shown in Figure 54 below.

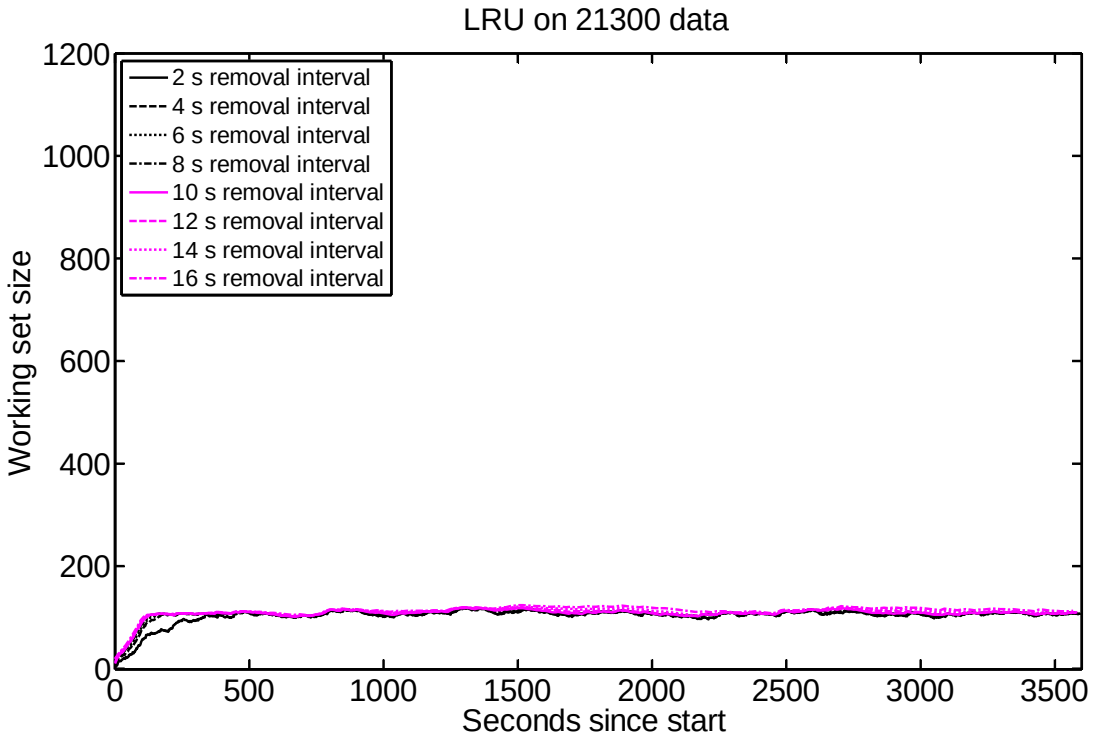

Figure 53: LRU on real 21300 data without DDoS attack packets.

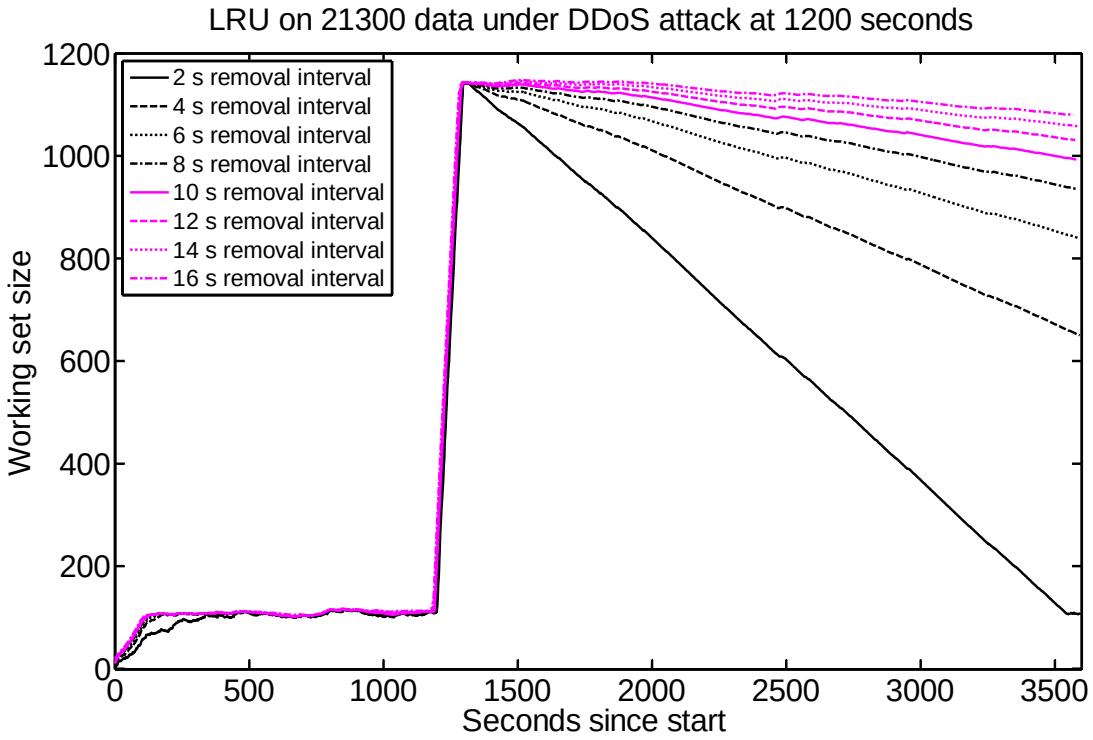

Figure 54: LRU on real 21300 data with DDoS attack packets.

In Figure 54, the attack appears as a spike in the working-set size beginning at 1200 seconds and ending about 102.4 seconds later. The size of the working-sets increase by about 1024, which is the number of machines involved in the attack. After the attack ends, packets are removed at an approximately linear rate based on the removal interval. For the 2 s removal interval, the curve has the steepest downward slope, while for the 16 s removal interval, the curve has the shallowest downward slope.

We also analyzed the Pure LFU synthetic data packets with 21300 seed, which were discussed in Chapter 8, both without and with the inserted DDoS attack packets using our LRU working-set

algorithm. The results for the dataset without the attack packets are shown in Figure 55 below, and the results for the dataset with the attack packets are shown in Figure 56 below.

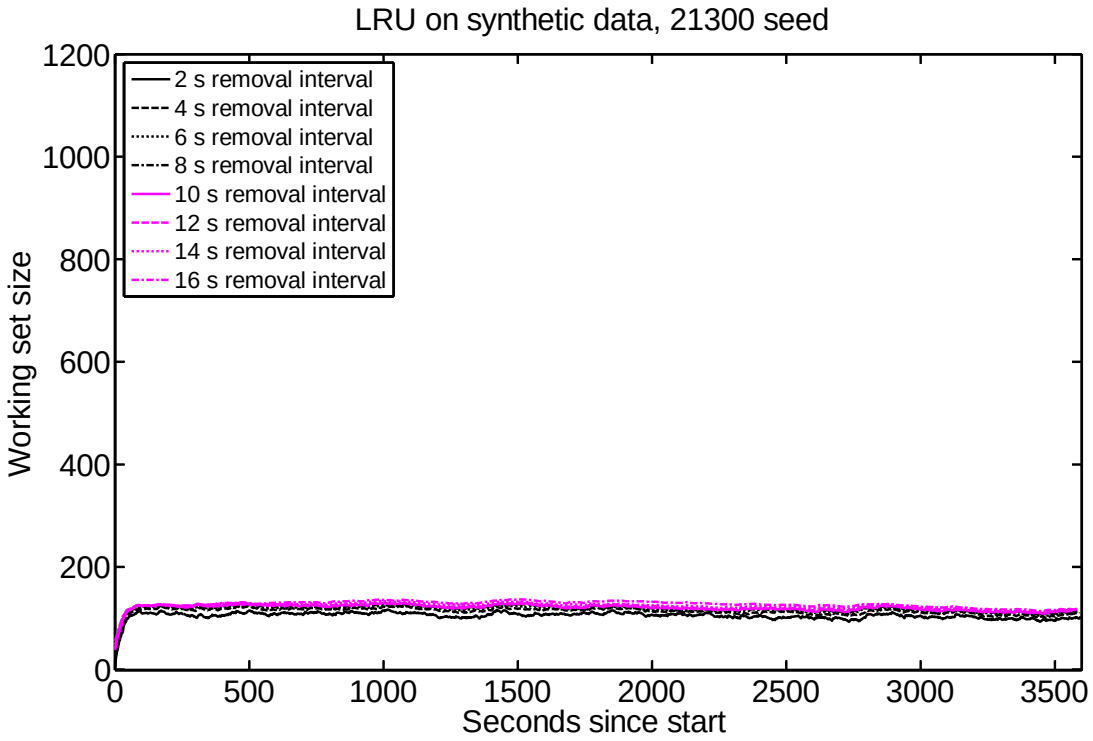

Figure 55: LRU on Pure LFU synthetic data, 21300 seed, without DDoS attack packets.

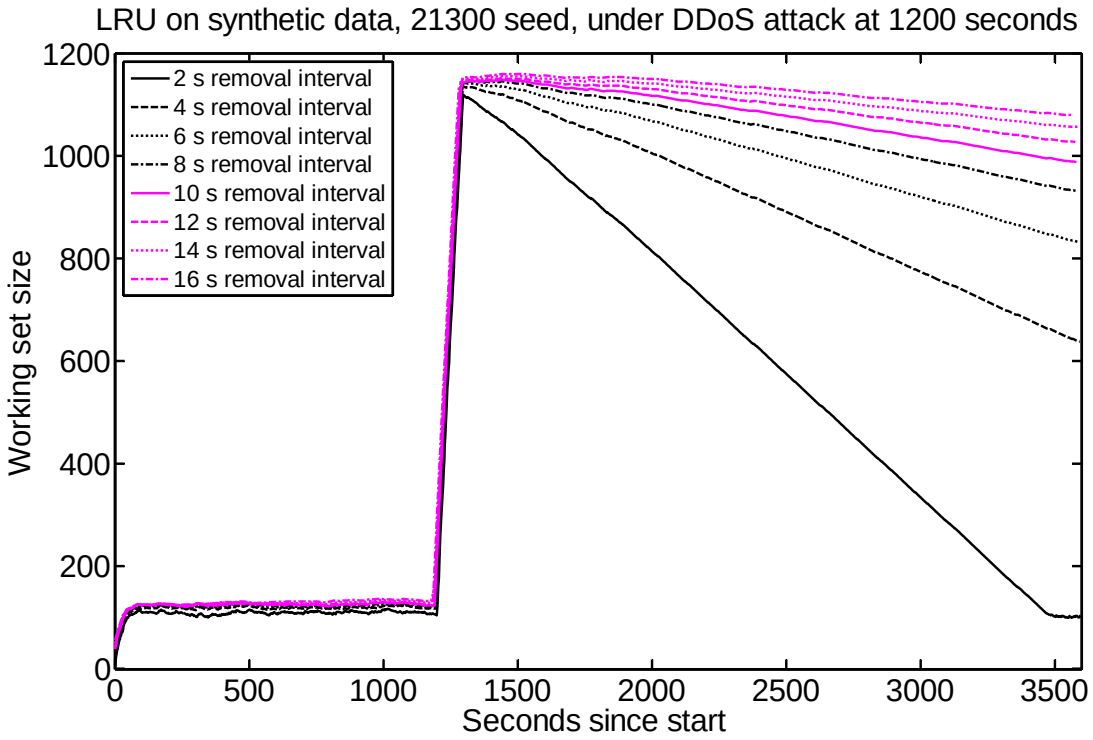

Figure 56: LRU on Pure LFU synthetic data, 21300 seed, with DDoS attack packets.

The results are similar to those for the real 21300 data with DDoS attack packets.

CHAPTER 10: ANALYSIS

10.1 Introduction

In this chapter, we present some theoretical analysis, including the complexity of our LRU working-set algorithm, the application of queueing theory in predicting the equilibrium working-set size, and a theorem regarding the relative sizes of working-sets given different removal intervals.

10.2 Complexity Analysis

The LRU working-set algorithm uses a binary min-heap data structure for the working-set. This structure was chosen to minimize the amount of time required for a removal, which occurs frequently. Because the binary min-heap uses the access timestamp of each connection as the key for placement within the heap, the root of the binary min-heap will always have the minimum timestamp; hence, the connection at the root is always the one to be removed under LRU.

As we have shown, under normal conditions, the working-set size soon reaches a constant, say C. There are two types of events that affect the working-set: the removal of the least recently used connection and the arrival of an incoming packet.

Removal of the least recently used connection is O(1) because the root is always removed. A sift-down is then required to restore the heap property, and this is O(lg(C)). An arriving packet either belongs to a connection already in the working-set, in which case it is an in-packet, or it does not, in which case it is an out-packet. To determine whether a packet is an in-packet or an out-packet requires a linear search of the binary min-heap (which is implemented as an array) for the corresponding source IP address, and this search is O(C). If the packet is an out-packet, its addition to the working-set is O(1), and the subsequent sift-up to restore the heap property is O(lg(C)). If the packet is an in-packet, the update of its timestamp is O(1), and the subsequent sift-down to restore the heap property is O(lg(C)).

10.3 Queueing Theory Model for Working-Set Behavior

The working-set can be viewed as a queue, with connections waiting in the queue until they are served and removed. In particular, the M/D/1 model from queueing theory best fits the working-set. The three components of the model are M, for a Poisson distribution of the incoming connections; D, for a deterministic (fixed-interval) removal; and 1, which means that there is one server processing the queue. There are some limitations to the application of this model. For example, of all the packets arriving at the server, only out-packets count as new arrivals to the queue. In-packets belong to connections already in the working-set, and those connections are already in the queue. Also, it is not clear when and for how long a connection is served, because

it is simply removed from the queue at the moment the removal interval elapses. Nevertheless, we were able to achieve good predictions of the equilibrium working-set size using the model.

For the M/D/1 model, the equilibrium queue size is predicted by the following formula:

$$\rho + \frac{\rho^2}{2(1-\rho)}$$

where $\rho = \lambda / \mu$, λ is the long-term mean arrival rate, and μ is the long-term mean departure rate from the queue. From the formula, we see that the queue size will be large when ρ is close to, but less than 1.

We decided to determine how closely the actual 21300 data matched the model by using the LRU working-set algorithm with varying removal intervals and counting the number of new arrivals (out-packets) and removals after 1000 s had elapsed, at which point the working-set size would have reached equilibrium. These counts gave us the actual values for λ and μ. We then used the actual mean working-set size along with the formula for the equilibrium queue size to solve for the predicted value of ρ, which was then used with the actual value of λ to compute the predicted value of μ. The results are shown in Table 10 below.

Table 10: Comparison of actual and predicted values of μ for 21300 data (LRU).

Removal interval (s)	New arrivals after 1000 s	Removals after 1000 s	Actual λ	Actual μ	Predicted μ
2	1306	1300	0.055529572	0.055274459	0.055795691
4	653	650	0.027764786	0.027637229	0.027896082
6	436	433	0.018538203	0.018410647	0.018625416
8	326	325	0.013861134	0.013818615	0.013926002
10	260	260	0.011054892	0.011054892	0.011106385
12	215	216	0.009141545	0.009184064	0.00918367
14	183	186	0.007780943	0.0079085	0.007816176
16	162	162	0.006888048	0.006888048	0.006918446

From the table above, we can see that the actual and predicted values of μ are very close. In fact, for the 12 s removal interval, the prediction was accurate to three digits. The slight errors in the prediction may have been caused by the choice of 1000 s as the starting point for the count of new arrivals and removals. We are confident that M/D/1 does accurately model the behavior of the LRU working-set as a queue.

10.4 Observations of Out-of-Band Working-Set Behavior

This section will serve partly as an introduction to the working-set subset theorem discussed in Section 10.5 below. Previously, we found that the LRU working-set algorithm produces working-set sizes that reach a stable equilibrium, given removal intervals from 2 s to 16 s. We investigated whether the same would occur given removal intervals outside of that band. Figure

57 below shows the results of processing the 21300 data using the LRU algorithm with removal intervals from 16 s to 60 s.

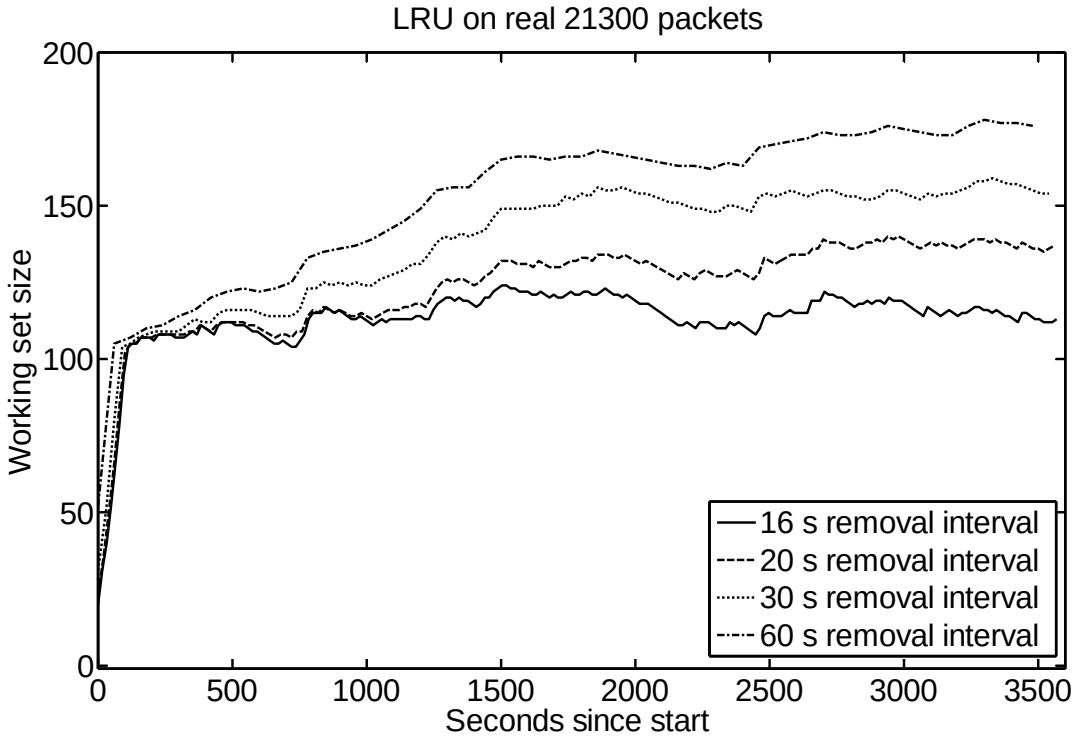

Figure 57: LRU on real 21300 packets, with removal intervals from 16 s to 60 s.

The curves in Figure 57 above show that, for removal intervals above 16 s, the working-set size does not reach a stable equilibrium. Figure 58 below shows the results of processing the 21300 data using the LRU algorithm with removal intervals from 0.6 s to 2 s.

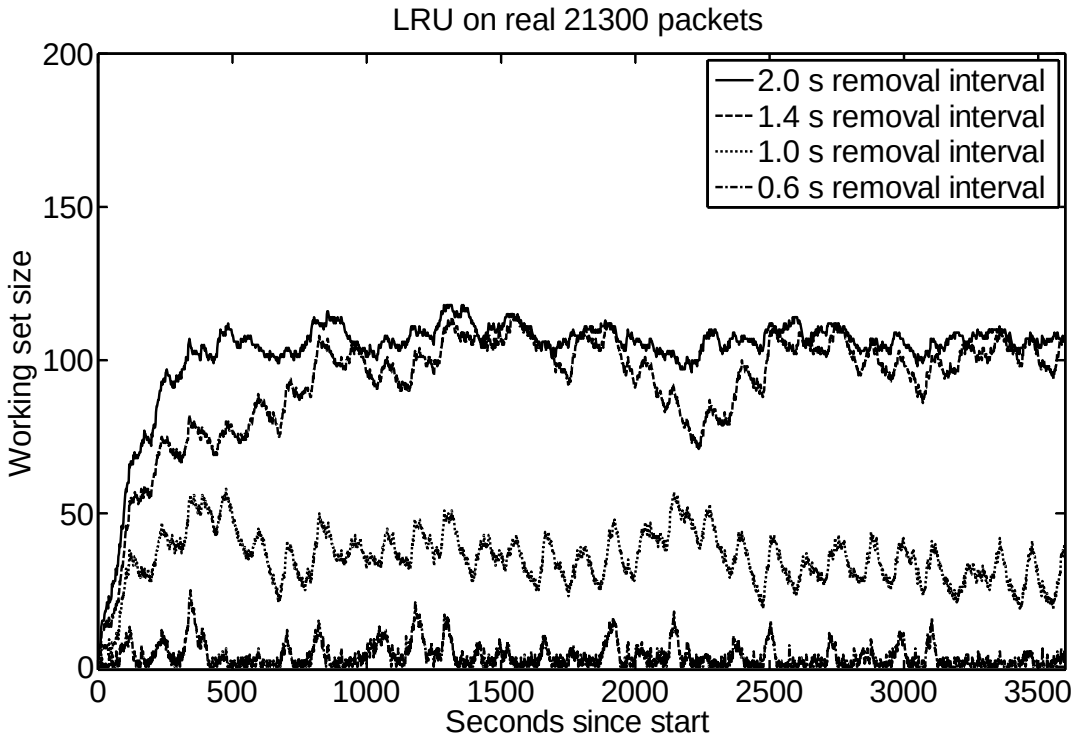

Figure 58: LRU on real 21300 packets, with removal intervals from 0.6 s to 2 s.

The curves in Figure 58 above show that, for removal intervals below 2 s, the working-set size is unstable. Note that, for the 0.6 s removal interval, the working-set size repeatedly reaches zero, and this creates a false sense of stability. The curve for the removal interval of 2 s more clearly reaches a stable equilibrium.

In Figures 57 and 58, we notice that a curve corresponding to a smaller removal interval always remains below a curve corresponding to a larger removal interval. This was the case for earlier curves using LRU as well. The reason why this is true is discussed in the next section.

10.5 Working-Set Subset Theorem

10.5.1 Theorem

Let n be a nonnegative integer and let r_1, r_2 be positive real numbers such that $r_1 \leq r_2$. Each working-set is associated with an algorithm that processes it. Let LRU1 denote the LRU algorithm with removal interval r_1, and let LRU2 denote the LRU algorithm with removal interval r_2.

Let $S_1(n)$ denote the working-set associated with LRU1 after $n \cdot r_1$ time has elapsed since the start of processing. Let $S_2(n)$ denote the working-set associated with LRU2, also after $n \cdot r_1$ (not $n \cdot r_2$) time has elapsed since the start of processing. We claim that $S_1(n) \subseteq S_2(n)$ for all n.

Note that we are measuring time in multiples of r_1. We will assume that, at time $n \cdot r_1$, the n-th removal by LRU1 has been completed; similarly, at time $n \cdot r_2$, the n-th removal by LRU2 has been completed.

10.5.2 Proof

First we will consider the case that $r_1 < r_2$, and handle the case that $r_1 = r_2$ later. We will use induction on n, the number of multiples of r_1 that have elapsed in time since the beginning of processing. We assume that packets are processed one at a time in the order in which they arrive.

We have $S_1(0) = S_2(0) = \varnothing$ because no time has elapsed, so no packets have entered the working-set; thus, $S_1(0) \subseteq S_2(0)$.

Assume for some integer $n \geq 0$ that $S_1(n) \subseteq S_2(n)$. We will show that $S_1(n+1) \subseteq S_2(n+1)$.

There are only two events that can change the contents of a working-set: the removal of a connection already in the working-set and the arrival of a packet belonging to a connection not already in the working-set.

We will first consider arrivals. Let p be the first packet that arrives after time $n \cdot r_1$. At this time, $S_1(n) \subseteq S_2(n)$ is still true. We will refer to the property that the working-set for LRU1 is a subset of the working-set for LRU2 as the "subset property". There are three cases.

Case 1: Packet p belongs to a connection, say c_p, in $S_1(n)$ (and hence that connection is also in $S_2(n)$). This is a trivial case, because LRU1 and LRU2 will simply update the timestamps for c_p both in $S_1(n)$ and in $S_2(n)$. The connections in the two sets will not change. Thus, the subset property is preserved.

Case 2: Packet p belongs to a connection, say c_p, in $S_2(n)$, but does not belong to any connection in $S_1(n)$. LRU1 will add c_p to its working-set to form the set $S_1(n) \cup \{c_p\}$. LRU2 will not change

the connections in its working-set. Because $S_1(n) \cup \{c_p\} \subseteq S_2(n)$, the subset property is preserved.

Case 3: Packet p does not belong to any connection in $S_2(n)$. LRU1 will add the connection to which p belongs, say c_p, to its working-set to form the set $S_1(n) \cup \{c_p\}$ and LRU2 will add c_p to its working-set to form the set $S_2(n) \cup \{c_p\}$. Because $S_1(n) \cup \{c_p\} \subseteq S_2(n) \cup \{c_p\}$, the subset property is preserved.

The subset property continues to hold if further packets arrive, so long as a removal does not occur.

Now we look at removals. Notice that $r_1 < r_2$ implies that, by the Pigeonhole Principle, there exists at most one multiple of r_2, say $t \cdot r_2$ where t is a nonnegative integer, such that $n \cdot r_1 \le t \cdot r_2 \le (n+1) \cdot r_1$. We will first assume that $n \cdot r_1 < t \cdot r_2 < (n+1) \cdot r_1$ and consider the other case later. See Figure 59 below for one possible timeline of the working-set relationships.

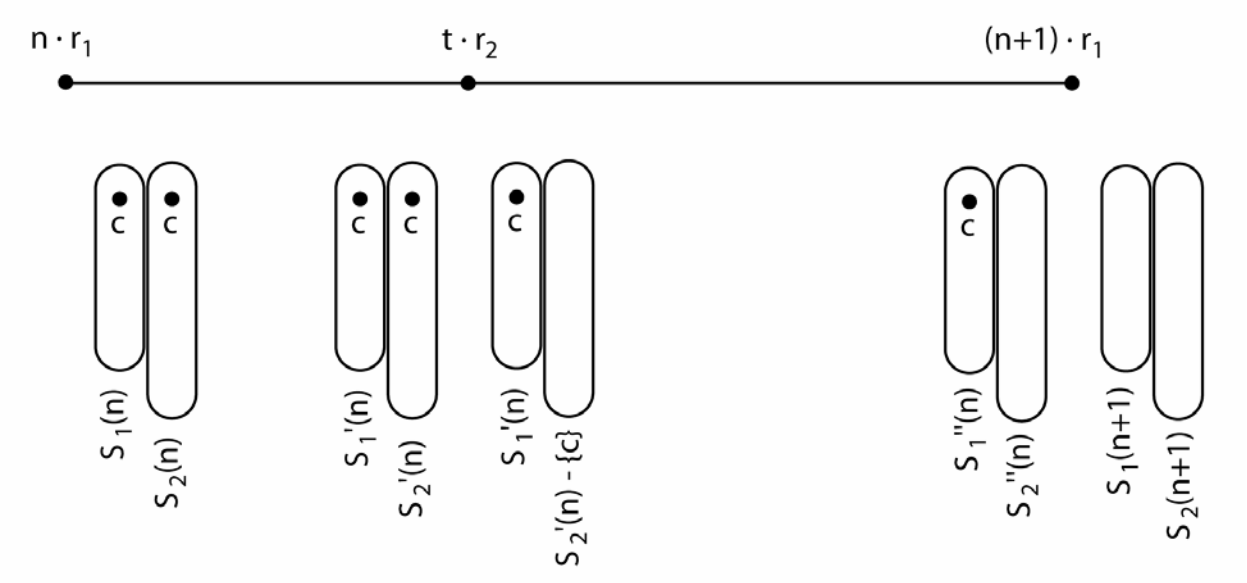

Figure 59: One possible timeline for the working-set relationships.

Immediately before the removal at $t \cdot r_2$, let the working-set for LRU1 be denoted by $S_1'(n)$, and let the working-set for LRU2 be denoted by $S_2'(n)$. If any packets arrive between $n \cdot r_1$ and $t \cdot r_2$, we have shown that the subset property still holds. Thus, $S_1'(n) \subseteq S_2'(n)$.

Let connection c be the one removed from $S_2'(n)$ by LRU2 immediately before time $t \cdot r_2$. We know that c is the least recently used connection in $S_2'(n)$.

Case 1: If $c \notin S_1'(n)$, then $S_1'(n) \subseteq S_2'(n) - \{c\}$, and the subset property is still preserved. If any packets arrive between $t \cdot r_2$ and $(n+1) \cdot r_1$, we have shown that the subset property will continue to be preserved. Immediately before $(n+1) \cdot r_1$, LRU1 will remove at most one connection from

its working-set (it cannot remove any connections if the working-set is empty), and any such removal will not affect the subset property. Thus, $S_1(n+1) \subseteq S_2(n+1)$, as desired.

Case 2: If $c \in S_1'(n)$, then the fact that c is the least recently used connection in $S_2'(n)$ together with $S_1'(n) \subseteq S_2'(n)$ imply that c must be the least recently used connection in $S_1'(n)$. (Note that $S_1'(n)$ is unchanged by the removal that occurred immediately before $t \cdot r_2$, because that was a removal by LRU2.) Now we consider any packets that arrive between $t \cdot r_2$ and $(n+1) \cdot r_1$. Extending our previous notation, immediately before the removal at $(n+1) \cdot r_1$, let the working-set for LRU1 be denoted by $S_1''(n)$ and let the working-set for LRU2 be denoted by $S_2''(n)$.

Notice that c is the only connection in $S_1'(n)$ that is not in $S_2'(n) - \{c\}$, because $S_1'(n) \subseteq S_2'(n)$. We will call this the "almost-subset property", and we claim that this is preserved under any packet arrivals that occur between $t \cdot r_2$ and $(n+1) \cdot r_1$ except for a packet belonging to c. (If a packet belonging to c arrives between $t \cdot r_2$ and $(n+1) \cdot r_1$, then c will be added back to the working-set for LRU2, and the working-set for LRU1 will again be a subset of the working-set for LRU2. We already know that this subset property will be preserved until the next removal, which occurs immediately before $(n+1) \cdot r_1$, and because that removal cannot affect the subset property, we are done.) Let p be the first packet that arrives after $t \cdot r_2$ and does not belong to connection c. There are three cases.

Case 2a: Packet p belongs to a connection, say c_p, in $S_1'(n)$ (and hence that connection is also in $S_2'(n) - \{c\}$, due to the almost-subset property). This is a trivial case, because LRU1 and LRU2 will simply update the timestamps for c_p both in $S_1'(n)$ and in $S_2'(n) - \{c\}$. The connections in the two sets will not change. Thus, the almost-subset property is preserved.

Case 2b: Packet p belongs to a connection, say c_p, in $S_2'(n) - \{c\}$, but does not belong to any connection in $S_1'(n)$. LRU1 will add c_p to its working-set to form the set $S_1'(n) \cup \{c_p\}$. LRU2 will not change the connections in its working-set. Because c remains the only connection in $S_1'(n) \cup \{c_p\}$ that is not also in $S_2'(n) - \{c\}$, the almost-subset property is preserved.

Case 2c: Packet p does not belong to any connection in $S_2'(n) - \{c\}$. LRU1 will add the connection to which p belongs, say c_p, to its working-set to form the set $S_1'(n) \cup \{c_p\}$ and LRU2 will add c_p to its working-set to form the set $(S_2'(n) - \{c\}) \cup \{c_p\}$. Because c remains the only connection in $S_1'(n) \cup \{c_p\}$ that is not also in $(S_2'(n) - \{c\}) \cup \{c_p\}$, the almost-subset property is preserved.

The almost-subset property continues to hold if further packets arrive that do not belong to c, so long as a removal does not occur. Thus, the only connection in $S_1''(n)$ that is not also in $S_2''(n)$ is c.

Therefore, at its next removal (immediately before $(n+1) \cdot r_1$), LRU1 will remove c from $S_1''(n)$. (Note that $S_2''(n)$ is unchanged by the removal that occurs immediately before $(n+1) \cdot r_1$, because that is a removal by LRU1.) Hence $S_1(n+1) = S_1''(n) - \{c\} \subseteq S_2''(n) = S_2(n+1)$, as desired.

What if $n \cdot r_1 = t \cdot r_2$ or $t \cdot r_2 = (n+1) \cdot r_1$? Because the removals by LRU1 and LRU2 occur at the same time, it is not possible for c to be removed from the working-set for LRU2 without simultaneously being removed from the working-set for LRU1, because if c is present in both working-sets, it must be the least recently used connection in both. Therefore, the subset property is preserved. For a similar reason, if $r_1 = r_2$, the subset property always holds. □

10.5.3 Corollary 1

Assume that $r_1 < r_2$. The only situations in which the working-set for LRU1 is not a subset of the working-set for LRU2 manifest in the following manner. First, both working-sets are empty. A packet belonging to connection c arrives. Because both working-sets are empty, connection c is added to both. Packets belonging to connections other than c may arrive without affecting the status of c as the least recently used connection. Now a removal for LRU2 occurs, and c is removed from the working-set for LRU2. After this removal and before the next removal for LRU1, the working-set for LRU1 will contain c, while the working-set for LRU2 will not; furthermore, packets belonging to connections other than c may arrive during this period without

affecting the non-subset relationship. (Note that it is not a problem if a removal for LRU1 occurs while both working-sets are empty, because nothing will be removed.)

10.5.4 Corollary 2

Assume that $r_1 < r_2$. The set difference between the working-set for LRU2 and the working-set for LRU1 consists precisely of those connections that remain in the working-set for LRU2 but have already been removed from LRU1, minus any connection c for which a packet has arrived since c was removed from LRU1. (The arriving packet restores c to the working-set for LRU1, so c is no longer part of the set difference.)

CHAPTER 11: CONCLUSION

11.1 Summary

We have demonstrated that traffic incoming to a typical network server does exhibit properties of locality. Due to these properties, we can build a normal profile for the server, and violations of that profile indicate that the server may be under attack. Our mechanism for detection of locality involves a working-set algorithm. Specifically, there is a working-set of connections to the server, which can be identified by their IP addresses. For each connection in the working-set, there is an associated working-set of ports used by that connection. By attacking the server using standard attacks that utilize many ports in a short period of time, we showed that our algorithm can detect violations of locality using the ports working-set. The algorithm performs efficiently enough to be deployed in real time to detect attacks. We compared various removal methods for the working-set algorithm, including LRU (least recently used), FIFO (first in first out), LFU (least frequently used), and Pure LFU (a variant of LFU), and concluded that LRU performed best in terms of detecting locality.

To understand better why locality existed as a property of the server traffic, we proceeded to build a model of that traffic. The models used a Poisson distribution for the interarrival times between the packets and the Mersenne Twister pseudorandom number generator to create the various random aspects of the data. Our first model was based on a LRU working-set with

probabilistic determination of the incoming IP address, and our second model was based on achieving a Zipf distribution of the packet counts. Both of these models generated packets that produced stable working-sets under analysis by the LRU working-set algorithm, but we felt we could achieve a result closer to the real data by combining aspects of the two models. Our most successful model used a Pure LFU working-set with probabilistic determination of the incoming IP address that was specifically targeted towards achieving a Zipf distribution of the packet counts. The probabilistic determination was simple: there were 2 primary categories of whether a packet should be in or out of the current working-set, and within the in-working-set category, 6 sub-categories of which connection should be chosen. This model generated packets that behaved similarly to the real packets under analysis by the LRU working-set algorithm. Our model for network packets can easily be used for further studies of locality, because there are very few parameters and the effects of changing these parameters can be readily observed.

Because we had not yet explicitly tested the connection working-set aspect of the LRU working-set algorithm for its ability to detect violations of locality, we decided to insert DDoS (distributed denial-of-service) attack packets into both the real data and into synthetic data, then subject the altered packets to analysis. We found that the LRU working-set algorithm was able to detect the violation of locality caused by the DDoS attack, and furthermore, both the real data and the synthetic data with inserted DDoS packets behaved in a similar manner under analysis.

We presented some theoretical analysis. The complexity of our LRU working-set algorithm is such that real-time processing of network traffic is feasible. Applying the M/D/1 model from

queueing theory does accurately predict the equilibrium working-set size. Finally, we proved a theorem that explains why the curve for working-set size under the LRU algorithm using a smaller removal interval always appears to be lower than the curve for the same using a larger removal interval.

11.2 Limitations and Future Work

One requirement for successful use of our working-set algorithm in intrusion detection is a training period during which normal behavior is observed and thresholds are set for detection of abnormal behavior. It may be necessary to alter these thresholds if there is a major change in the network, for example, in its topography or in the number of its users.

We would like to study further the effects of a distributed attack on a server. One way to do this would be to set up a network of computers and have them attack a single server on which our intrusion detection system has been installed. We expect that the results would be similar to those in our simulated DDoS attack, but unpredictable events can occur in a real setting.

REFERENCES

1. Zhou, M., Lee, R., Lang, S.-D.: Locality-Based Profile Analysis for Secondary Intrusion Detection. In: Proc. 8th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 166-173. IEEE Computer Society Press, Washington, DC (2005)
2. McHugh, J., Gates, C.: Locality: a New Paradigm for Thinking About Normal Behavior and Outsider Threat. In: ACM New Security Paradigms Workshop. ACM Press, New York (2004)
3. Berk V., Bakos, G.: Designing a Framework for Active Worm Detection on Global Networks. In: Proc. 1st IEEE International Workshop on Information Assurance. IEEE Computer Society Press, Washington, DC (2003)
4. Hofmeyr, S.: An Immunological Model of Distributed Detection and Its Application to Computer Science Security. Ph.D. dissertation, Dept. of Computer Science, Univ. of New Mexico (1999)
5. Williamson, M.: Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. In: 18th Annual Computer Security Applications Conference. IEEE Computer Society Press, Washington, DC (2002)
6. Cui, W., Katz, R. H., Tan, W.: BINDER: An extrusion-based break-in detector for personal computers. Technical report, Hewlett-Packard Laboratories, Palo Alto, CA (2004)

7. Sekar, V., Xie, Y., Reiter, M. K., Zhang, H.: A multi-resolution approach for worm detection and containment. In: Proc. International Conference on Dependable Systems and Networks, pp. 189-198. IEEE Computer Society Press, Washington, DC (2006)
8. Sekar, V., Xie, Y., Reiter, M. K., Zhang, H.: Is host-based anomaly detection + temporal correlation = worm causality? Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (2007)
9. Wireshark Network Protocol Analyzer, http://www.wireshark.org/
10. Nmap Network Security Scanner, http://nmap.org/
11. Smart, M., Malan, G. R., Jahanian, F.: Defeating TCP/IP Stack Fingerprinting. In: Proc. 9th USENIX Security Symposium. (2000)
12. Gil, T. M., Poletto, M. MULTOPS: A data-structure for bandwidth attack detection. In: Proc. 10th USENIX Security Symposium. (2001)
13. Cheng, C., Kung, H. T., Tan, K. Use of Spectral Analysis in Defense Against DOS Attacks. In: Proc. IEEE Global Telecommunications Conference. (2002)
14. Feinstein, L., Schnackenberg, D., Balupari, R., Kindred, D. Statistical Approaches to DDoS Attack Detection and Response. In: Proc. DARPA Information Survivability Conference and Exposition. (2003)
15. Wang, H., Zhang, D., Shin, K. G. Detecting SYN Flooding Attacks. In: Proc. IEEE INFOCOM. (2002)
16. Lemon, J. Resisting SYN Flood DOS Attacks with a SYN Cache. In: Proc. USENIX BSDCon. (2002)

17. Adamic, L. A., Huberman, B. A. Zipf's Law and the Internet. In: Glottometrics 3, pp. 143-150. (2002)

18. Breslau, L., Cue, P., Cao, P., Fan, L., Phillips, G., Shenker, S. Web Caching and Zipf-like Distributions: Evidence and Implications. In: Proc. IEEE INFOCOM. (1999)

19. Cooper, R. B. Introduction to Queueing Theory. North Holland (1981)

20. Matsumoto, M., Nishimura, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. In: ACM Transactions on Modeling and Computer Simulation. (1998)