TOWARDS MORE EFFICIENT DELAY MEASUREMENTS ON THE INTERNET

A Thesis

by

PATRICK JORDAN WEBSTER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Dmitri Loguinov
Committee Members,	Riccardo Bettati
	A. L. Narasimha Reddy
Head of Department,	Hank Walker

August 2013

Major Subject: Computer Science

ABSTRACT

As more applications rely on distributed systems (peer-to-peer services, content distribution networks, cloud services), it becomes necessary to identify hosts that return content to the user with minimal delay. A large scale map of delays would aid in solving this problem. Existing methods, which deploy devices to every region of the Internet or use of a single vantage point have yet to create such a map. While services such as PlanetLab offer a distributed network for measurements, they only cover 0.3% of the Internet. The focus of our research is to increase the speed of the single vantage point approach so that it becomes a feasible solution.

We evaluate the feasibility of performing large scale measurements by performing an experiment using more hosts than any previous study. First, an efficient scanning algorithm is developed to perform the measurement scan. We then find that a custom Windows network driver is required to overcome bottlenecks in the operating system. After developing a custom driver, we perform a measurement scan larger than any previous study. Analysis of the results reveals previously unidentified drawbacks to the existing architectures and measurement methodologies. We propose novel methods for increasing the speed of experiments, improving the accuracy of measurement results, and reducing the amount of traffic generated by the scan. Finally, we present architectures for performing an Internet scale measurement scan.

We found that with custom drivers, the Windows operating system is a capable platform for performing large scale measurements. Scan results showed that in the eleven years since the original measurement technique was developed, the response patterns it relied upon had changed from what was expected. With our suggested improvements to the measurement algorithm and proposed scanning architectures, it may be possible to perform Internet scale measurement studies in the future.

To my family

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Dmitri Loguinov, for introducing me to the interesting field of networking. I am thankful for the guidance and assistance that he has offered throughout the course of this thesis. He has helped me hone my analytical skills, research methodologies, and demonstrated the need to keep an open mind when approaching a problem.

I would also like to thank Drs. Riccardo Bettati and Narasimha Reddy for their time serving on my committee and their insightful questions concerning my research. I extend my appreciation to Mr. Willis Marti and his staff at the Networking and Information Security department, as well as Brad Goodman and the Computer Science Group for their patience and cooperation while performing my experiments.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The idea of an Internet map has been around for quite some time [15, 32, 25, 9, 27] due to the possible benefits from knowing the delays between arbitrary hosts. This is especially true for peer-to-peer services, content distribution networks, and cloud services where it is desired to send content from a host closest to the end user. If a delay map were available, it could be used as lookup table to easily find the nearest host. Additionally, if the map can be generated quickly enough and frequently enough, the information can be used to find routing problems. If links become congested or go down entirely, the delays would become apparent in the scan results.

The time delay between arbitrary hosts is challenging to obtain since one cannot just ask two hosts to measure the delay between themselves. Previous estimates have been performed by deploying beacons or tracers [15] throughout the network (i.e., using PlanetLab [36]) and measuring the delay between those. However, it is possible to make use of the DNS protocol to perform measurements between arbitrary DNS servers. This technique was devised in 2002 [17], but has yet to be performed at an Internet scale (the largest previous study used a 5000 x 5000 matrix [45]). The primary focus of current research around delay measurements has been about making it more accurate [26, 28, 23], and not how to perform the scans faster.

In order to perform an Internet wide delay measurement (475k x 475k), the scanner will need to be able to operate at a high rate of speed and perform measurements without negatively impacting the hosts involved. The default networking stacks available on most operating systems are not typically designed for sending a large number

of packets at a high rate. This problem has been addressed on the Linux operating systems by the availability of custom networking drivers [11, 12, 13, 20, 1], but none are currently available for the Windows operating system.

Our focus in this work is to evaluate the feasibility of performing large scale measurement scans. We do this by developing a scanning algorithm which can perform measurements efficiently, with minimal memory requirements, and without negatively impacting the systems involved. A custom, high speed, networking driver has been written to allow for the usage of the Windows operating system during the study. The scanning algorithm is then deployed using 20,000 hosts to evaluate its capability as well as the capability of the measurement method. Deficiencies in the measurement method and unexpected characteristics of the experiment's workload are also discussed. Finally, architectures are proposed for performing a measurement scan at an Internet scale.

CHAPTER II

RELATED WORK

2.1 Delay Measurements via DNS

Previously, latency measurements required access to specialized nodes placed throughout the desired network [15] or with the assistance of volunteers running special software [27]. While those methods focused on using multiple vantage points to perform the measurements, the usage of a single vantage point was also studied. A technique was developed which allowed for the use of existing open, recursive DNS resolvers to perform measurements.

2.1.1 King

The King project [17] developed a method for measuring latency using open, recursive DNS resolvers, based on the assumption that the name servers are typically close to the end host. The advantage of King is that it does not require the deployment of additional infrastructure and has a significantly larger set of nodes that can be used for measurement. As of April 7, 2013, there were 25 million open recursive resolvers [33], compared to PlanetLab's 1,200 nodes [36]. Developed in 2002, the King method is widely used as a source for obtaining accurate measurements [35, 44, 2, 21, 18, 4, 5, 9, 16, 7].

2.1.2 Turbo King

The King project was improved in terms of accuracy and efficiency by Turbo King [23]. Originally, King required the use of ten DNS queries to compute the latency;

Turbo King reduced this to six. Turbo King also addressed inaccuracies caused by the existence of DNS forwarders and multiple name servers. It also drastically reduced the amount of cache pollution caused by the tests.

2.2 Drivers

Developing high speed network stacks is not a new problem and has been addressed several times on Linux operating systems [11, 12, 13, 20, 1] or through the creation of custom network cards [38]. There has been very little [43, 40] work done on Windows operating systems for achieving high throughput networking without the aid of custom hardware. Cross-platform capture cards do exist that support gigabit injection rates, but they are proprietary and cost several thousand dollars [38].

2.2.1 Intel DPDK

Intel's Data Plane Development Kit (DPDK) provides a framework for developing fast packet processing applications. It does not provide a full network stack, but the libraries necessary to perform high speed packet processing in user space. The DPDK was benchmarked by Intel at 10Mpps per core with 64 byte packets. The DPDK is limited to Linux operating systems using Intel network cards. 6WIND makes use of this library for their Intel specific integration.

2.2.2 PF_RING

A common library used in Linux for high speed networking is PF_RING, developed by ntop [12]. PF_RING previously required directly patching the Linux kernel, but newer versions are now shipped as kernel modules. The main benefit of PF_RING is that packets are stored in a circular buffer instead of constantly allocating and

de-allocating memory for each packet. In the standard operation mode of the driver, PF_RING polls the network device through the Linux network interface (NAPI). NAPI copies the packets into the circular buffer and the user application reads the packets out of the circular buffer.

2.2.2.1 PF_RING DNA

In version 4.7 of PF_RING, ntop introduced the DNA module, allowing Direct NIC Access. The goal of this module is to remove the NAPI polling to lower CPU usage and reach 10 gigabit speeds for capture and injection. This is accomplished by directly mapping the NIC memory and registers into user space so that only one memory copy takes place; from the NIC to the user. The drawback to this approach is that it is bound to very specific network devices, namely Intel's 1 and 10 gigabit cards.

2.2.3 6WINDGate

The 6WIND organization offers a commercial solution for developing large scale network infrastructures, primarily focused on software defined networking. The 6WINDGate product is a collection of driver modules and software that run partially in isolation from the Linux operating system by reserving several cores specifically for networking tasks. The cores are further split into processing control paths and data paths. The data path cores provide optimized processing modules for certain protocol types and operate completely outside of the Linux operating system. The control path cores interact directly with the Linux networking stack and share processing tasks with the operating system. Similar to PF_RING, this requires optimization for specific network cards. However, due to their method of interacting with the operating system, this solution only works with specific Linux distributions.

2.2.4 WinPcap

On Windows, the only alternative socket library is WinPcap [43], which was largely designed for packet capture, but does offer packet transmission. WinPcap is implemented as an NDIS filter driver and avoids the problems associated with traversing through Winsock. However, the driver allows the packets to pass through, with no means to drop or redirect packets. This means that the receive path will suffer from the CPU overhead of Windows processing the packets. As seen in [40], while WinPcap's multi-destination performance is somewhat better than Winsock's, it still requires 75% CPU to reach that rate.

2.2.5 IRLstack

Our research lab has previously worked on a driver for Windows that has improved performance over Winsock and WinPcap [40]. This driver was designed and tested on Windows 2008 SP2, but no longer works properly on Windows 2008 R2 SP1 or Windows 8. Furthermore, the receive side performance is degraded when passing packets from the kernel level to user space. This can result in packets being dropped when running at high rates. In order for this driver to be suitable for incorporation into other research projects, it would need to have its design re-evaluated and corrected.

CHAPTER III

IRLSTACK 2.0

The primary effort in performing delay measurements has typically been focused on improving the accuracy of the results. While this is important, it is also desirable to rapidly perform large scale measurements to obtain snapshots of the Internet as a whole. In order to do that, the tools used to perform the measurements need to be efficient. Improvements have been made for Linux users, but improvements for the Windows operating system have been lacking.

3.1 Motivation

One of the major hurdles to performing an Internet wide delay measurement is that most operating systems' network stacks are not capable of performing well with a large number of connections. Previous work has modified the Linux kernel's network drivers directly to work around these problems. Windows, however, is essentially a black box since the operating system's source code is not available.

Windows provides a networking API called Windows Sockets (Winsock) for application level users to easily interact with network devices. The API abstracts away the inner workings of the underlying protocols, such as TCP/IP, and allows users to easily send and receive data. Over the years, the Windows network stack has evolved to offer more capabilities, such as a closely integrated firewall, network discovery, transport providers, and more. As a result, the Winsock API now sits several layers above the kernel level networking API. By passing data through so many layers, a delay is incurred between the time when a packet arrives and when it is sent to the user.

This delay is typically not noticed in most situations, but it becomes a bottleneck when an application is designed to communicate with a large number of destinations.

In Windows Server 2008, it was previously possible to disable the Windows Firewall service and the Network List services to avoid passing through some of the more expensive network layers [40]. This no longer appears to be the case for Windows Server 2008 R2. Testing showed that Winsock could not exceed rates of 150,000 packets per second to a single destination with the services enabled or disabled. In order to perform large scale network scanning, an application will need to connect to a large number of unique destinations at a single time. Windows incurs an additional penalty in this situation. As the number of unique destinations increases, the throughput rapidly degrades. With the previously mentioned services disabled, it was not possible to exceed 20,000 packets per second with all unique destinations in our tests. As can be seen in Table I, increasing the number of threads (six cores were available, with one socket per thread) did not significantly improve the sending rate for either single destination tests or multiple destination tests.

		Rate in pps			
Socket Type	Destination	1 Thread	2 Threads	3 Threads	6 Threads
Regular	single	108,448	145,730	146,002	146,751
	all unique	15,433	16,715	16,926	19,002
Raw UDP	single	113,017	146,101	146,326	147,014
	all unique	15,582	16,772	16,751	18,570
Raw IP	single	123,314	127,730	144,626	145,879
	all unique	15,396	16,851	16,595	19,429

Table I. Windows Server 2008 R2 using Winsock to send 1,000,000 UDP packets with 100B payloads

Networking on Windows revolves solely around Winsock, with no third-party libraries available. The only option available was to implement our own network

driver. Windows provides multiple kernel level driver extensions for authors to use to interact with the network. At the top is the Windows Filtering Platform (WFP), which allows for packet inspection and modification. Below that is the Winsock Kernel (WSK), which provides a socket-like interface at the kernel level. At the very bottom is the Network Driver Interface Specification (NDIS) API driver, which acts as a wrapper for accessing the Network Interface Card (NIC). WFP would allow us to filter out the packets we are interested in, but we would still not be able to solve the bottleneck that occurs while sending packets. WSK would allow us to bypass the user space layers, but packets must still be processed by the Windows TCP/IP stack. For our purposes, we do not need Windows to perform any processing on the packets, so that just leaves the very last layer, NDIS. NDIS allows us to send and receive packets directly above the NIC without passing through any of the other Windows networking layers.

3.2 Overview

Since Windows does not provide access to its kernel directly, nor many of its libraries, it is not possible to patch the Windows networking API (Winsock). However, Windows does provide the ability to mount custom drivers into the middle of the network stack. This allows users to redirect packets before they reach processor intensive libraries. There are multiple types of network drivers available (miniport, filter, and protocol) and two of them will be employed in this project. See Figure 1 for the new network path which bypasses the Windows libraries. A filter driver will rewrite the protocol type of packets of interest and a protocol driver will be registered to accept that custom protocol. Since it will be using a non-standard protocol type, the packet will not be sent to the Windows TCP/IP protocol driver and thus avoids unnecessary

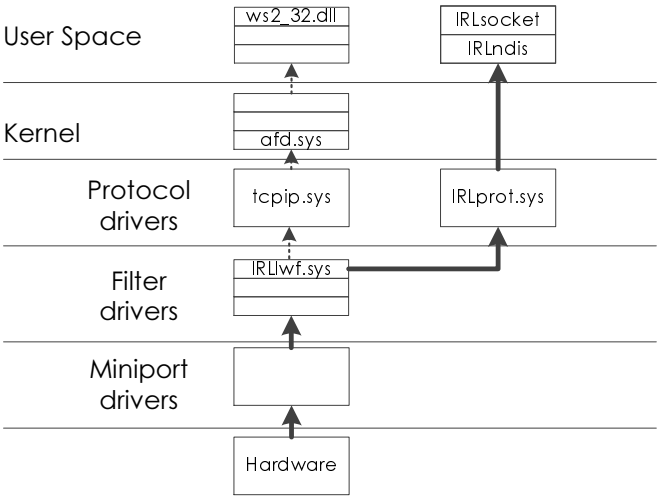

Fig. 1. Network path through kernel

overhead.

3.2.1 Lightweight Filter Driver

The filter driver's only purpose is to flag packets of interest for the protocol driver. This is done by checking the IP address of the inbound packet. If the packet is destined for an IP address that is not the primary IP address of the computer, the packet's Ethernet type will be rewritten to 0xABAB. Any packet that is not an IP packet or is sent to the primary IP address will be allowed to simply pass through the driver without modification. By redirecting packets this way, the custom drivers will not impact the operating system's normal networking capabilities when they are activated. Any experiment that needs to use the drivers can be performed on secondary IP addresses.

3.2.2 Protocol Driver

When the protocol driver is mounted, it registers two function hooks: `IRP_MJ_READ` and `IRP_MJ_WRITE`. This provides the primary entry points into the driver from user

space. Users cannot interact with the driver like a normal socket, instead they use the Windows API `ReadFile` and `WriteFile` operations to read and write data on the network device. The user space interface uses an I/O Completion Port (IOCP) architecture, which means that the user will initially issue multiple `ReadFile` requests and wait for the driver to alert them when a packet has been delivered. Each `ReadFile` request creates an I/O Request Packet (IRP) inside of the driver, which signals to the driver that an application is ready to receive packets.

3.2.2.1 Receiving Packets

When a packet arrives from the filter driver, the `NdisprotReceiveNetBufferList` function of the protocol driver is invoked. One of the parameters to this function is a list of `NetBufferLists`, each of which contains a set of packets from the transmission. The first IRP is retrieved from the queue of all pending read requests and packet processing begins. Packets are then placed into the user space buffer associated with the current IRP. In order to minimize the number of interrupts, packets are sent to the user in batches. Since packet lengths need to be known by the user, a custom frame header containing the length is written to the user buffer prior to writing the packet. If the user's buffer becomes full, the IRP is marked as complete. This indicates to the driver that it needs to alert the user application that the read has completed. The user space application is notified by an interrupt which triggers a completion packet that the user can wait for.

The original version of this driver only processed one `NetBufferList` before completing the IRP, which led to a large number of interrupts being issued and the CPU dropping packets while it was busy servicing interrupts. To avoid that situation, all `NetBufferLists` that were received are processed together. Additionally, since it may be possible for only a few packets to have arrived in one list, it would be more

efficient to provide as many packets at once to the user before returning the buffer to them. Therefore, packets are processed until the user's buffer is completely full. A kernel timer is used to mark the first pending IRP as complete every 50ms to prevent the user's buffer from being held for too long.

3.2.2.2 Sending Packets

The sending side of the driver is more straightforward. A user writes the data to the driver handle with a special frame header in front indicating the length of the packet. This allows the sender to perform batching like the receive side does. When the protocol driver receives the packets from the user, it creates the necessary `NetBuffer` structures for the packets and places them all in a single `NetBufferList` structure for sending. The list is then sent to the filter driver, which passes it on to the miniport.

3.2.3 User Space Interface

Unfortunately, since we are bypassing Winsock, it cannot be used transparently in existing programs. Two user space classes were written to abstract away the complexities of interacting with a raw protocol driver. The first class interacts directly with the driver and handles all of the IOCP details such as queuing up a sufficient number of read requests at all times and holding write requests for batching. The class also provides UDP demultiplexing so that sockets can receive specific subsets of the incoming packets. Users are able to specify which IPs and ports their socket should receive packets for, and arriving packets are placed into queues for those sockets. If a packet is destined to a local (IP, port) pair for which no socket is listening, then that packet is simply dropped and not passed back to the Windows TCP/IP stack.

The class which the users directly interact with is a socket-like wrapper around

the previous class. The main functions are to set a filter (IPs and ports) to listen for packets, bind to a destination IP address, send data, and fetch all packets that have arrived. The last function is slightly different from a normal socket in that you receive multiple packets simultaneously, instead of singularly. This requires adjusting control logic in the application, but it allows us to reduce the amount of time locking the packet queues.

For a detailed overview of the new kernel and user space models, see Figure 2.

3.3 Alternative Model

The current architecture could be improved upon to reduce the amount of interrupts and memory copies that take place. The concept would be similar to how PF_RING works, with shared memory between kernel space and user space. Instead of using `ReadFile` and `WriteFile` to generate IRPs which pass pointers to memory buffers that must be memory-mapped each time, it would be more efficient to use two shared ring buffers. When incoming packets arrive, they would be placed directly into the shared ring. User applications can poll the head/tail pointers on the ring to see if new packets have arrived and process them. The same could be done for the sending side as well. The user applications place outgoing packets into a ring and the kernel has a timer which polls for new arrivals. Batching would no longer be needed on the receive side since interrupts would not be generated since IRPs are not being used. The sending side could still perform some batching to minimize the number of `NetBufferList`s that must be created and sent to the miniport.

Additionally, the socket style approach to the user space classes could be changed to be more efficient. Currently, users allocate a buffer and write their data to it and then pass that buffer to the socket library. Since the user typically only writes data

and not a full frame, the socket library must allocate a new buffer with enough space to add the other headers (i.e., Ethernet and IP). The user's data must then be copied into the new buffer. A more efficient approach would be for the user to request a buffer of a specific size and have the socket library return a pointer to the buffer. The socket library would create a single buffer with enough room for the missing headers and return a pointer to the user data section of the buffer. This way, the user would be writing their data directly into the correct location without requiring a second memory allocation and copy by the underlying socket library.

By removing interrupts and reducing the number of memory allocations and copies, this model should be even more efficient than the current architecture being used. It could be taken even further by directly mapping the miniport registers so that they are shared with user space, but that limits the driver to specific NICs.

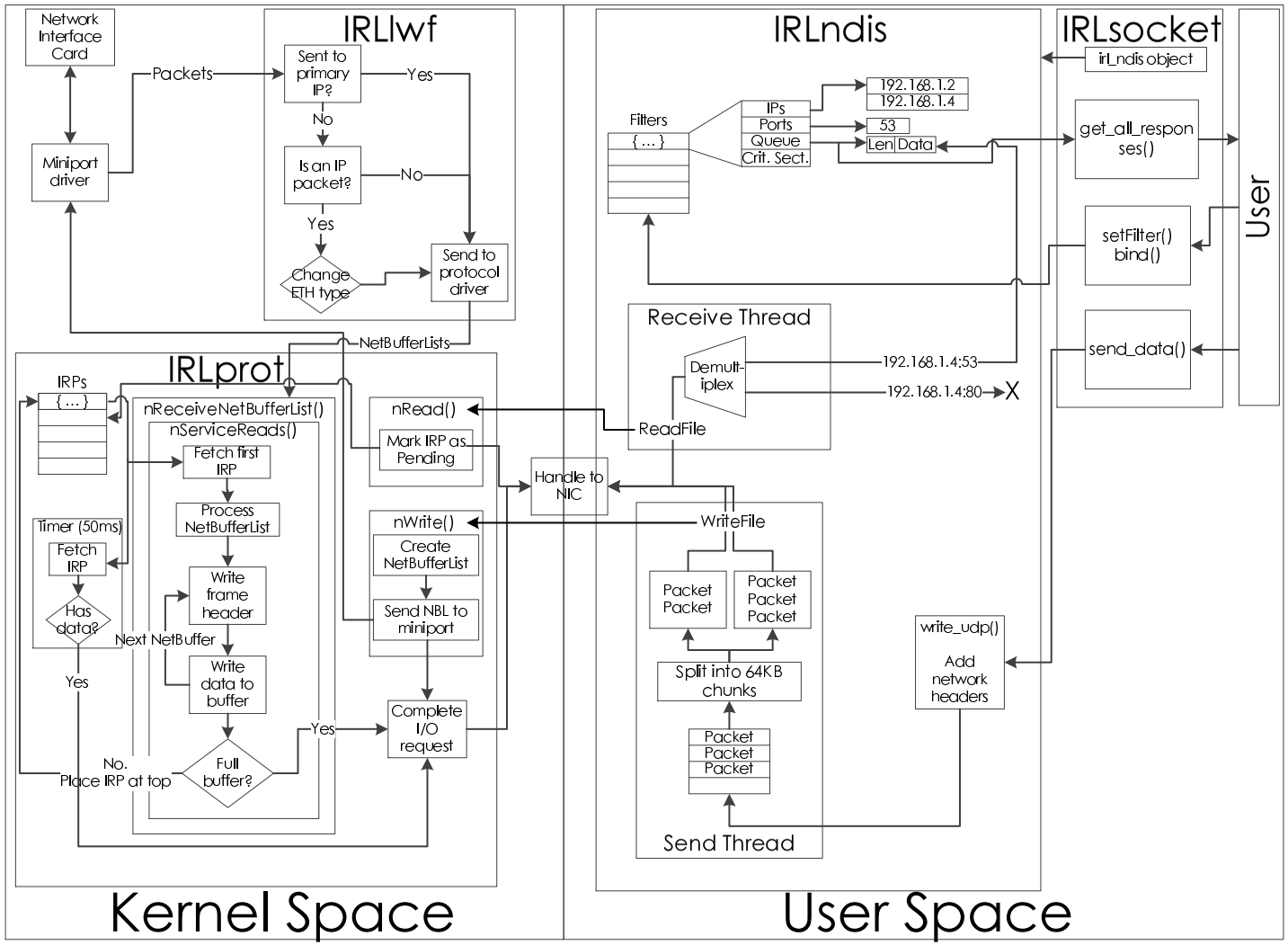

Fig. 2. Proposed architecture

CHAPTER IV

EXPERIMENT

To verify that the architecture works well in measurement scenarios, an experiment was performed based on the Turbo King system. Turbo King involves using a list of open, recursive resolvers and forcing them to send DNS queries to each other by using specially crafted responses.

4.1 Setup

In order to perform the experiment, an authoritative DNS server was required. An authority is required so that recursive resolvers will acknowledge the response we sent as credible. A domain name is also required so that we can direct traffic to our authoritative server. Originally, the domain name for our research lab was used (irl.cse.tamu.edu). This domain is delegated from the university resolvers to a different server which answers all queries for the domain. With dns.irl.cse.tamu.edu being used as the domain for tests, a third server was used as its authoritative name server as well as the actual experiment server. Once the authoritative servers were added to the target servers' caches, the university and lab resolvers should not be contacted. However, we found that many extra lookups were being made directly to the lab authority server and it was unable to sustain the workload. This created a bottleneck in the experiment, so a different domain was required. irl-dns.info was then registered with the experiment server set as the primary name servers. Traffic would no longer be sent through the university or lab resolvers, it would all be directed through this single machine. Since this server was running the driver previously described, the

workload would be easily managed.

The change to a new registrar did not go as smoothly as anticipated. A short test was performed to ensure that the .INFO resolvers and the test server could manage the workload of the experiment. The test was identical to the experiment, except that the duration was only a few minutes instead of hours. Everything ran without issue, until the next attempt which resulted in a large number of failed responses. The responses were not errors, but timeouts. After a quick diagnosis, it was found that the `.INFO` resolvers were dropping all requests for the new domain. Inspection of the domain's registration status showed that we were now in a blocked state. After explaining the purpose of the domain to the registrar and their subsidiaries, the domain was unblocked. While they did not provide a reason for blocking the domain, it was most likely blocked because they thought it belonged to malware. The domains being used in the test all have short TTLs to avoid cache pollution, so they may have looked like fast flux domains [22]. Additionally, the naming convention used for the test domains may have been mistaken as being generated by a domain generation algorithm which is often employed by malware [37].

In order to perform a test involving high throughput, a large number of open, recursive resolvers would be required. To avoid performing a scan of the entire Internet, a list obtained from a scan in 2008 was used. Another scan could be easily performed, but they often cause complaints to be sent to the university's network department if an intrusion detection system detects the scan. The previous scan contained 125,000 servers that were still active. To reduce the risk of being detected by intrusion detection systems, IPs belonging to the same subnets needed to be removed. Furthermore, it should be sufficient to test only one server in each BGP prefix since the other servers in the same prefix should be relatively close to the chosen target. A current listing of BGP prefixes was obtained from Route Views [39] and it was found

that 24,000 unique BGP prefixes were contained in the list of active servers from the previous scan. This was sufficient to perform the experiment, so another scan was deemed unnecessary.

4.1.1 Scanning Algorithm

The algorithm used for selecting which servers to measure must be efficient and have minimal impact on the target hosts. The algorithm must meet the following criteria: maximum politeness, maximum stealth, scalable, resumable, and allow for changes in the host selection. The measurement scan must be polite in the sense that we do not want to overburden the machines performing the measurements. If the hosts become too busy, delays will be introduced, which invalidates the measurement. The measurement scan should be stealth so that intrusion detection systems are not alerted, causing the scan to be blocked. The algorithm should be scalable and work on measurement matrices of any size. The amount of memory required for the algorithm to operate should not grow to the point where it is not feasible to perform an Internet scale scan. Large scale measurements cannot be performed instantly, so it may be necessary for a measurement scan to be stopped partway through. The algorithm in use should allow for the scan to be stopped and resumed at a later time. It should also allow for changes in the host selection. If a host becomes unavailable during the measurement scan, it should be possible to replace the host with a new one, without the need to stop the scan.

Leonard et al. [24] introduced the idea of a reverse linear congruential generator. It uses a normal linear congruential generator with constants known to produce a random distribution, $x_i = (214,013)x_{i-1} + 2,531,011$. The bits of the resulting 32 bit integer are then reversed. This is stored the next IP address for use in their scan. They go on to show that it does produce an uncorrelated sequence of addresses.

For our purposes, we will use two RLCGs to produce the scanning matrix axes. This ensures that both the source and target selections are randomized, which will make the scan achieve maximum politeness. In addition to a sufficiently random host selection, the TKing client will use 16 different source IP addresses for performing the experiment. This will aid in achieving maximum stealth. With multiple source addresses, intrusion detection systems will be less likely to detect the scan.

Once the matrix is formed, hosts are selected along the diagonal to perform measurements. A single diagonal is measured at a time, with some time allowed for responses to return before the diagonal is shifted by one for the next set of measurements. This method of scanning requires a minimal amount of memory for operation, even at an Internet scale. The two RLCG seeds require a total of eight bytes of memory, the iteration count requires four bytes of memory, and the two lists of IP addresses will require a maximum of 3.6MB (475,000 addresses at four bytes each). The scan order can be seen in Figure 3, where an X indicates a pair that has been skipped since both the source and destination IP are the same.

Should the scan need to be stopped, the seeds and iteration counts can be stored to disk. When the scan is resumed, the seeds can be restored, the matrix axes recomputed from the RLCGS, and the iteration count used to advance the RLCGs to their last location. The measurements will then resume exactly from where they left off.

In the event that a host becomes unavailable, it is possible to update the two arrays of IP addresses which serve as the axes of the matrix. The two entries can simply be updated with a new host from the same BGP prefix and the scanning algorithm will be unaffected. No further changes are necessary.

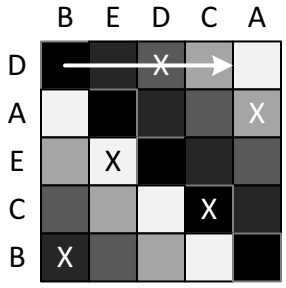

Fig. 3. Scanning order for hosts A through E

4.2 Operation

The experiment used 20,000 IPs which belonged to unique BGP prefixes, randomized according to the scanning algorithm. Each diagonal of the matrix was tested over a period of two seconds to further decrease the risk of overburdening any of the hosts involved in the experiment. The test performed between each pair of IPs typically consists of six DNS queries, but can require several more depending on how the target servers behave. Given a pair of servers, A with IP 1.0.0.0 and B with IP 2.0.0.0, the ideal test path can be seen in Figure 4. To ensure that the `irl-dns.info` name server information is in the servers' caches, we initially seed all of the servers with an initial recursive query for heartbeat.irl-dns.info. Any servers that did not have the name server information in their cache should add it at this point. The `.INFO` TLD name servers provide our records with a one day TTL, which is sufficient for the time frame of the experiment.

The first query issued by our TKing client is a recursive query for an A record for `A-B.irl-dns.info` to server A. Server A will then send an iterative request to our authoritative server requesting the IP address for `A-B.irl-dns.info`. We reply with a referral to server B. The referral consists of a NS record for `ns-A-B.irl-dns`.info in the authority record section and an A record for `ns-A-B.irl-dns.info` with

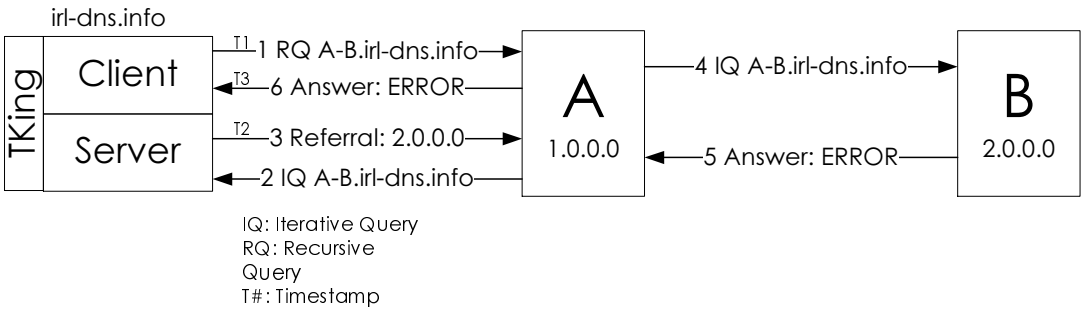

Fig. 4. Turbo King

server B's IP address in the additional record section. The additional record is provided to avoid an extra query back to our name server for the IP address of `ns-A-B.irl-dns.info`. Since we are the authoritative server for irl-dns.info, the IP address should be accepted and cached.

At this point, server A should send an iterative request to server B for `A-B.irl-dns.info`. Server B is not authoritative for that domain and should respond with an error. Server A will then send an error response back to the TKing client indicating a failure.

It is possible to determine the delay between servers A and B by taking timestamps after sending request 1, receiving request 2, and receiving the error code at request 6. The delay can be computed using $d_{36} - d_{12}$, where d_{36} is the time between requests 3 and 6 and d_{12} is the time between requests 1 and 2.

The experiment was performed over a period of 11 hours, operating at 150,000 packets per second with a throughput of 120Mbps. This server, which was used as both the TKing client and server, maintained a CPU usage of roughly 20% for the entirety of the scan.

CHAPTER V

RESULTS

Ideally, only six queries would be sent in total during this experiment; however, there are several intermediate queries that occur which can affect the timing measurement. The original King and Turbo King papers do not make any mention of extra requests that may be generated during the test. Either these queries are the result of changes in the DNS software running on the servers, or the queries were somehow missed during the original assessments.

5.1 Queries Received by the Server

In addition to the queries shown in Figure 4, there are NS record requests for the primary domain, A, AAAA, and A6 record requests for both of the primary name servers (`ns1.irl-dns.info` and `ns2.irl-dns.info`), and A, AAAA, and A6 record requests for `ns-A-B.irl-dns.info`. This experiment was performed a second time, with a prefix of "facebook-" applied to the domain names. This was to evaluate the effects of censorship on the links. The tables below show statistics from that experiment.

Type	Iterative	Recursive
A	822,934,601	1,796,925
NS	779,083	0
TXT	80,300	0

Table II. Queries sent to TKing server, for `facebook-#-#.irl-dns.info`

The requests for the test domains were fairly reasonable. Table II shows that we received double the amount of A requests than we issued because most of the

hosts sent a query to both of our name servers. The 1.8 million recursive requests were somewhat unexpected. Since the target servers are all recursive resolvers, they should send us an iterative query and not a recursive query. There may be a bug in some of the hosts' resolvers which are not changing the query type bit when sending the next query.

Type	Iterative	Recursive
A	118,502,656	0
AAAA	1,566,715,884	113,904
A6	27,350,904	0

Table III. Queries sent to TKing server, for `ns-facebook-#-#.irl-dns.info`

As can be seen in Table III, the bulk of the traffic during the experiment came from lookups for the intermediate name server for the test domain (ns-facebook). This is the name we use as the authoritative server for the test domain. When server A comes to us asking for `facebook-A-B.irl-dns.info`, we provide an NS record stating that
`ns-facebook-A-B.irl-dns.info` is the authority for that domain. We also provide an A record in the additional section providing B's IP address for
`ns-facebook-A-B.irl-dns.info`. Since we are the authority server for `irl-dns.info`, resolvers should be using the information provided in the additional section. However, many servers still request A records. This indicates that server A did not trust the information it was provided. If a DNS resolver properly implements bailiwick rules and checks that we set the authority bit, there should be no reason to ignore the records provided.

In July 2000, the A6 resource record type was proposed to map domain names to IPv6 addresses and replace the AAAA resource record [10]. The A6 record type was later downgraded to experimental status in August 2002 [6]. ISC's BIND added

support for A6 record requests in version 9.0.0, which was released in September 2000. In October of 2003, BIND version 9.2.3 was released which removed their usage of A6 during normal lookups. However, even after ten years, there are still resolvers running versions of BIND which issue these requests as part of the normal lookup process. During the experiment, they accounted for over 30 million lookups.

The 1.6 billion AAAA requests was not anticipated. Initially, it was observed that several resolvers continued to issue AAAA requests after the experiment ended. Since the experiment was not designed with IPv6 support, the requests were originally responded to with an empty response which indicates that another record for the domain exists. After changing the response to an error code (Not Implemented), those hosts stopped issuing requests. The result, however, was the 1.6 billion requests.

Type	Count	
	ns1.irl-dns.info	ns2.irl-dns.info
A	42,377	46,856
AAAA	38,817,051	38,812,238
A6	5,107,092	5,106,673

Table IV. Iterative queries sent to TKing server, for primary name servers

The primary domain's name servers also suffered from a large number of IPv6 requests, as can be seen in Table IV.

Since both the recommended method of an empty recordset and the Not Implemented error code did not appear to reduce the amount of AAAA requests, several small experiments were performed with varying responses given. Each test was the full experiment in scope, except that it was only run for ten minutes. The results are shown in Table V. An empty response indicates that no record of type AAAA existed, but there are other records for the domain available. This is the recommended approach to use. Returning an error code is not recommended, but certain error

Response Type	Code	Queries Received
Empty	0	2,178,424
Name Error	3	2,202,844
Refused	5	4,566,869
SOA	0	4,462,500
Not Implemented	4	6,597,407

Table V. AAAA queries received with error codes sent

codes are less damaging than others. Name Error and Refused are not that kind. It is recommended not to use this for fear of resolvers misinterpreting their meaning. In the past, resolvers have treated a Name Error as an error for every record request for the domain, instead of just the AAAA request. So if both an A and AAAA request were in-flight at the same time, but the AAAA response came back first with a Name Error, the resolver would report that the domain did not exist; instead of returning the answer that came with the A response [30]. The same issue occurred with refusals; some servers treated it as if the server refused all queries. An error code of Not Implemented was deemed as less harmful, but from the above results this is the most poorly supported error code for AAAA requests. The Start of Authority (SOA) response is similar to an empty response in that it does not contain an answer. It does contain, however, a SOA record in the authoritative records section of the response. The reasoning behind this is that it contains a Minimum TTL field, which resolvers should make note of before sending another request for the same record. Since the test lasted only ten minutes and the minimum TTL value was set to an hour, it does not seem to be the case that resolvers are checking this properly.

While SOA records can be returned to provide resolvers with extra information [29], there is little incentive to do this during the experiment if it doubles the amount of AAAA traffic and requires additional bandwidth. The most efficient response is to reply with an empty response with an OK return code. Name Error also showed

minimal results, but due to the concerns previously mentioned it should not be used.

Type	Iterative	Recursive
A	14	2
AAAA	16	0
NS	1,263,124	508
CNAME	20	0
SOA	2	2
MX	20	2
DS	28	0
ANY	0	2

Table VI. Queries sent to TKing server, for `irl-dns.info`

The behavior shown in Table VI is very curious. There are resolvers which are asking for NS records for the primary domain, `irl-dns.info`. This is interesting because the `.INFO` TLD provides these records. If a server is asking us directly for them, it would seem to indicate a partial lack of trust between the resolver and the TLD. One possible explanation is that the resolver is attempting to verify that our name servers have not changed, but it should not be requesting them so frequently. The TTL provided by the `.INFO` TLD is one day, while the TTL provided by our own server for those records is one week. With a measurement being performed every two seconds, the resolver should not evict our records from their cache unless they were extremely busy. Not only must the resolver decide to verify our NS records, but also choose to not cache them.

The full set of interactions that occurred during the experiment can be seen in Figure 5. An arrow pointing down indicates a query that we received and an arrow pointing up indicates a response that we sent. Each column represents the domains being requested, with arrows grouped together by row based on the query type issued (i.e., A, AAAA, NS). Each arrow is also labeled with a number indicating which phase it is part of. Below the arrows is a count of how many of each packet type occurred. For labels under arrows in both directions, the count is the number

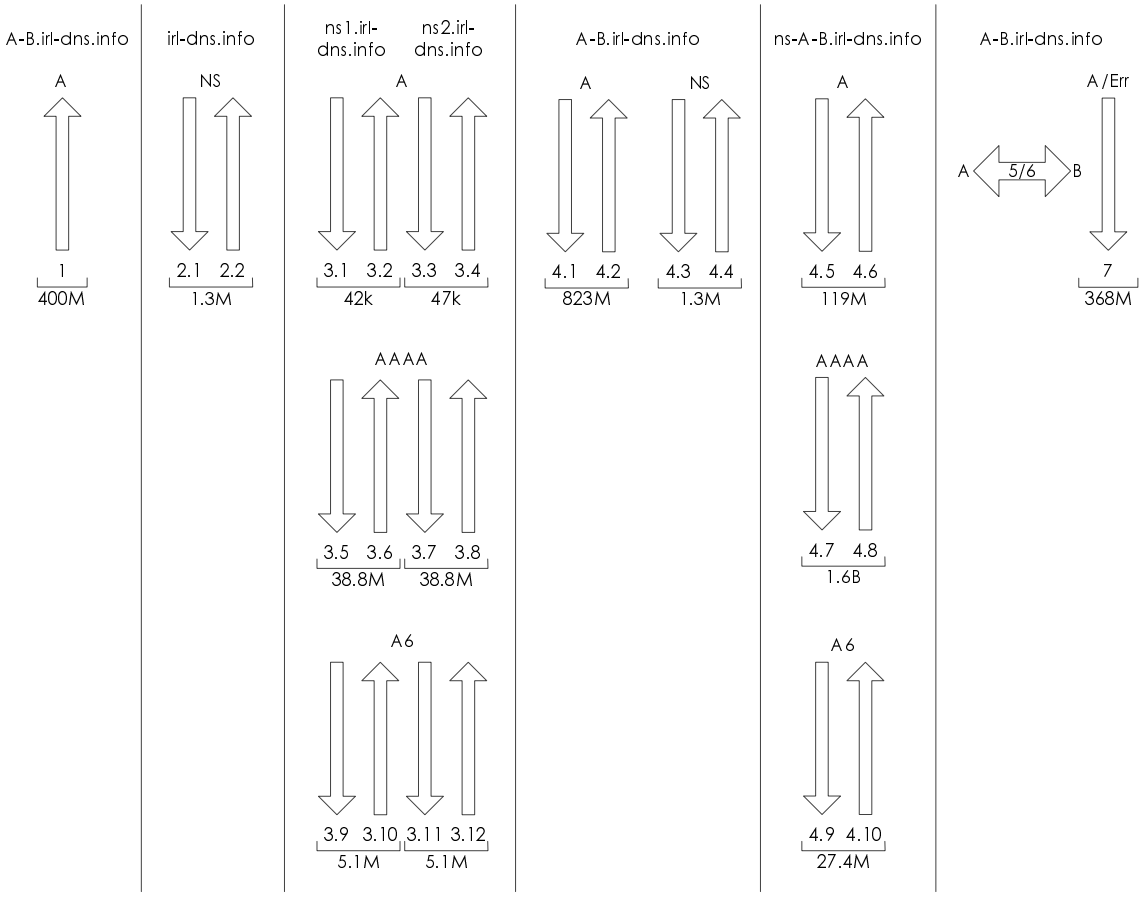

Fig. 5. Turbo King: Actual queries received

of packets sent or received in only one direction since it is mirrored in the other direction. The experiment begins at phase one, with the request we send for the test domain `A-B.irl-dns.info`. Phases two and three are extra requests about the primary domain, `irl-dns.info`. Phase four is when server A contacts us about the domain and we send a referral. It also contains extra responses for when server A does not use the additional records of the referral or requests IPv6 information (AAAA and A6 requests). Phases five, six, and seven are the last steps of the TKing process where server B is queried and the error is sent back to server A and then on to us.

5.1.1 Queries With Error Codes

The TKing server received 102,874 queries with the return code set to Server Failure; primarily from nine hosts. The original packet received by a resolver is often used to create the response since the DNS header fields and question remain the same. One possible explanation for queries with return codes is that there is a bug in the resolvers causing the query/response bit to not be flipped from "query" to "response."

5.2 Return Codes Received by the Client

During the final step of the measurement process, an error code should be returned by server A indicating that server B did not have any information about the domain we requested. The most appropriate error code in this case would be Server Failure (code 2), which indicates that there was a problem with the name server. While many resolvers return this error code, several do not. The set of return codes received by the TKing client can be seen in Table VII.

Response Type	Code	Responses Received
Format Error	1	60,383
Server Failure	2	353,458,498
Name Error	3	113,647
Refused	5	1,150,025

Table VII. Return codes for responses sent to TKing client

The Format Error and Name Error codes were primarily caused by only a few hosts. Seven hosts accounted for 99.9% of the Format Errors, with only 20 other hosts providing one or two responses with this error. Name Errors were also dominated by seven hosts, accounting for 62.7% of the errors. Many other hosts accounted for the remaining errors, only contributing a few responses each. While most of these

hosts only responded two or three times with a name error, three destination hosts frequently appeared as the target server of the measurement. These three hosts accounted for 36.1% of the name errors. The Refused error code was presented by 209 hosts, with 140 of them replying with the error more than 5,000 times.

While Refused errors are understandable since the target server most likely returned the refusal and the source server forwarded it on, the Name Errors should not occur. A Name Error is only useful when it is returned by servers which are authoritative for the domain in question.

5.3 Queries Received by the Client

The TKing client should only perform two steps: send a question to server A about server B and receive the response to that question. While this was how it operated for the majority of the time, a few queries were actually received by the client. Of these queries, 4,437 were the same queries which the clients themselves were sent. Over half of these queries came from the Verizon network. The host sending the most recursive requests back to the client was a Verizon digital video recorder (DVR). The rest of the hosts were largely various brands of home routers.

In addition to the queries that were sent back to the client, we also received two sets of peculiar queries. The first set of queries was a request for the root. What makes the request interesting is that the request contained records for the root servers. This does not appear to be malicious since the records provided were all correct. We received 16,132 of these requests, all of which originated from an IP address operated by the European Fusion Development Agreement Task Force. The other set of queries we received was a completely blank packet; the UDP payload was 512 bytes of zeros. This is technically a valid packet, but it asks no questions and

contains 500 bytes of padded zeros at the end. We received 11,474 of these requests from Travelsky Technology Limited (a Chinese public aviation company) and 34 from the China Aviation Computer Information Center.

5.4 Non-Error Responses Sent to the Client

The responses sent to the TKing client should only consist of non-zero return codes, indicating an error. Most of the responses were indeed marked as errors, but 13 million of them used a return code of OK (code 0). Table VIII shows response counts for the test domains (facebook-A-B.irl-dns.info) which contained extra records, but no answer records. The remaining 7 million records contained an answer section and will be discussed in the next chapter.

Response Type	Count
facebook-A-B.irl-dns.info	
NS and Additional records	989,731
NS records only	761
irl-dns.info	
NS records only	154,831
NS and Additional records	30,433
NS records + 1 Additional	9,138
Other	
No records	4,931,031
Roots and .com records	19,727
Roots	5,411
.info records	4,697
Roots subset	489

Table VIII. Responses returned to TKing client with OK status code

Roughly 1 million of the responses contained the name servers for the test domains (ns-facebook-A-B.irl-dns.info) and its IP address in the additional section. A few of those only contained the name server records in the authority section. We also

received about 200,000 responses which contained records for the primary domain (irl-dns.info). The responses either contained just an authority section with both NS records, both NS records and both IP addresses, or one NS record and its IP address. These responses appear to be terminating the lookup early and are simply passing the referral back to the sender instead of resolving it completely.

Almost 5 million of the responses did not contain any records at all. The response only contained the question and an OK return code. There were also responses which contained information about the root servers or the other top level domains (TLDs). 20,000 of the responses contained authority and additional records for the 13 root servers and the 13 `.COM` name servers. 5,400 contained records for just the 13 root servers. 5,600 of them contained authority and additional records for the `.INFO` name servers. Lastly, 500 responses contained records for just six of the root servers. It is not uncommon for a non-authoritative name server to send a referral back to the root servers or to the TLD's name servers, but since these are recursive requests, the recursive resolver should not be returning the referrals back to the sender. Server A should be following the referral to completion or returning an error.

5.4.1 Headers Only

A valid DNS response should contain a return code indicating the state of the response, the domain for which the original query was issued, and, if applicable, resource records containing information about the domain. Like many protocols, the DNS protocol is prone to attack. If an attacker can inject a malicious record into a resolver's cache, many unsuspecting users can be sent to a malicious site instead of their original destination. For this reason, it is important to verify DNS responses to ensure that they match the question that was asked and contain reasonable answers. Verification should check that the source port, transaction ID, and question match

that of the original query.

Of the responses received by the TKing client, 737,262 of them contained only the 12 byte DNS header with an error code set. The header contains information such as the transaction ID, number of questions, number of answers, and so on. What it does not contain is the domain requested or other resource records. The only verification which can be performed is on the transaction ID and source port. If a resolver only performs checks based on the transaction ID alone, an attacker can easily perform a Denial of Service attack on the resolver by exhausting all transaction IDs and responding with an error code. No resolver should ever accept a response with the DNS header alone, and no resolver should ever send such a reply. These responses came from 72 hosts, with the majority of the responses coming from France, Germany, and Brazil. The set of return codes used by these responses can be seen in IX.

Response Type	Code	Responses Received
Server Failure	2	514,494
Format Error	1	220,220
Refused	5	2,548

Table IX. Return codes for header-only responses

Upon manual inspection of the top 50 IP addresses, which account for 99.99% of the responses, 10 of the devices were manufactured by bintec [41]. There was also a Cisco router, an Untangle router [42], three broadband modems, and two phone systems. The remaining devices did not present a response to a HTTP request for identification.

5.4.2 Other Responses

We also received a few responses which were malformed or possibly malicious. There were 78 responses from Chinanet IP addresses which were classified as malformed. Upon inspection, the data did not appear to be DNS traffic. The packets may have been for another protocol, but it was impossible to tell which one as the contents did not contain ASCII strings that may have helped in its identification.

While those packets may have been benign, but misdirected, 23 of the responses may have been sent with malicious intent. These responses contained the results of an ANY request for isc.org. The response is 3,000 bytes long. This is suspected as malicious due to their prominent use in DNS amplification attacks [8]. Since this was a response and not a request, we were specifically the target in this case. The number of responses was extremely small and originated from only two hosts. Interestingly, one of those hosts was the European Fusion Development Agreement Task Force which was previously mentioned for sending root requests while providing the answers. The other host was from within the Ngee Ann Polytechnic network.

5.5 Bogus Questions

During the experiment, we received both queries and responses for domains which we are not authoritative for. While one was a crawler, the rest appeared to be mistakes.

5.5.1 Queries to Client

The TKing client received the least amount of incorrect queries. It received several queries from internal networks within the university which were broadcasting requests, it received several from a DNSSEC crawler (http://www.dnssecready.net/) checking for DNSSEC capability, but the only two interesting queries came from the Verizon

network. Those queries can be seen in Table X.

Question	Count
1.0.0.127.in-addr.arpa	9
www.myspace.com	2

Table X. Queries sent to TKing client

5.5.2 Responses to Client

The TKing client also received 400 responses for 200 kinds of questions we did not ask.

Question	Count
sabi.np.edu.sg	45
nova.np.edu.sg	33
lc2.midhq.gov.sg	21
activation.windstream.com	14
verify.kx8.cn	14
isc.org	12
secdns1.MIDHQ.GOV.SG	11

Table XI. Top responses sent to TKing client

Two hosts showed up several times in the responses: the Ngee Ann Polytechnic university and the Windstream ISP. Aside from what is in Table XI, Polytechnic also requested several reverse lookups for its own IP addresses. Polytechnic has been mentioned previously for being used to send the 3,000 byte isc.org responses. The isc.org requests are present here as well, but their requests were less than 512 bytes in size.

The responses showed great variance in the questions being returned. Some of the domains included: logmein.com, cisco.com, msn.com, gmail.com, and facebook.com. The responses all contained legitimate records for their respective domains and did not contain malicious entries. There was only one instance that had the potential to

be malicious. One of the responses was for the query "localhost" and a record was returned. The source of the response was from a Chinese ISP, Chinanet, providing a record pointing to another address within Chinanet. It is possible that the ISP was simply responding to a lookup for "localhost" and returning the user's own IP address, but the query was somehow redirected to us.

5.5.3 Queries to Server

The TKing server received 310 queries for three types of domains. There were two recursive requests for www.irl-dns.info, which were most likely performed manually. The rest of the queries were iterative and fit into two formats: `<random string>.facebook-A-B.irl-dns.info` and `ns-A-B.irl-dns.info`.

An example of the first type is ealhnhaaaaesb0000dkaaabbaacbaggj.facebook-04814-09467.irl-dns.info. The string at the beginning partially changes between queries. It does not appear to actually be random, but block based. The first six characters always change, but the next five are always "aaaae" followed by two letters which are either "sb", "ka", or "ra". The next five characters are always "0000d" followed by a single random letter and then the string "aaabbaacb" with the last four digits being random. One possible explanation is that this is a delegated DNS server which is attaching an authentication token to the question before forwarding it. A similar approach is used by Pagekite [34], but the authentication token format is different. Interestingly, the source server of the test domain was almost always the same. Only two queries used a different source, the rest were all for 04814. Even though the source of the experiment was always the same, the hosts which were sending these queries to the TKing server varied. The queries came from Google, Indonesia, and China. This is most likely a bug where the forwarding server is not removing the authentication token before resolving the domain. This query type appeared 175 times.

The second query form (ns-A-B.irl-dns.info) may be a bug in the resolver which has somehow removed "facebook" from the name server address (ns-facebook-A-B.irl-dns.info). This query type appeared 131 times.

CHAPTER VI

ADDITIONAL INVESTIGATIONS

6.1 Censorship and Forwarders

A recent report [3] claimed that China's censorship of DNS traffic created collateral damage on networks that were not actually part of China's own networks. We performed our scan a second time and pre-pended the query with "facebook" to see if we received any censored results. Normally, we should receive an error message from server A since we are issuing a fake request. However, in the case of censorship, we will receive an answer with an IP instead of an error.

The results we found were quite underwhelming; only 1.7% of the results came back with an IP address. We did see that some of our requests were being censored, but all of these requests did actually pass through a network in China. We did not find any signs of censorship between pairs of nodes that did not exist within a Chinese network as the paper suggested. The original study used 43,482 open resolvers in 173 countries for their study. While we only used 20,000 servers, we did perform all-to-all measurements between them, which covers far more routes than the original study. The servers in our study spanned 171 countries including Japan, China, Korea, Canada, Australia, Germany, Russia, Taiwan, and Indonesia. It is unlikely that none of the countries tested never passed through a network within China. It is possible that the filtering rules are not simply keyword based or that the routes in our study did not happen to pass through a censored network.

The main finding of this investigation was that the bulk of the requests that received a fake response were not being censored, but were being redirected. Of

the 6.9 million requests that returned an IP address instead of an error code, 85% were OpenDNS redirects to their "Domain not found" search page. Only 6% of the requests were to a site on a Chinese network, but either one or both of the hosts in the measurement were within a Chinese network. The remaining 9% were primarily redirects to corporate "Domain not found" pages, such as ISPs and hotels. This result is not surprising. For the last three years, OpenDNS has grown by 100%, increasing their coverage to 50 million users. Our own university's local resolvers now forward to OpenDNS as well. The number of forwarding DNS resolvers will most likely continue to increase as more companies make use of commercial services such as OpenDNS.

6.2 Potential Improvements

While Turbo King managed to remove several unnecessary queries from the original King process, we wanted to investigate the possibility of adding more improvements. If we consider only one test at a time, those six queries are the absolute best we can do. However, if we consider caching and planning for future tests between nodes, is it possible to provide some answers ahead of time to reduce the amount of traffic? Is it also possible to request multiple tests in one query? These kinds of questions are subject to ambiguities in the DNS specification, which in turn makes the results specific to certain DNS resolver implementations.

6.2.1 Test Seeding

Since we are performing tests between every IP pairing, we will be visiting each sever multiple times. Additionally, since we know the order in which the tests will be performed, we also know the next servers that we will be requesting server A to contact. With this information, it would be beneficial to preload server A's cache

with the IP addresses of the future targets. The reason for this is that it would remove one RTT between us and server A (requests 2 and 3) and would also reduce the overall bandwidth requirement since we can benefit from answer compression. The idea is to provide records for future name servers and their IPs when server A asks for that information about the current test. In addition to receiving ns-A-B.irl-dns.info, it would receive ns-A-C.irl-dns.info, ns-A-D.irl-dns.info, and so on. Since our server is the authoritative server for the irl-dns.info domain, this information should be accepted by DNS resolvers. It turns out, however, that not all resolvers consider this valid. Microsoft's DNS resolver module accepts these answers and does indeed cache them, but ISC's BIND considers this a formatting error and discards the extra records. Since BIND holds 75% of the DNS resolver market share, they are the primary factor as to whether this optimization would work. With BIND rejecting the extra records, this method would actually increase the amount of bandwidth required, which is counterproductive.

The DNS specification allows multiple questions to be provided, but does not specify how the response should be crafted. Possible responses could contain answers to both questions, but that has problems when the error codes are different. There could be two responses sent, but they would be sharing a transaction ID, which is not ideal. Due to these ambiguities, BIND, Microsoft, PowerDNS, and several others have chosen to send a format error code when multiple questions are provided in a single request.

6.2.2 Zero TTLs

The Turbo King paper briefly noted that a few resolvers do not properly handle TTLs of zero. A zero length TTL indicates that a record is volatile and should only be used for the current transaction. Like Turbo King, we also found servers which

behaved improperly when given records with a zero TTL. The resolvers would repeat the query multiple times until they eventually gave up. One possible explanation for this is a DNS resolver which places the record into the cache before using it. Once the resolver attempts to process the pending request, the information is no longer in the cache and must be fetched again. While cache pollution is an important aspect to keep in mind while performing large scale tests, having servers repeatedly requesting the same domain increases the amount of traffic. For this experiment, a small TTL value of 30 seconds was found to be sufficient for reducing the amount of repeated queries.

6.2.3 Halving the Bandwidth Requirement

Due to a requirement by our domain registrar, two name servers were provided for the `irl-dns.info` domain. As a result of this, the amount of traffic doubled during the experiment. This can be easily seen in Table II where we received 800 million requests instead of the expected 400 million. Most resolvers sent a record request to both of the name servers. For intermediate lookups, the resolvers would tend to round-robin the requests between our two servers. One possible way to reduce bandwidth would be to find a registrar which allows the use of only one name server. This is unlikely, as RFC 2182 [14] actually recommends the use of three to seven resolvers. It may be possible, however, to use a subdomain for the tests and have the subdomain's authority delegated to a single server. For example, the new test domain could be `A-B.m.irl-dns.info`, with a separate name server being responsible for `m.irl-dns.info`.

During the experiment's setup, the heartbeat query would be changed to `hearbeat.m.irl-dns.info` which would cause the authoritative server for the `m.irl-dns.info` subdomain to be cached. When the experiment is performed, the two

primary name servers will no longer receive queries regarding the experiment and only the one delegated for the subdomain will receive queries. With only one server being authoritative for the subdomain, the amount of queries sent should be reduced by half.

	Normal	Subdomain
NS1	84,081	2,253
NS2	83,652	2,335
NS3	0	169,031

Table XII. Iterative requests received by each name server

A short experiment was performed which used only 300 hosts and the subdomain of `m.irl-dns.info` served solely by `ns3.irl-dns.info`. The experiment was performed again using the normal naming convention, with the same 300 hosts. The results are shown in Table XII. We did achieve the desired effect of directing nearly all of the traffic to the single name server, however, we still received twice the amount of requests for both tests. The most likely cause for this is that many recursive resolvers issue two lookups regardless of how many unique name servers a host has.

6.3 Increasing Accuracy

Due to the existence of previously unmentioned query paths, delay measurements can become inflated if timestamps are taken too early. The method described in King and Turbo King rely on knowing the time at which step three begins. This step is when our server returns a referral to server A about server B. Our experiment showed that server A may send additional queries depending on its trust behavior regarding the additional record section of a response. If server A makes additional queries, the timestamp should not be taken until server A makes its last query. This is the most

likely point at which server A will then query B and will provide the most accurate measurement.

6.4 Architectures for Large Scale Scanning

By the end of our experiment, roughly 13% of the hosts had stopped responding. Before the test began, each host was checked to ensure that the server was still an open, recursive resolver. Many of the hosts that stopped responding belonged to residential ISPs, such as Verizon and Cox Communications, which frequently re-assign user IP addresses. This would cause the target host to move without our knowledge and the new host may not have the capability to perform DNS resolution. In order to peform an Internet scale scan, each BGP prefix will need to have multiple hosts available for redundancy. If one host stops responding, it can be replaced with another host in that BGP space. There are currently over 475,000 prefixes and 2.6 billion IP addresses announced through BGP [19].

6.4.1 Dynamic Scanning

Ideally, one would scan the Internet and perform delay measurements at the same time so as to obtain the most relevant results. As soon as an open recursive resolver is found, it is added to a list of available resolvers and measurements would begin. There are a few difficulties to overcome though. The first is that we would like to cause minimal impact on the networks where these resolvers reside. We currently do this using a reversed linear congruential generator, which guarantees that the IP space is randomly distributed and no resolver would be accessed more frequently than the others. If hosts are used as they are found, the earlier hosts will be performing more scans than the new ones. One way around this would be to divide the complete

scanning matrix into blocks of a smaller size. Once enough hosts have been found to measure the entire block, the measurements can begin. While the scan and measurements continue, new hosts can be found for the next block of the matrix. This would at least allow us to guarantee some distribution of the address space in each block.

The second problem occurs when hosts are no longer available. As previously mentioned, residential ISPs may change a host's address and measurements will no longer be available. As the IP space is scanned, multiple hosts for each BGP prefix can be stored in the instance that one goes away. The new host will need to be checked again before it is used to ensure that the host is still active. If it is active, it can serve as the replacement. If there are no replacements available, then there will be holes in the measurement matrix. In order to revisit the missing measurements, the holes will need to be tracked and stored somehow. It is not possible to store the entire scanning matrix in memory due to its size (225.6 billion entries), and it is unlikely that the matrix required for missing measurements will fit in memory either if it shows a 13% migration rate like the experiment. If missing measurements are stored on disk, each time a new BGP prefix host is found, all missing measurements will need to be evaluated to see if the prefix is required to complete them. The old measurements cannot be immediately performed though, as the new host would be bombarded with measurements, which goes against our desire to evenly distribute the amount of work any single server must perform. The missing measurements would have to wait until no current block of the matrix requires their BGP prefixes. Even then, there must be enough missing prefixes found so that a matrix block can be formed in such a way that the hosts will not be queried too quickly.

Ensuring that no measurements are missed and that no hosts are overworked makes developing a framework in this manner quite complex.

6.4.2 Static Matrix

Rather than attempting to complete any missing measurements as soon possible, they can be delayed until later. A single scan is performed to find possible hosts for each BGP prefix, without any measurements being performed. Once the hosts have been found, a complete matrix is built in the same way as our original experiment. One pass of the entire delay matrix is then performed. New hosts are found for any missing prefixes by scanning the redundancy lists created by the initial scan. Once new hosts are verified, they are used to build a smaller scanning matrix and the scan begins again. If hosts migrate during this scan, the process is repeated again. This continues until all entries in the matrix have a measurement.

This method has a higher chance of hosts becoming unavailable for future scans since they are not re-evaluated until the end of the first scan. However, it does provide for a more random distribution of the hosts participating in subsequent delay measurements.

6.4.3 Pre-Screening

As was mentioned in the Results chapter, many hosts do not behave in the most ideal manner. They can eventually stop responding, ignore error codes for AAAA records, request records which have already been sent, or constantly need to request records for the primary domain. These behaviors can all be detected prior to using a host as part of the measurement matrix. After an initial scan has been performed to find potential hosts, they are monitored for several days. Each host is periodically sent a request to test for undesired behaviors. If a host is found to be behaving in such a way, it is removed from the pool of usable hosts. Once a set of stable and well-behaving hosts is found, they can be used to perform the all-to-all delay measurement scan.

In the case that a measurement is missed, it should be possible to simply replace the host with one from the list of redundant hosts for that prefix and continue on with the experiment.

6.5 Forwarders

If a resolver does not actually perform the resolution itself, but forwards the request to another server, delays will be introduced into the measurement. The Turbo King paper explicitly removed forwarders by ensuring that only the target host contacts the experiment servers. If other hosts are involved, then it is removed.

	Query	Response
Client	151	19,970
Server	37,798	0

Table XIII. Unique IP addresses contacting TKing client and server

During our experiment, we did not remove forwarders from the list of target servers. The number of unique IP addresses participating in the experiment can be seen in Table XIII. Unlike the delays caused by intermediate lookups, delays created by forwarders should be consistent since the forwarder will be used for every test domain. The goal of the delay measurements is to determine the amount of delay between two hosts with the expectation of finding the actual time delay. The exact time value is not actually important. Most applications really only need to know which host is more optimal. For this purpose, a relative time measurement can be used. The delay introduced by a forwarder now becomes a constant which has no impact on the actual measurement, since the goal is to only know which server has the least delay.

6.5.1 Minimum Delays

Once the matrix has been computed, it is possible to refine the measurements a bit more. By inspecting the columns of the matrix, it is possible to find the minimum delay experienced for a particular host. With a full, Internet scale matrix, this will represent the delay between the current host and its closest neighbor. This delay can be used as a substitute for the host's propagation delay and subtracted from every other measurement in the column. It can also be used to identify hosts which always have a large delay and should be removed from future measurements. Hosts which are routinely slow are either very busy or simply not suitable for delay measurements.

CHAPTER VII

CONCLUSION

Delay measurements can provide useful metrics to a wide audience including researchers, Internet engineers, businesses, and even gamers. Performing these measurements quickly and efficiently has not received much attention since accuracy is typically more important. While Microsoft Windows is the most widely deployed operating system [31], very little effort has been made towards performing high speed measurements using that system.

We developed a custom driver to facilitate the use of the Windows operating system for large scale measurements. This enabled us to perform a measurement scan that was larger and faster than any previous study. We developed a novel scanning algorithm and used it to perform 400 million measurements, generating over 3.4 billion packets in 11 hours. Analysis of the results revealed previously unidentified drawbacks to the existing architectures and measurement methodologies. We proposed novel methods for increasing the speed of experiments, improving the accuracy of measurement results, reducing the amount of traffic generated by the scan, and further enhancements for Windows network drivers.

REFERENCES

[1] 6WIND, "6WINDGate," http://www.6wind.com/products, accessed May 13, 2013.

[2] M. Ahmad and R. Guha, "Evaluating End-User Network Benefits of Peering with Path Latencies," in *Proc. ICCCN*, 2012, pp. 1–7.

[3] Anonymous, "The Collateral Damage of Internet Censorship by DNS Injection," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 21–27, Jun. 2012.

[4] A. Baggio and M. van Steen, "Distributed Redirection for the World-Wide Web," *Computer Networks*, vol. 49, no. 6, pp. 743–765, Dec. 2005.

[5] H. Ballani, P. Francis, and S. Ratnasamy, "A Measurement-based Deployment Proposal for IP Anycast," in *Proc. ACM IMC*, Oct. 2006, pp. 231–244.

[6] R. Bush, A. Durand, B. Fink, O. Gudmundsson, and T. Hain, "Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)," *IETF RFC 3363*, August 2002.

[7] C. Chambers, W. Feng, and W. Feng, "A Geographic Redirection Service for On-line Games," in *Proc. ACM Multimedia*, Nov. 2003, pp. 227–230.

[8] CloudFlare, "Deep Inside a DNS Amplification DDoS Attack," http://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack, accessed May 24, 2013.

[9] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, Distributed Network Coordinates," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 34, no. 1, pp. 113–118, Jan. 2004.

[10] M. Crawford and C. Huitema, "DNS Extensions to Support IPv6 Address Aggregation and Renumbering," *IETF RFC 2874*, July 2000.

[11] L. Deri, "nCap: Wire-Speed Packet Capture and Transmission," in *Proc. IEEE E2EMON*, 2005, pp. 47–55.

[12] L. Deri, "Improving Passive Packet Capture: Beyond Device Polling," in *Proc. SANE*, vol. 9, 2004.

[13] L. Deri, J. Gasparakis, J. Waskiewicz, Peter, and F. Fusco, *Wire-Speed Hardware-Assisted Traffic Filtering with Mainstream Network Adapters*. Springer US, 2011, ch. 5, pp. 71–86.

[14] R. Elz, R. Bush, S. Bradner, and M. Patton, "Selection and Operation of Secondary DNS Servers," *IETF RFC 2182*, July 1997.

[15] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin, "An Architecture for a Global Internet Host Distance Estimation Service," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 210–217.

[16] L. Garcés-Erice, K. Ross, E. Biersack, P. Felber, and G. Urvoy-Keller, "Topology-Centric Look-Up Service," in *Proc. NGC*, Sep. 2003, pp. 58–69.

[17] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts," in *Proc. ACM IMW*, Nov. 2002, pp. 5–18.

[18] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How Much Anonymity Does Network Latency Leak?" in *Proc. ACM CCS*, 2007, pp. 82–91.

[19] G. Huston, "BGP Routing Table Analysis Reports," http://bgp.potaroo.net, accessed May 26, 2013.

[20] Intel, "Intel Data Plane Development Kit (Intel DPDK)," http://dpdk.org/, accessed May 13, 2013.

[21] M. Jelasity, A. Montresor, and O. Babaoglu, "T-Man: Gossip-Based Fast Overlay Topology Construction," *Computer Networks*, vol. 53, no. 13, pp. 2321 – 2339, 2009.

[22] Know Your Enemy: Fast-Flux Service Networks, http://www.honeynet.org/papers/ff/, accessed May 19, 2013.

[23] D. Leonard and D. Loguinov, "Turbo King: Framework for Large-Scale Internet Delay Measurements," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 430–438.

[24] D. Leonard and D. Loguinov, "Demystifying Service Discovery: Implementing an Internet-Wide Scanner," in *Proc. ACM IMC*, Nov. 2010, pp. 109–122.

[25] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing Internet Coordinate System Based on Delay Measurement," in *Proc. ACM IMC*, Oct. 2003, pp. 129–142.

[26] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, "On the Accuracy of Embeddings for Internet Coordinate Systems," in *Proc. ACM IMC*, Oct. 2005, pp. 125–138.

[27] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane Nano: Path Prediction for Peer-to-Peer Applications," pp. 137–152, 2009.

[28] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An Internet Distance Estimation Service for Large Networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 6, pp. 2273–2284, Dec. 2006.

[29] P. Mockapetris, "Domain Names – Concepts and Facilities," *IETF RFC 1034*, Nov. 1987.

[30] Y. Morishita and T. Jinmei, "Common Misbehavior Against DNS Queries for IPv6 Addresses," *IETF RFC 4074*, May 2005.

[31] NetMarketShare, http://netmarketshare.com/, accessed Jun. 9, 2013.

[32] T. Ng and H. Zhang, "Predicting Internet Network Distance With Coordinates-Based Approaches," in *Proc. IEEE INFOCOM*, vol. 1, 2002, pp. 170–179 vol.1.

[33] Open DNS Resolver Project, http://openresolverproject.org/, accessed Apr. 7, 2013.

[34] Pagekite, "DNS-Based Authentication," http://pagekite.net/wiki/Howto/DnsBasedAuthentication, accessed May 24, 2013.

[35] A. H. Payberah, "Live Streaming in P2P and Hybrid P2P-Cloud Environments for the Open Internet," Ph.D. dissertation, KTH, 2013.

[36] PlanetLab, http://www.planet-lab.org/, accessed Apr. 2, 2013.

[37] P. Porras, H. Saidi, and V. Yegneswaran, "Conficker C Analysis," *SRI International*, 2009.

[38] Riverbed Technology, "Ethernet Packet Capture and Injection with Turbocap," http://www.riverbed.com/us/products/cascade/wireshark_enhancements/turbocap.php, accessed Apr. 17, 2013.

[39] Route Views, http://www.routeviews.org/, accessed Oct. 25, 2012.

[40] M. Smith and D. Loguinov, "Enabling High-Performance Internet-Wide Measurements on Windows," in *Proc. PAM*, Apr. 2010, pp. 121–130.

[41] Teldat GmbH, http://www.teldat.org/, accessed May 24, 2013.

[42] Untangle, http://www.untangle.com/, accessed May 24, 2013.

[43] WinPcap: The Windows Packet Capture Library, http://www.winpcap.org/, accessed May 13, 2013.

[44] H. Yamamoto and K. Yamazaki, "An Issue of Network Coordinate System-An Impact and Evaluation of Temporal Latency Variation," in *Proc. IEEE ACT*, 2013, pp. 718–723.

[45] B. Zhang, T. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space," in *Proc. ACM IMC*, Oct. 2006, pp. 85–98.