UNIVERSITY OF CALGARY

# Characterizing D2L Usage at the U of C

by

Sourish Roy

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 2017

# Abstract

Over the last decade, online Learning Management System (LMS) services have been utilized by many universities. Desire2Learn (D2L) is the official LMS used by the University of Calgary (U of C). Every student, teaching assistant, and faculty member has access to D2L services.

This thesis presents a workload characterization study of the D2L Web site for on-campus and off-campus users based on a period of two calender years, 2015 and 2016. D2L mainly provides online learning services, delivers course content, and monitors student progress. It uses content delivery networks consisting of geographically dispersed nodes. Persistent and parallel connections are used extensively throughout D2L sessions with users.

This thesis sheds light upon the usage of modern LMS services like D2L. It utilizes network-level data for an extended period of time. Our measurement results highlight the impacts of network latency on the user-perceived D2L performance at the U of C.

# Acknowledgements

Firstly, I would like to sincerely thank my supervisor, Dr. Carey Williamson, for sharing his wisdom and knowledge with me, for his kindness, and his insightful suggestions, which made this thesis possible. His constant guidance allowed me to improve my technical writing skills as well. I would like to acknowledge Yang Liu (Marvin) for helping me in difficult times, and D'Arcy Norman for providing me with valuable information regarding the TITL Web server, internal D2L functionalities, and the CAS server details. I am also indebted to a special friend in India for constantly supporting me. I would like to thank all my friends at the U of C as well. I am grateful to U of C for offering me this great opportunity to learn and to explore the field of Computer Science. Finally, I would like to thank my family, especially my parents Swarup and Suranjana for their tremendous support. Thank you for believing in me, for respecting my decisions, and for encouraging me throughout my life.

# Table of Contents

# List of Tables

# List of Figures and Illustrations

# List of Symbols, Abbreviations and Nomenclature

| Symbol | Definition |
|---|---|
| ADR | Average Data Rate |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| ASN | Autonomous System Number |
| CAS | Central Authentication Server |
| CANARIE | CAnadian Network for Advancement of Research, Industry, and Education |
| CDF | Cumulative Distribution Function |
| CDN | Content Delivery Network |
| CIDR | Classless Inter-Domain Routing |
| CSS | Cascading Style Sheets |
| DHCP | Dynamic Host Configuration Protocol |
| DNS | Domain Name Service |
| D2L | Desire2Learn |
| GUI | Graphical User Interface |
| HDD | Hard-Disk Drive |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ICMP | Internet Control Message Protocol |
| IP | Internet Protocol |
| ISP | Internet Service Provider |
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |

| | |
|---|---|
| LLCD | Log-Log Complementary Distribution |
| LMS | Learning Management System |
| MAC | Media Access Control |
| NAT | Network Address Translation |
| OS | Operating System |
| PAT | Port Address Translation |
| PoP | Points of Presence |
| PPP | Point-to-Point Protocol |
| RAM | Random Access Memory |
| RTT | Round Trip Time |
| SSD | Solid-State Drive |
| SSL | Secure Socket Layer |
| TCP | Transmission Control Protocol |
| TFDL | Taylor Family Digital Library |
| TITL | Taylor Institute for Teaching and Learning |
| TLS | Transport-Layer Security |
| UDP | User Datagram Protocol |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| U of C | University of Calgary |
| VLAN | Virtual Local Area Network |
| WAN | Wide Area Network |
| WLAN | Wireless Local Area Network |
| WWW | World Wide Web |

# Chapter 1

# Introduction

The Internet has undergone many changes since its inception. Usage by Internet users has drastically increased, and continues to grow. The World Wide Web has changed the way we exchange and use information. With cloud computing, Virtual Private Networks (VPN), social networking Web sites, Content Delivery Networks (CDN), and Learning Management Systems (LMS), the linked data available on the Internet has seamlessly shared concepts and software, which is extremely useful for research, education, entertainment, and innovation.

This thesis examines the usage of the Desire2Learn (D2L) LMS at the University of Calgary (U of C). More specifically, we perform a network traffic measurement study, generate usage statistics, identify how network resources are utilized, and suggest potential improvements for the LMS system.

## 1.1 Motivation

In computer networking, traffic measurement and analysis are crucial to the design, operation, and maintenance of local and wide-area networks that are a part of the Internet. This area of research has matured over the years and is used extensively in both academia and industry. As industry and academic networks continue to expand, it becomes necessary to understand the different types of traffic flowing through specific networks.

By collecting network traffic measurement data, we can assess the usage of a network. This in turn helps us in comprehending the traffic characteristics on an existing network, and possibly developing improved models of traffic for network simulation of future networks.

In the context of computer networking, workload characterization of traffic is of particular importance. It helps identify the type of traffic generated, as well as its growth and evolution over

time.

The Web traffic usage pattern typically shows that downloads exceed uploads. In an academic setting, this property also holds since the course instructors mainly upload content, while many students download course content and lectures. With the advent of LMS systems like D2L, the process has become much more structured for instructors, teaching assistants, and the students, as detailed in Chapter 4.

Since D2L is the official LMS adopted by U of C, every instructor and student on-campus can access D2L. Thus, thousands of on-campus users are using D2L every day to upload / download content, to record / view lectures, to upload / view grades, to read / write feedback, etc. These activities generate large volumes of network traffic using thousands of connections, from a variety of IP addresses, and through multiple devices. The traffic patterns for D2L help us understand its impact on the learning environment at the U of C. These patterns allow us to monitor peak traffic hours, heavy tails for session durations, and also understand if traffic is human-generated or machine-generated. The workload characterization of D2L traffic at U of C forms the basis of our study.

Anecdotal reports suggest that D2L is 'slow' (perhaps because its content is hosted remotely in Ontario, Canada). An analysis of the user perceived performance of D2L at U of C could allow us to be aware of the reason behind this slow behavior. A specific concern about D2L performance is that a file upload process is much slower than a file download process. Figuring out the TCP version in place would enable us to suggest an improved version and refine D2L performance for all of its clients.

## 1.2 Brief Overview of the History of D2L

D2L was started by John Baker [65] in 1999 as a system to manage courses and student learning. The company is headquartered in Kitchener, Ontario, Canada, and has more than 800 employees in the US, Canada, the UK, Brazil, Singapore, and Australia [72].

In 2004, D2L added support for Competency-based Learning [67], where concrete skill development was given priority over abstract learning. They introduced ePortfolio to its existing system [21]. Between 2011 and 2013, D2L acquired several smaller companies, namely Captual Technologies and Knowillage Systems.

In 2014, D2L rebranded itself and created an 'Integrated Learning Platform' known as Brightspace [23], where student usage statistics are collected to make adaptive learning possible. They created an open ecosystem including integrations with several other corporate organizations, including IBM, and the Smarter Planet initiative for Education. Wiggio, Microsoft Office 365, and Google Apps became integrated components of D2L, which allowed sharing of available education tools when appropriate. Moreover, D2L launched its mobile application "Brightspace Binder", which allows students to access content on essentially any mobile device. This app is available in the Google Play Store (Android), and the App store (iOS). It allows students to add course content from the D2L learning environment, or directly import resources from external sources into Brightspace Binder [20]. The company name was recently changed to D2L Corporation.

Many universities all over North America use D2L. It provides instructors, students, and teaching assistants (TAs) a platform to exchange content and provide useful feedback. A student can check grades for enrolled courses, upload assignments, and view announcements made by the instructors or TAs, all on a single platform. The system is Web-based, and provides easy access to course content and for viewing classroom statistics.

## 1.3 Historic Context for D2L at U of C

In May 2014, U of C adopted D2L, which replaced the previous LMS named Blackboard. According to the U of C Web site, initially almost 40 courses from various faculties were transferred to D2L in a pilot program led by the Faculty of Science and the Faculty of Education. Soon after that, D2L was broadly used by the remaining disciplines at the U of C, making it the official LMS used on-campus.

Initially, we started our analysis using data from the year 2016, and then looked back in time to understand changes in D2L usage since 2015. This longitudinal study is detailed in Chapter 5.

## 1.4 Objectives

The primary objectives of this thesis are to answer the following research questions:

1. How does D2L work 'under the hood'?
2. How is D2L traffic distributed at the U of C?
3. How can the user-perceived D2L performance be improved?

## 1.5 Contributions

This thesis makes the following contributions:

1. We analyze D2L in detail to understand the client-side and server-side structure.
2. We use Wireshark [82] and Bro [14] to understand the D2L session structure.
3. We identify the role of NAT, DHCP, and two other on-campus servers, CAS and TITL, in the typical Internet path for on-campus and off-campus D2L users.
4. We characterize typical D2L usage in 2016, and compare it to D2L usage in 2015.
5. We use the `mitmproxy` software to analyze HTTPS traffic.
6. We provide recommendations for D2L and U of C to improve D2L performance.

## 1.6 Thesis Structural Overview

The thesis is organized as follows. Chapter 2 presents background knowledge on computer networking including TCP/IP, DNS, HTTP, and HTTPS protocols, and discusses related work

on the relevant domains of network traffic measurement and workload characterization. Chapter 3 discusses the active and passive measurement tools used in this thesis, including the network monitor infrastructure and the Bro logging system. Chapter 4 analyzes the network traffic of D2L from a microscopic point of view. Specifically, we explain the D2L IP address space, server infrastructure and software, cognate infrastructure, network, server and client side, and the role of NAT and DHCP servers. Chapter 5 analyzes the network traffic of D2L from a macroscopic point of view. Specifically, we characterize and compare traffic between two calendar years (2015 and 2016). Chapter 6 provides recommendations for improved D2L operation. Chapter 7 summarizes results, presents conclusions, and outlines directions for future work.

# Chapter 2

# Background and Related Work

This chapter provides an overview of the background knowledge underlying the concepts and technologies involved in our research. Section 2.1 briefly introduces Learning Management Systems. Section 2.2 describes Content Delivery Networks. Section 2.3 explains the widely-used TCP IP protocol stack and also discusses UDP. Section 2.4 discusses the Web, and protocols involved in the application layer, like HTTP and HTTPS. Section 2.5 presents prior related work produced in the field of network traffic measurement. Finally, Section 2.6 summarizes the chapter.

## 2.1 Learning Management Systems (LMS)

Learning Management Systems have recently gained widespread popularity. This technology is used by almost every university all across North America. This form of learning aims at augmenting the more traditional form of classroom learning by engaging in online training and education. The concept of e-Learning has been around for a long time, but it wasn't until the early 2000's that the implementation of LMS saw a rise in demand. Prior to their migration to the Internet, they were typically closed systems for single institutions.

An LMS is essential for managing learning activities for courses online. It provides instructors with techniques to create and deliver content. It also allows them to check student participation and involvement in course curriculum, and assess student performance in assignments, midterms, and tests.

D2L is one such LMS, which has been employed by the U of C to perform the foregoing learning tasks. There are various roles for users to select from (i.e., instructor, teaching assistant, and student), which we later discuss in detail in Chapter 4.

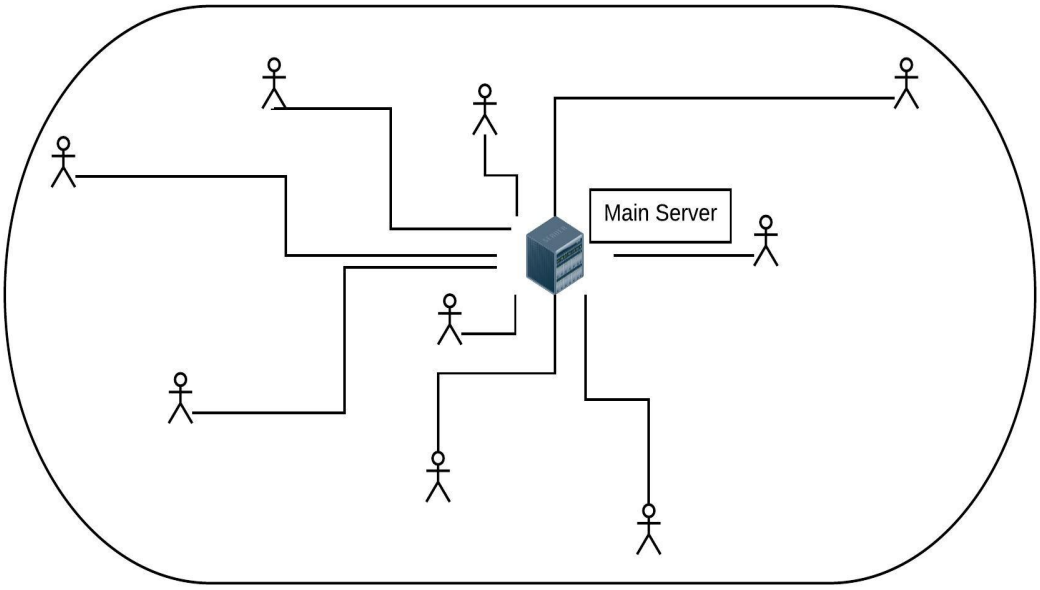
D2L uses the Brightspace Cloud Content Delivery Network (CDN) [13]. It reduces latency, and

improves the D2L application's load time on browsers. Their CDN stores static versions of D2L's user interface components like images, CSS, HTML, Java Script, and JSON resources. These resources rarely change, and are stored in the CDN [12]. This makes caching easy.

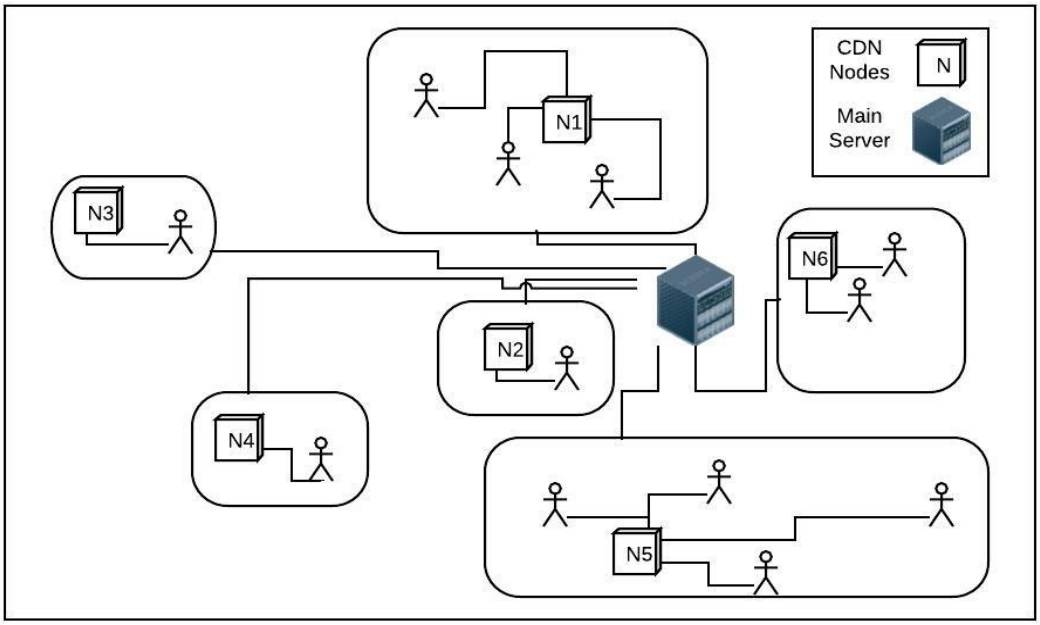
The Brightspace Cloud contains multiple CDN nodes. When users login to the D2L page from a browser, they connect to the geographically closest CDN node, and obtain the static resources (e.g., logos, images, CSS, HTML, etc) which do not change frequently. However, all the D2L course data is obtained from the application's hosted service located in Toronto. In the next section, we detail what a CDN is and how it works.

## 2.2 Content Delivery Network (CDN)

A CDN is a system comprised of a collection of distributed servers forming a network, which is designed to deliver Web content efficiently to an institution or even a single user based on the geographic location of the institution / user. Content is replicated across multiple servers. A CDN tries to reduce the distance between clients and servers by storing cached versions of the same content (in the case of D2L, these are the static resources) in several locations that are geographically dispersed. This is achieved through Points of Presence (PoPs) [68]. Each PoP is responsible for delivering relevant content to its clients within a certain proximity with the help of caching servers. The PoPs that are nearest respond to the D2L request first, and reduce the round trip time (RTT) for clients.




(a) Without Content Delivery Networks




(b) With Content Delivery Networks

Figure 2.1: A Network without CDN versus a Network with CDN

The CDN contains several caching nodes. These caching nodes in turn facilitate the storage and delivery of cached files, and mainly reduce bandwidth consumption and network latency, thereby accelerating the Web site load times. These CDN caching servers possess numerous storage hard drives and large amounts of random-access memory (RAM) resources. There are several ways of storing these cached files inside the caching servers, namely RAM, hard-disk drives (HDD), or solid-state drives (SSD). The files that are accessed most frequently are generally stored in the fastest medium i.e., the RAM [17].

Figure 2.1 provides a conceptual illustration of a CDN. Figure 2.1a shows a network that does not use a CDN. It relies directly on the main server irrespective of the geographical locations of clients. This results in poor latency and high bandwidth consumption. Figure 2.1b shows a network that uses a CDN with multiple geographically dispersed nodes deployed over several locations. It allows users in a particular location to access the CDN in that location rather than accessing the main server each time, which saves bandwidth. Since a CDN node acts like a proxy cache there is however a one-time cost for the initial download of the requested content before storing it in its cache. CDNs provide higher content availability, and better performance. On the Internet, CDNs might be operated by third-party providers like Akamai or Limelight.

## 2.3 TCP/IP Model

The modern Internet has evolved greatly following its origin from the ARPANET project [1]. The Internet is a well-established system of computer networks that are interconnected using communication protocols from the TCP / IP protocol stack shown in Figure 2.2. This model represents how networks function [40].

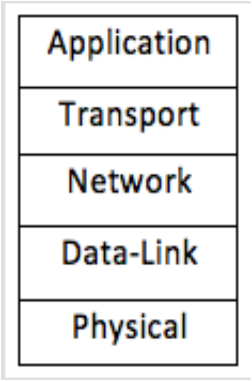



Figure 2.2: The Internet TCP/IP Protocol Stack

Since the Internet TCP/IP protocol suite is the current globally-deployed protocol suite, we discuss the TCP/IP model in detail in this chapter. From the bottom to the top, the five-layer model contains the Physical, Data-Link, Network, Transport, and Application layers. See Figure 2.2 for an illustration of the five-layered TCP/IP protocol stack. TCP/IP can also be used in private networks (intranet or extranet).

The reason behind the TCP/IP nomenclature is the importance of TCP and IP at the Transport and Network layers, respectively. The TCP layer is responsible for the breaking down and reassembly of packets into a message. The lower layer, IP, contains the address portion of each packet so that the correct destination is reached. Routers in a network are responsible for forwarding these packets by checking the source / destination address.

### 2.3.1 The Physical Layer

The physical layer contains the necessary functions to transmit the raw bit stream over a physical medium [40]. The transmission media forms a bit pipe. This layer specifies mainly four parts: mechanical, electrical, functional, and procedural [18] [34]. The physical layer is a combination of both hardware and software programming and might include a few electromechanical devices. For example, this layer is assigned the role of deciding how to put a bit stream from the data link (upper) layer to an optical fiber transmitter, a radio carrier, or several other physical interfaces.

### 2.3.2 The Data-Link Layer

The Data-Link layer moves frames across links between parts of the network [40]. It is a combination of methods and communications protocols that function on a physically-connected host link. A few commonly-used link-layer protocols include Ethernet, Wi-Fi, and Point-to-Point Protocol (PPP). Datagrams sent from the upper layers may be transported through various physical and link protocols [40]. The link layer is responsible for conveying bits across the link and might help in making the raw physical communication link more reliable. It usually adds a header as well as a trailer to the data provided by the Network layer. The data-link layers are mainly responsible for delivering data between directly connected systems, which are just one hop away (adjacent systems). This thesis does not focus on the information from the physical and data-link layers.

### 2.3.3 The Network Layer

The biggest difference between the Data-Link layer and the Network Layer is that the Network Layer can deliver datagrams to systems that are not directly connected to the source. This layer sends data in the form of packets from the source to destination across multiple links, and allows them to move between different hosts [40]. In order to know where packets originate, the network layer uses a network address, which is commonly known as the IP address. The main protocol used presently is the Internet Protocol Version 4 (IPv4) [51][75]. This IP address is a form of logical addressing, and is a series of four octets, represented as `a.b.c.d`, where every octet is a number between 0 and 255.

The Network layer relies upon routing tables inside routers to store information on how to reach destination systems. A router is basically an intermediate system that acts at the network layer and interconnects network IDs [16]. They filter and forward packets based on IP addresses, whereas link-layer switches filter and forward packets based on MAC addresses. Routers do not care about where a packet came from, only where it is going. This routing takes place using the 'best' suitable path [40]. The use of these routing tables for packet delivery is termed 'forwarding',

which takes place hop by hop. IPv4, however, only provides best-effort delivery, and it does not guarantee the reliable delivery of packets. In the network layer, each host has a unique IP address.

Classless Inter-Domain Routing (CIDR) is used within routing tables inside the routers across the Internet, and it helps reduce the exhaustion of IPv4 addresses [29] [66]. The CIDR notation is used to manage IP addresses in networks more easily, and is a shorthand method of representing a range of IP addresses. In an IPv4 network, a chunk of addresses is assigned to an institution, and the blocks are written in the following way: `a.b.c.d/x`. The '/x' denotes the 'block' of addresses assigned, which are also referred to as a subnet. For example, the CIDR subnets for D2L are 199.30.176.0/21 and 174.90.126.0/23. The '/21'subnet means that D2L can use any of the 2048 IPs assigned to it between 199.30.176.0 and 199.30.183.255. The D2L IP used for the U of C is 199.30.181.42. The '/23' subnet denotes that D2L has also been assigned 512 total host IPs between the range of 174.90.126.0 and 174.90.127.255. Thus, 2560 total IPs are available for D2L to select from and assign to each of its client universities. We discuss the D2L IP address space further in Chapter 4.

Network Address Translation (NAT) is a process by which network devices (firewalls, routers, etc) assign a public IP address to a group of computers within a private network [76]. NAT is useful in reducing the total number of public IP addresses an institution uses. We performed an in-depth analysis for network traffic in this thesis based on the IP information extracted from the network layer.

### 2.3.4 Transport Layer

The main task of the transport layer is process-to-process delivery. It is necessary to determine which process should receive the packet's content. In addition, transferring large files in a single packet isn't possible. Since data units might exceed the maximum allowable size of a packet, 'segmentation' becomes necessary, which is the process of dividing a message into several packets. The Network layer is responsible for forwarding each and every packet, and is unaware of the possible relationships between these packets. In contrast, the transport layer is responsible for

making sure that the entire message is delivered intact. This is achieved through flow control and error checks in the transport layer [74]. The protocol that performs all of these functions is TCP.

There are, however, two possible protocols at the transport layer [40]:

1. UDP: A connectionless protocol that is 'unreliable' and does not guarantee ordered delivery of data.

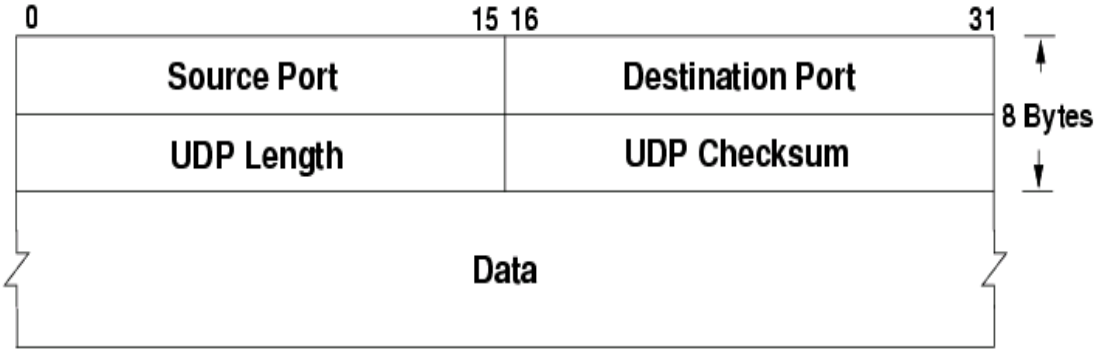
2. TCP: A connection-oriented protocol that provides 'reliable' service and ordered delivery of data.

Figure 2.3 shows that the UDP header size is 8 bytes, whereas the TCP header size is 20 bytes [63][64]. In the following subsections, we explore UDP and TCP, and discuss the role of port numbers.

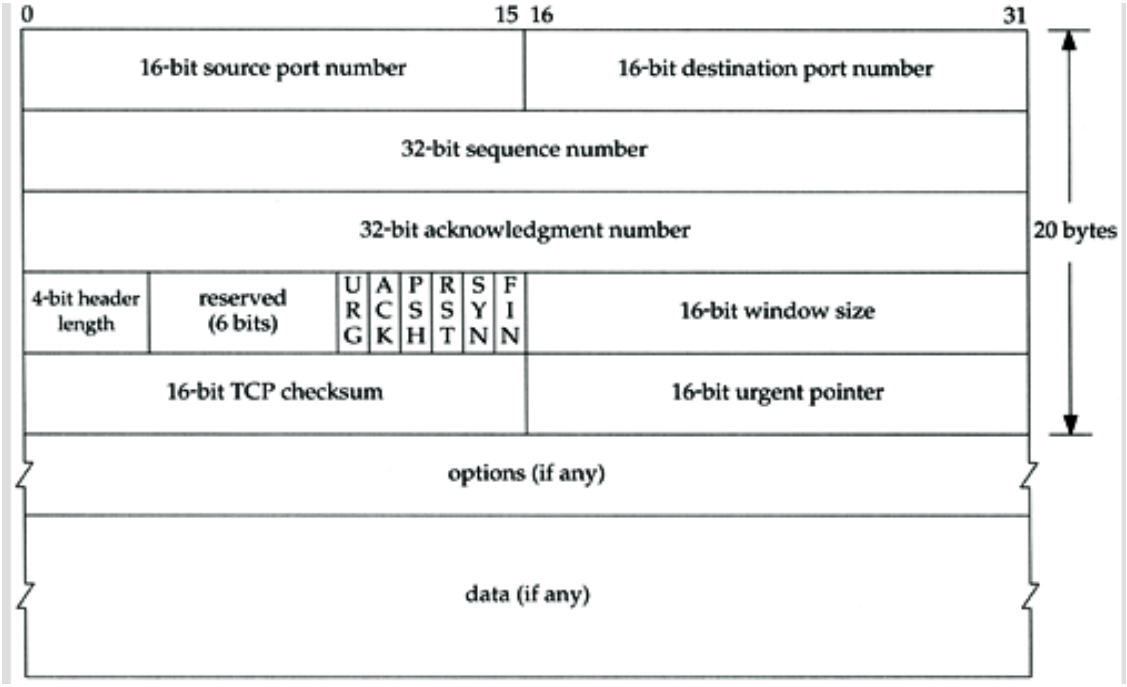
**UDP**

The primary use of UDP is for low-latency and loss-tolerant applications on the Internet. It is ideal for simple applications where purely connectionless data delivery is sufficient. Single request-response pairs of messages are sent more effectively since there is no need to exchange a series of opening TCP segments for connection establishment. It is used for real-time applications, streaming content, online gaming, and video or voice communications [78].

UDP is a connectionless protocol, and does not have any hand-shaking model for guaranteeing reliability, data integrity, or ordering. It is unreliable, and might lead to missed datagrams or datagrams arriving out of order. UDP is not concerned with error checking, and presumes that it is either not necessary, or already performed in the application. UDP is frequently utilized for short transactions that fit into one datagram. TCP offers more reliability features than UDP, which we shall see next.




(a) UDP header




(b) TCP header

Figure 2.3: Packet Headers for UDP and TCP [63][64]

**TCP**

TCP is the dominant protocol for the majority of Internet applications. It can break large data files into separate packets, check, resend lost packets, and reassemble packets in the proper sequence [52]. Figure 2.3b illustrates a TCP header with various fields such as packet sequence numbers, acknowledgement number, window size, source and destination port numbers, several flags, and various other options. Synchronize and Acknowledge messages are indicated by the SYN bit and the ACK bit, respectively, within the TCP header. The sequence and acknowledgement numbers help in keeping track of the transmitted data. TCP has reliability features like sequence numbers, which detect missing or out-of-order segments, and also flow control features to prevent senders from overwhelming a receiver. These additional services add overhead, and cause delays. That is the reason why TCP is avoided for real-time streaming of audio or video over the Internet.

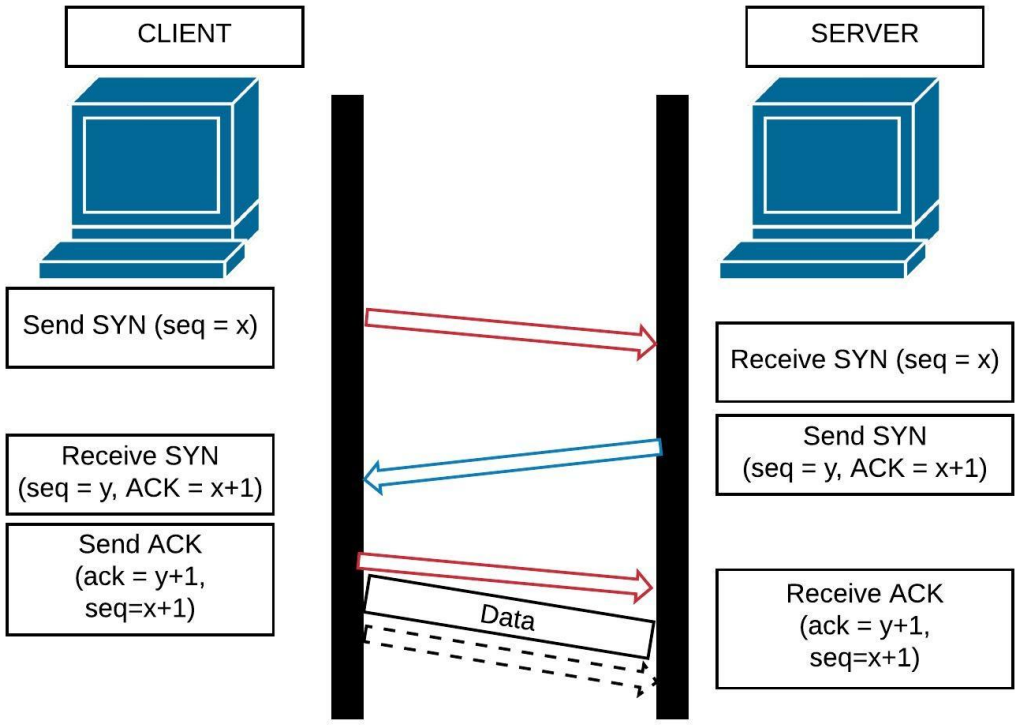



Figure 2.4: TCP Three-Way Handshake Procedure for Connection Establishment

TCP uses a three-way handshake mechanism to establish a logical connection on an IP-based

network. The TCP SYN handshake packet exchange sequence is illustrated in Figure 2.4. TCP's handshaking mechanism includes a SYN, SYN-ACK, and ACK, which are three messages transmitted by TCP to begin a TCP session between two computers [52]. This handshaking technique is developed in such a way that two hosts can negotiate the parameters of the TCP socket connection in a network before sending data.

This also allows both parties to initiate multiple TCP socket connections simultaneously, which in turn makes it possible for single physical interfaces like the Ethernet to be multiplexed to transmit several streams of data concurrently.

The connections are terminated by a special message with the FIN (finish) bit set. A connection is not terminated entirely until both computers have completed the procedure by transmitting a FIN and accepting an ACK.

**Port Numbers**

In TCP and UDP, port numbers are a part of the addressing information used to identify the transmitters and recipients of messages. They allow multiple applications on the same device to share network resources concurrently, and primarily assist in the transmission of data between the network and an application. In case of an incoming datagram, the IP address helps in identifying the destination computer, but the port number further identifies the intended destination program in that computer. Likewise, outgoing datagrams contain application port numbers inside the packet header that allows the receiver to distinguish between several specified applications. These 16-bit numbers range between 0 and 65,535 [80]. Port numbers less than 1,024 are reserved for specific services or applications, while numbers greater than 1,024 are unreserved port numbers. Whenever a client application does not explicitly request a specific port number, that number is an ephemeral port number. These ephemeral ports are ports assigned by a computer's OS or IP stack [70].

Table 2.1 shows some well-known port numbers. Although some applications might change their port numbers, a few frequently-used Internet services are allocated with port numbers that are globally recognized. For example, port 80 is for HTTP, 443 for HTTPS, 23 for Telnet, 25 for

Table 2.1: Well Known Ports

| Transport Protocol | Port Number | Service |
|---|---|---|
| TCP | 22 | SSH |
| TCP | 23 | Telnet |
| TCP | 25 | SMTP |
| TCP | 80 | HTTP |
| TCP | 443 | HTTPS |
| UDP | 53 | DNS |
| UDP | 67/68 | DHCP |

SMTP, etc. This thesis mainly focuses on HTTP, HTTPS, DNS, and DHCP connections handled by ports 80, 443, 53, and 67/68, respectively.

### 2.3.5 Application Layer

The uppermost layer of the TCP/IP protocol stack is the Application layer. Important protocols like HTTP, SMTP, and FTP [40] are included here, and well-known applications include file transfer, Web surfing, voice over IP (VOIP), Web chat, Email, network data sharing, virtual terminals, and several others. The application layer consists of protocols focused on providing client-server or peer-to-peer communication across a network, on any end-user device. Communications are sometimes encrypted at this level, using either TLS [24], or its predecessor SSL [77]. This thesis shows an extensive traffic analysis from the HTTP and SSL logs recorded for this layer.

## 2.4 The Web, HTTP, and HTTPS

Following Tim Berners-Lee's creation of the World Wide Web (WWW) [9], the networking world saw a change, and the WWW quickly became the primary method of exchanging data over the Internet. By the end of 1990, three fundamental technologies that still remain the foundation of today's Web technology were established:

1. HTML: HyperText Markup Language. This is the formatting markup language for the Web. It focuses on how Web pages are designed and displayed.

2. URI: Uniform Resource Identifier, commonly known as a URL (Uniform Resource Locator). This is a form of addressing used to identify individual resources on the Internet. For example, an HTTP URL may look like `http://google.com/`.

3. HTTP: Hyper Text Transfer Protocol. This application-layer protocol allows the retrieval of linked resources across the Internet, using a simple request-response paradigm.

**Hyper Text Transfer Protocol (HTTP)**

HTTP is one of the most useful protocols used by the World Wide Web [28]. It defines how messages are formatted and transmitted, and the actions Web servers should take to respond to commands. The client-side implementation of HTTP takes place through a Web browser (for example, Google Chrome, Mozilla Firefox, Microsoft's Internet Explorer, etc). The Web server is a computer system that processes such client requests via HTTP [79]. When a URL is entered in a browser, a HTTP command is sent to the Web server with instructions to fetch and transmit the Web page that was requested.

In some scenarios, an HTTP proxy server is deployed. In computer networks, a proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers [41]. The main benefit of having a proxy server is that its cache can serve several other users. When one or more Web sites have many requests for the same content, that content can be stored in the proxy's cache. This improves the user response time.

HTTP is a stateless protocol since each command is run independently, without prior knowledge of the previously executed commands. For interactive Web sites, a number of new technologies, like Java, JavaScript, cookies, and others are often used.

HTTP has two main versions. HTTP/1.0 [10] is an older version, and HTTP/1.1 [27] [39] is a revised version of HTTP/1.0. A new version, HTTP/2.0 [7], supports header compression,

priority, query multiplexing, and more intelligent packet streaming management. This reduces latency and accelerates content download on modern Web pages. However, HTTP/2.0 is not yet widely supported. The most prevalent version of HTTP on today's Internet is HTTP/1.1.

A few advantages of HTTP/1.1 over HTTP/1.0 are shown below:

1. HTTP/1.1 allows persistent connections. This is the default mode for all HTTP/1.1 connections, while HTTP/1.0 generally supports non-persistent connections. HTTP/1.0 can allow persistent connections by adding a message header 'Connection: Keep-Alive', and it has no compatibility issues with HTTP/1.1 servers [26]. Persistent HTTP connections reuse the existing TCP connection while non-persistent HTTP does not [40]. This means that HTTP/1.1 allows more than one request/response pair on the same TCP connection, and for HTTP/1.0 a new TCP connection needs to be established for every request/response pair. When a response is received, the HTTP/1.0 connection is closed. This is inefficient due to TCP slow start, which controls the amount of outstanding data being injected into the network [2].

2. HTTP/1.1 has a specific Host header, which is useful for proxies, whereas HTTP/1.0 does not officially have one.

3. The method in the HTTP request helps in identifying the action to perform on the object identified by the request URL [27]. In HTTP/1.0, there are only three methods: GET, POST, and HEAD. In HTTP/1.1, several new methods have been added, namely OPTIONS, PUT, DELETE, TRACE, and CONNECT.

4. HTTP/1.1 uses 'entity tag', and expands its caching support. For a pair of identical resources, the entity tags are the same. It can additionally have a Cache-Control header that defines a caching policy. HTTP/1.0 supports caching with the conditional GET mechanism, using the 'If-Modified-Since' header.

5. HTTP/1.1 supports additional features like new status codes, chunked transfer en-

coding, new connection headers, digest authentication, proxy authentication, enhanced transfer encoding, and more.

Figure 2.5 shows an HTTP request header while visiting the UCalgary Web site.

```
GET / HTTP/1.1
Host:  www.ucalgary.ca
User-Agent:  curl/7.43.0
Accept:  */*
```

Figure 2.5: HTTP Request Header

Each HTTP request consists of three sections, namely the request line, headers, and an optional message body. The request line begins with a method specification, which is a 'GET' in this case. Most of the HTTP requests use the 'GET' method. A GET request is used to download a specified resource (like a page, image, etc) from the server. There are other request methods as well, like HEAD and POST. A HEAD request is similar to GET, but it transfers only the header section containing meta-data, and status line without any body section. POST requests are useful for passing some sort of information (form data, or a file) that needs to be uploaded to the server.

The method specification is followed by the host information, which shows the URL representation of the Web site requested by a user. There can also be additional header fields. In the example above, the user-agent information shows 'curl/7.43.0' [69], which is a command-line utility for transferring files over the Web. The user-agent field represents client operating systems and browsers. It provides information on the software program used by the client. In the final line, we see an 'Accept' header, which might vary from request to request [62]. However, since there is nothing mentioned with 'Accept', we can assume that text/plain and text/html are accepted.

The client might pass additional data regarding the request, and about the client itself, to the server through these Request-header fields. These Request-header fields act as request modifiers [60]. Following is a list of some important ones that might be present in the headers:

1. Referer: Generates back-links to documents. When someone visits a Web site, a piece of information in the form of a URL records where that person came from. For example, a user selects a link on a Web page they are presently on, which then brings them to a target Web site, like D2L. This piece of information for any target Web site is known as the referer URL.

2. Wildcards: Figure 2.5 shows a Wildcard entry for the foregoing request example. This allows clients to receive content types of which they might be unaware. An asterisk '*' may be used in place of the content-type value [62].

3. If-Modified-Since: It is used with GET method to make it conditional. If the requested document has not changed since the specified time in this field, then the document is not sent, but instead a `Not Modified 304` reply is sent [62].

**HTTP Response**

Figure 2.6 shows an example of a HTTP response received from the UCalgary Web site. The first line of a server response is a Status-Line consisting of the protocol version followed by a numeric status code. In the response example, the server supports HTTP version 1.1. In this case, the status code is `200 OK`.

Status codes are three digit numbers. The first digit of the code defines the response class. A few other status codes are 1XX: Informational (the request was received and the processing is continuing), 2XX: Success (the action was successfully received, deciphered, and accepted), 3XX: Redirection (further action is required to complete the request), 4XX: Client Error (request consists of incorrect syntax and cannot be fulfilled), and 5XX: Server Error (server failed to fulfill a valid request) [61].

```
HTTP/1.1 200 OK
Date: Wed, 12 Jul 2017 17:12:28 GMT
Server: nginx
Cache-Control: public, max-age=1800
Content-Type: text/html; charset=utf-8
Link: <http://www.ucalgary.ca/>;
Content-Language: en
Access-Control-Allow-Origin: *.ucalgary.ca
Accept-Ranges: bytes

<html>
. . .
</html>
```

Figure 2.6: HTTP Response Header

In our example, the status line is followed by the date the response was received, and the server name, which is `nginx` for the `www.ucalgary.ca` Web site. `Cache-Control` is the cache policy used, and caching is the ability to store and reuse previously fetched resources to optimize performance. `English` is the content language described in the field `Content-Language`, and `Content-type` shows the type of content in the requested Web page, which in this case is `text/html` with `charset=utf-8`.

**HTTPS**

Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP [53]. It is the protocol over which data is sent between a user's browser, and a secure Web site. The 'S' at the end of HTTPS denotes 'Secure'. HTTPS encrypts all communications between a browser and a Web site to make every transaction protected and confidential. A padlock icon in the top left corner of the address bar of popular Web browsers such as Chrome, Internet Explorer, and Firefox is a visual indication of a HTTPS connection. The URL identifiers for HTTPS begin with 'https://' instead of the standard 'http://'. HTTPS is also called HTTP over Transport Layer Security (TLS) [53], HTTP over SSL, and HTTP [31]. The TLS and SSL use an asymmetric system with 'public' and

'private' keys to encrypt communications. The 'private' key is strictly protected and should be accessed only by the key owner. The 'public' key is distributed to everyone to decrypt information that was previously encrypted by the corresponding 'private' key. For this thesis, whenever we see a TLS or SSL handshake over port 443, we consider it as HTTPS service.

## 2.5 Related Work in Network Traffic Measurement

Network traffic measurement provides a means to analyze and manage network usage, facilitate network control, and understand network performance. A voluminous amount of research has previously been conducted in measuring and characterizing Internet traffic from the early 1990's to the present day (e.g., Web traffic, peer to peer file sharing, Content Delivery Networks, YouTube, online social networks, Netflix, and more [6] [8] [30] [38] [45] [54] [55]). The main reasons why a network measurement study is useful is because it helps in network troubleshooting, protocol debugging, performance evaluation, and workload characterization.

In 1991, Cáceres et al. [15] performed network traffic measurement research showing that TCP was the most dominant protocol in the Internet at that time. Over 90% of the TCP connections lasted less than a few seconds, and transferred fewer than 10 kilobytes of data. They showed that interactive applications generated ten times more data in one direction compared to the other, with packet sizes between 1 byte and 512 bytes.

In 1994, Paxson analyzed traffic in Wide Area Network (WAN) environments. He used eight month-long traces (November 1990 to April 1994) of all wide-area TCP connections between the Lawrence Berkeley Laboratory (LBL), and the rest of the world [48]. He observed significant growth for TCP traffic (both in terms of number of connections, and bytes transferred). SMTP connections grew 70% per year, and FTP data grew 100% per year. He also mentioned how new protocols exhibit growth during their first introduction, which in turn significantly affected the Internet traffic profile. He also performed geographical profiling of wide-area traffic, which he found was diverse and dynamic, with the countries influencing the traffic changing over time. In his

work, the tcpdump [32] packet capture tool was used extensively for studying growth trends in Internet traffic over time. Tcpdump uses the Berkeley Packet Filter architecture for capturing TCP/IP packets. It captures and filters a network's IP packets by specifying host addresses, protocol types, or port numbers.

Network measurement can be classified based on the placement of the network monitor (edge network or core network), network analysis tools used (hardware-based or software-based), probing mechanism (passive or active), and the number of vantage point perspectives (single viewpoint or multiple viewpoints) [25][81].

A Poisson arrival process exists only if the time between individual events follow an exponential distribution. Paxson and Floyd studied telnet traffic and found that a Poisson arrival process is effective for modeling such traffic, assuming a fixed hourly connection arrival rate [49]. A similar Poisson arrival process was seen by Arlitt and Williamson while modeling user requests for individual Web pages on a Web server [5].

In 1996, Arlitt and Williamson performed an analysis on Web server access logs. These logs record client requests for Web site URLs, type of content accessed, request time, client IP address, and even document size. Six different data sets were used in their study: three from academic (university) environments, two from scientific research organizations, and one from a commercial Internet provider [4][5]. They identified that over 90% of the requests to a Web server result in successful return of a document, and that HTML and image documents account for most of the objects (over 90%) transferred by Web servers. This work also suggests that almost 80% of Web document transfers are less than 10 kilobytes in size, though a significant heavy tail distribution could be seen [4][5]. Finally, on the basis of their findings, they proposed improved caching systems, and cache management strategies like frequency-based replacement. In this thesis, we have found similar heavy-tailed distributions while plotting IP frequency rank, and inbound/outbound data per connection, which is discussed in Chapter 5.

A long-term traffic measurement study was conducted by Newton et al. [46]. They used

the TCP/IP packet headers (1999-2012) in packet traces collected over 13 years on the Internet link that connects the University of North Carolina (UNC) at Chapel Hill campus network to its ISP. They showed how the Internet traffic changed over time, and proposed a novel method for splitting Web traffic over ports 80 (HTTP) and 443 (HTTPS) into Activity Sections. They performed an in-depth analysis of HTTP request sizes and responses, monitored the growing size of Web pages, increased use of cookies, and the referrer fields. They showed how the methodology implemented in [35] was still effective. Specifically, using the limited information available in the TCP/IP protocol headers for one direction of a TCP connection, produced excellent results for characterizing TCP connection-level properties such as request/response sizes, and number of request/response exchanges.

In 2001, Almeida et al. performed an analysis on server log data for educational media servers at two major public United States universities (University of Wisconsin-Madison and University of California, Berkley) [3]. Their paper focused on the eTeach system and the BIBS (Berkley International Broadcasting System), which delivered high quality content. They discussed the overall load on each of the servers, and found a high degree of resemblance in the measures that were obtained from both servers. They illustrated that 500-1800 requests were made to the BIBS servers per weekday. They show that the BIBS server has 500-800 sessions on weekdays, which might occasionally rise to 2000 sessions. The duration of media per session was 15-25 minutes on average. They study the variabilities of requested file sizes and the percentage of files requested. They analyze the request arrival rates for both eTeach and BIBS on different dates. To determine the distribution of time between requests, curve fitting for exponential, gamma, Weibull, and Pareto distributions were used on each server and each file. They analyzed session characteristics, and showed that media file length determines the distribution of the media delivered per session or per request. They show heavier-tailed characteristics for videos delivered per request, which are longer than five minutes, and typically exponential for short videos. Their study provides a benchmark against which future server workloads can be compared.

## 2.6 Summary

This chapter reviewed background knowledge in computer networks. We started by describing the foundation of Learning Management Systems and Content Delivery Networks. Then we discussed the TCP/IP protocol stack, and the application-layer protocols HTTP and HTTPS. Finally, we presented a series of related studies in the domain of network traffic measurement and LMS.

In the next chapter, we discuss our measurement methodology and metrics. We also introduce the tools used for active and passive traffic measurements in this thesis.

# Chapter 3

# Methodology

In this chapter, we provide details on our network traffic measurement methodology. Section 3.1 discusses the traffic logging system, and the hardware/software infrastructure. Section 3.2 provides specific details regarding the trace format by providing real examples from the logs. It also describes passive measurement tools like Wireshark and others, which have been used throughout this analysis. Section 3.3 explains the active measurement tools used, namely Ping and Traceroute. Section 3.4 describes the relevant steps involved in processing our collected data. Section 3.5 discusses ethical considerations regarding protection of user identities and security vulnerabilities. Finally, Section 3.6 summarizes the chapter.

## 3.1 Data Collection and Logging Infrastructure

Our network monitor records information about the inbound and outbound network traffic passing through the university's edge routers. This collection takes place through a mirrored stream of all packet-level traffic passing between the Internet and the University of Calgary network.

The network monitor is a Dell server, which processes the mirrored traffic stream. It is equipped with two Intel Xeon E5-2690 CPUs (32 logical cores @2.9 GHz), 64 GB RAM, and 5.5 TB of local hard disk storage for the logs. The operating system (OS) on this server is CentOS 6.6 x64. Data from the hard disks are backed up overnight to a file storage server with more storage space. The monitor utilizes an Endace DAG 8.1SX for capturing the traffic and filtering it. It was designed for 10 Gbps Ethernet, and uses several programmable functions in the hardware to boost the performance of packet processing. The primary use of the Endace DAG card is to split the incoming traffic into streams for processing by the Bro logging system.

The Bro Network Monitor [47] produces connection-level logs regarding all the visible traffic.

Bro is open-source, and is essentially a network analysis and security framework. The Bro logging system monitors all packet-level network activities and provides detailed summary information regarding transactions. Our primary interest is in the connection, HTTP, and SSL logs. The connection logs provide data regarding each observed connection, such as start time, end time, bytes transferred (inbound and outbound data), duration, and termination state. The HTTP log helps us identify the source and destination IPs, HTTP methods, hosts, URIs, referer URLs, and user agents. Finally, the SSL logs show us HTTPS connections, with fields like timestamps, TLS, or SSL encryption methods, plus source and destination port addresses.

Bro collects and generates hourly logs following the activation of the logging system. The D2L Web site uses secure HTTP. Throughout our analysis, we collect and analyze data from the HTTP, SSL, and connection logs to produce the statistical results reported in this thesis.

**HTTP Logs**

Figure 3.1 shows an example of the HTTP log format generated by Bro. The 'fields' show summary information in various categories. Note that our Bro logs have 27 fields, but only a subset of these fields are required for our analysis in this thesis. The following fields from HTTP logs are of primary interest to us:

1. `ts` denotes the time-stamp of a request (Linux epoch time format).
2. `id.orig_h` is the IP address (32-bit format) of the originating host.
3. `id.resp_h` is the IP address (32-bit format) of the responding host.
4. `id.orig_p` is the source port number in a TCP / UDP header.
5. `id.resp_p` is the destination port number in a TCP / UDP header.
6. `method` denotes the HTTP method in the request.
7. `host` is the name in the header of a Host request.
8. `uri` stands for Uniform Resource Identifier, and indicates the resource name.

9. `referer` is an HTTP header field that indicates the Web page (if any) from which the request was initiated.

10. `user_agent` field shows the user agent (i.e., browser and/or OS) used by the client.

11. `status_code` is the HTTP status code of the response.

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       trans_depth     method  host    uri     referrer
        user_agent request_body_len       response_body_len       status_code     status_msg      info_code       info_msg        filename
        tags    username   password       proxied orig_fuids      orig_mime_types resp_fuids      resp_mime_types
```

(a) HTTP Log Fields

```
1467309609.685839       CqIgeo1MYyHkO99vPb      136.159.49.118  62830   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36    0       0       302     Fou
n(empty)  -       -       -       -       -       -
1467309612.086987       CnygRT3VnAqFtTr8C3      136.159.160.128 49287   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0 0       0       302     Found   -       -       -       (empty) -
   -
1467309653.197628       CNY3Qa33u1E7sqw4Vj      136.159.49.118  64457   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36   0       0       302     Fou
n(empty)  -       -       -       -       -       -
1467309653.593816       CNY3Qa33u1E7sqw4Vj      136.159.49.118  64457   199.30.181.42   80      2       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36   0       0       302     Fou
n(empty)  -       -       -       -       -       -
1467309657.684187       CqFx9023o1HbyZ463b      136.159.49.121  53961   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36   0       0       302     Fou
n(empty)  -       -       -       -       -       -
1467309665.111489       CbdjPbRNwikWX4okk       136.159.49.124  58018   199.30.181.42   80      1       GET     d2l.ucalgary.ca /d2l/orgtool
s/style/colour.css       http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.6.1
7 (KHTML, like Gecko) Version/9.1.1 Safari/601.6.17       0       0       302     Found   -       -       -       (empty) -       -       -
   -
1467309665.208719       CbdjPbRNwikWX4okk       136.159.49.124  58018   199.30.181.42   80      2       GET     d2l.ucalgary.ca /images/ucal
gary-logo.png    http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.6.17 (KHTML
, like Gecko) Version/9.1.1 Safari/601.6.17       0       0       302     Found   -       -       -       (empty) -       -       -       -
   -
1467309665.255590       CqVt7AlYnRt3AEq02       136.159.49.124  58019   199.30.181.42   80      1       GET     d2l.ucalgary.ca /images/body
-bg.png  http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.6.17 (KHTML, like G
ecko) Version/9.1.1 Safari/601.6.17       0       0       302     Found   -       -       -       (empty) -       -       -       -
   -
1467309665.255962       CIpZ4e980XnPlJlde       136.159.49.124  58020   199.30.181.42   80      1       GET     d2l.ucalgary.ca /images/bg.j
pg       http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.6.17 (KHTML, like G
ecko) Version/9.1.1 Safari/601.6.17       0       0       302     Found   -       -       -       (empty) -       -       -       -       -
   -
1467309677.760991       Cy8nHo3SNHKDLD2kp1      136.159.49.114  52225   199.30.181.42   80      1       GET     d2l.ucalgary.ca /d2l/orgtool
s/style/colour.css       http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36    0       0       302     Found   -       -       -       (empty) -       -       -
   -
```

(b) HTTP Log Format

Figure 3.1: HTTP Log Example

### SSL Logs

Figure 3.2 shows an example of the Bro SSL logs. There are 20 fields in the SSL logs. The following fields from SSL logs are of primary interest to us:

1. `id.orig_h` is the IP address (32-bit format) of the originating host.

2. `id.resp_h` is the IP address (32-bit format) of the responding host.

3. `id.orig_p` is the source port number in a TCP / UDP header.

4. `id.resp_p` is the destination port number in a TCP / UDP header.

5. `version` is the TLS or SSL encryption version.

6. `cipher` shows the message digest in string format.

7. `server_name` is the server name of a user-requested Web site.

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       version cipher  curve   server_name     resumed last_alert
next_protocol   established     cert_chain_fuids        client_cert_chain_fuids subject issuer  client_subject  client_issuer   validation_status
```

(a) SSL Log Fields

```
1467310276.322062       Cwe3rG2SxjUSBsgxed      136.159.49.114  34665   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310276.355434       Cc0xhyBeV5lvlkbMf       136.159.49.114  34666   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310276.362385       Cfaznv3FnclUs62kA6      136.159.49.114  34667   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310276.368472       Cbz6pw4RPHXdYLtvPl      136.159.49.114  34668   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310277.164724       CWiTMA2sQ3Vb7Rtjlh      136.159.160.67  56220   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310277.623794       CE4o3HiZ5s1baI117       136.159.49.121  59747   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310278.791970       CwwER33BTBWGmfJKF8      136.159.49.118  56426   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310279.530708       CiIbpb3hohAIMlEt5g      136.159.49.121  59326   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310279.592897       C4gWS13aGJzRDMjMFc      136.159.49.117  52219   199.30.181.42   443     TLSv12  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   -
d2l.ucalgary.ca  T      -       -       T       -       -       -       -       -       -       -
1467310042.039853       C9gx5S3GsjrPV64EA1      136.159.160.73  63010   199.30.181.42   443     -       -       -       d2l.ucalgary.ca F       -
-
```

(b) SSL Log Format

Figure 3.2: SSL Log Example

**Connection Logs**

Figure 3.3 shows an example of the Bro connection logs. There are 20 fields in the connection logs. The following fields from the connection logs are used for this thesis:

1. `uid` stands for unique ID. It provides a means to associate individual HTTP transactions from the HTTP logs with their corresponding TCP connection(s) in the connection logs. This allows us calculate the bytes transferred, persistent HTTP connections during a session, and session durations.

2. `id.orig_h` is the IP address (32-bit format) of the originating host.

3. `id.resp_h` is the IP address (32-bit format) of the responding host.

4. `proto` shows the protocol for IP traffic type (TCP or UDP).

5. `service` shows the type of server used (HTTP, SSL, or DNS).

6. `duration` is the connection duration in seconds.

7. `orig_bytes` is the outbound data volume leaving the U of C network. It assumes that `id.resp_h` is the D2L IP.

8. `resp_bytes` is the inbound data volume entering the U of C network. It assumes that `id.resp_h` is the D2L IP.

For the inbound `resp_bytes` and outbound `orig_bytes` data traffic analysis, the `id.resp_h` (responding host) field is always set to the D2L IP (199.30.181.42). However, the `id.origin_h` field is by default a U of C IP, since the monitor records traffic passing only through the U of C's edge routers, which allows us to calculate inbound and outbound D2L traffic generated only by on-campus users. Our example in Figure 3.3b shows only U of C IPs under the `id.origin_h` field.

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       proto   service duration        orig_bytes      resp_bytesconn_state
      local_orig      local_resp      missed_bytes    history orig_pkts       orig_ip_bytes   resp_pkts       resp_ip_bytes   tunnel_parents
```

(a) Connection Log Fields

```
1467309884.215619       CGeXpm1lArSbsBUPzc      136.159.49.122  58537   199.30.181.42   443     tcp     ssl     10.463505       322     4188    SF
ShADadFf 8      654     10      4596    (empty)
1467309884.216853       CFsoYf1pPcpcG0XNqd      136.159.49.122  58541   199.30.181.42   443     tcp     ssl     10.462587       322     4188    SF
ShADadFf 8      654     10      4596    (empty)
1467309884.215869       Cpy9wQ1dvhSYRQz1v6      136.159.49.122  58538   199.30.181.42   443     tcp     ssl     10.463590       322     4188    SF
ShADadFf 8      654     10      4596    (empty)
1467309884.216121       CE3poH2kViRdQTawmf      136.159.49.122  58539   199.30.181.42   443     tcp     ssl     10.463852       322     4188    SF
ShADadFf 8      654     10      4596    (empty)
1467309884.217105       COqdxW1l8DWDJjVGW5      136.159.49.122  58542   199.30.181.42   443     tcp     ssl     10.463491       322     4188    SF
ShADadFf 8      654     10      4596    (empty)
1467309727.676596       Cgoe2o2X65PqZxFAm4      136.159.16.7    57246   199.30.181.42   443     tcp     ssl     167.988306      7184    46270   SF
ShADdaFf 64     10520   90      50958   (empty)
1467309783.548660       CfcS7g26pyB49IRaDj      136.159.160.71  63881   199.30.181.42   443     tcp     ssl     112.251813      167141  537918  SF
ShADadfF 600    191153  535     559326  (empty)
1467309505.317761       C1s9Dz6RNYCT8WJ74       136.159.49.116  62446   199.30.181.42   443     tcp     -       95.522035       13155   42157   OTH
DadA    43      15379   60      45277   (empty)
1467309600.926910       CeEEqX2vIabzgQCQpa      136.159.49.122  64160   199.30.181.42   443     tcp     ssl     0.175203        1320    2214    S1
ShADad  13      4154    12      2858    (empty)
1467309600.585295       CbMX453pnUsSwEvdF9      136.159.49.122  64158   199.30.181.42   443     tcp     ssl     0.617881        3562    6287    S1
ShADda  17      4446    19      7283    (empty)
```

(b) Connection Log Format

Figure 3.3: Connection Log Example

For measuring all the D2L traffic for this thesis, we used several tools as discussed further in the next sections.

## 3.2 Passive Measurements

During passive network measurements, data is gathered by passively listening to network traffic. Wireshark is a network traffic sniffing tool, which was formerly known as Ethereal [83]. It captures packets in real time, and helps visualize it in a human-readable format. This measurement tool is non-intrusive. It merely observes and records traffic information, and does not generate any traffic of its own.

With various useful features like filters, color-coding, and packet details, we can dig deeper into D2L session details. There are various network interfaces available, such as wired Ethernet and the wireless interface shown in Figure 3.4a. As soon as we click on the relevant interface name, our session is passively monitored, and we see that the packets appear in timestamp order, i.e., each packet received and sent from our system gets captured.

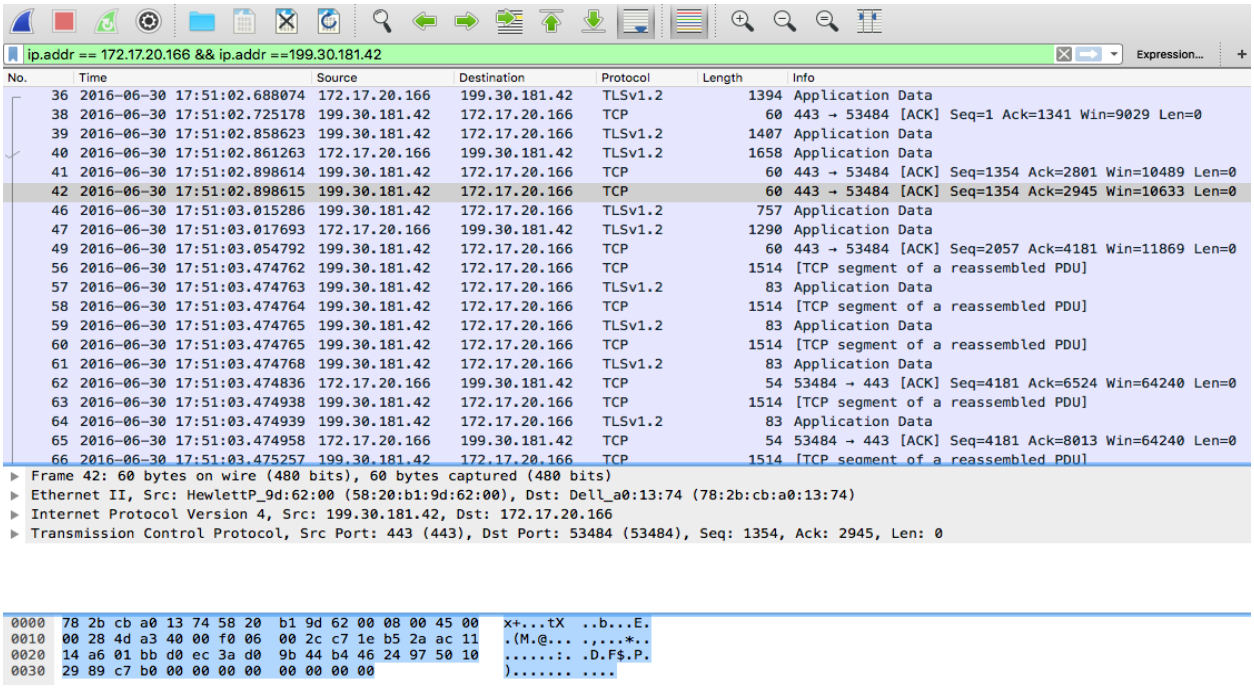
Figure 3.4b shows how these packets are represented in Wireshark. The color coding shows highlighted packets in blue, green, and black. These colors help identify the types of traffic measured. Table 3.1 illustrates the type of traffic associated with a few of these colors. These colors are generated by default, and can be customized easily.

Table 3.1: Color-codes identifying types of traffic captured

| Color-Code | Type of Traffic |
|---|---|
| Dark Blue | Domain Name Server (DNS) |
| Light Blue | User Datagram Protocol (UDP) |
| Green | Transmission Control Protocol (TCP) |
| Black | TCP packets with problems (out-of-order)) |

We can also inspect specific packets, by using filters to narrow down the traffic. This feature is useful, and can be performed by typing the filter name into the filter box at the top of the window and clicking Apply (we can also press Enter). For example, if we type `ip.addr == 172.17.20.166 && ip.addr == 199.30.181.42` as shown in Figure 3.4b, then the packets displayed are only the traffic sent and received between these two IPs, which helps us perform our session-related analysis in Chapter 4. The IP 172.17.20.166 is an on-campus client PC, and 199.30.181.42 is the D2L server IP. We could also type in `dns` as a filter to monitor only DNS packets. There are various other filters available, which can be found in the Wireshark manual.


(a) Available Interfaces




(b) Wireshark packet capture

Figure 3.4: Wireshark User Interface

The filters can also be customized based on a user's requirements. Wireshark provides auto-complete features, which completes the filters automatically and reduces the necessity of memorizing all the filters available. We can also follow TCP streams, perform bandwidth analysis, check round trip times and network throughput, draw TCP sequence number plots, and monitor client-server conversations. We can also select individual packets and see packet-level details like frame number, bytes transferred, source and destination IPs, IP version, source and destination port numbers, sequence numbers, and ACK numbers. Thus Wireshark is an extremely powerful network monitoring tool and was very useful for our analysis.

The main issue with passive measurements using Wireshark is the volume of data collected. During a two-minute session when a 177 KB file is downloaded, the total packet capture `PCAP` file size is about 1 MB. If there are multiple flows in the network, then the amount of data captured is quite large.

Since the number of packets captured can become overwhelming, Bro [14] was used to make our session analysis more insightful. It is particularly well-suited for research purposes, and enables a high-level study at the application layer. Bro was originally developed by a core team of scientists and software developers at the International Computer Science Institute (ICSI) in Berkeley, California, and the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, Illinois.

After generating our `PCAP` files through Wireshark, we uploaded them into Bro. This was another way to gather useful information like the flow of traffic for on-campus and off-campus users, the D2L IP address, session structures, DNS servers involved, parallel connections, persistent connections, establishment and termination of connections, protocols used, logging information (SSL, HTTP, connection), and bytes transferred for HTTP connections. This tool was extremely useful since it was efficient and matched with events in our monitor logs according to the timestamps. Specifically, the `PCAP` files generated by Wireshark contain entries that appear in three different logs of our monitor: connection, HTTP, and SSL.

Bro is a network intrusion detection system that passively monitors a network link [47]. We use Bro to obtain connection summaries of session traffic experiments and not to detect intrusive traffic in this thesis. Bro creates multiple logs [14]:

1. `capture_loss`: Shows lost packet details.
2. `conn`: Manages the tracking or logging of connection summary regarding TCP, UDP, and ICMP traffic.
3. `dns`: Tracks and logs DNS queries made and the corresponding responses.
4. `files`: Focuses on tracking FTP commands (file transfers if any) along with their metadata.
5. `http`: Logs HTTP events based on request/response pairs.
6. `known_hosts`: Shows every identifiable host IP address.
7. `known_services`: Displays the port numbers, identifies the protocols, and Internet services involved.
8. `software`: Provides server names and user agent information.
9. `ssl`: Shows HTTPS details and similar fields as the SSL logs in the monitor logs.
10. `stats`: Provides simple summary statistics on large streams of data.

### 3.2.1 Other Analysis Tools for Passive Measurements

A few other Web analytics tools, like BuiltWith® Technology Profiler, TCPIPUTILS.com, and Robtex were used as well. BuiltWith® provides Web server information, the Web page document standard, page structure information, the JavaScript libraries used, and the name of the Content Delivery Network (CDN) used. This was one of the ways we figured out the name of the Web server used by D2L. TCPIPUTILS.com was one of the ways by which we figured out the D2L

IP, geolocated it, and determined DNS server information, ASN number, and the IP-range /subnet including CIDR information. Robtex is a tool used to perform reverse DNS for resolving an IP address into its corresponding domain name. Robtex was available as one of the open source tools in the SLAC (Stanford Linear Accelerator Center) network monitoring tools Web site, which lists various open-source active/passive network measurement tools. HTTP Trace was also used to see the request / response headers, the exact elements requested, and the HTTP method names. This tool is offered as a Google Chrome browser extension under the developer tools category.

## 3.3 Active Measurements

Unlike passive measurements, active measurements generate probe packets that are transmitted over the network. These can be used to measure the time taken by a packet to reach a target destination in a network, the capacity available for a network path, or even the response time taken by an application. This category of measurement generates additional traffic in the network. That is why we have performed active measurements carefully, with basic active measurement tools like ping and traceroute, which have minimal impact on the normal flow of campus traffic.

### 3.3.1 Ping

A message can be sent to simply check if the destination IP address actually exists. Ping is a common technique for sending such a probe. It is a basic Internet program through which a user can verify the existence of a particular IP address, if requests can be sent to it, and if the host server is actually operating. It essentially works by sending out an Internet Control Message Protocol (ICMP) Echo Request to the relevant interface (in this case the D2L domain name), and waits for a reply. Figure 3.5 shows the network latency, in milliseconds, for D2L using this method.

```
PING d2l.ucalgary.ca (199.30.181.42): 56 data bytes
64 bytes from 199.30.181.42: icmp_seq=0 ttl=244 time=45.320 ms
64 bytes from 199.30.181.42: icmp_seq=1 ttl=244 time=45.231 ms
64 bytes from 199.30.181.42: icmp_seq=2 ttl=244 time=45.236 ms
64 bytes from 199.30.181.42: icmp_seq=3 ttl=244 time=45.878 ms
64 bytes from 199.30.181.42: icmp_seq=4 ttl=244 time=213.127 ms
64 bytes from 199.30.181.42: icmp_seq=5 ttl=244 time=172.027 ms
64 bytes from 199.30.181.42: icmp_seq=6 ttl=244 time=146.423 ms
64 bytes from 199.30.181.42: icmp_seq=7 ttl=244 time=44.473 ms
64 bytes from 199.30.181.42: icmp_seq=8 ttl=244 time=45.453 ms
64 bytes from 199.30.181.42: icmp_seq=9 ttl=244 time=45.967 ms
64 bytes from 199.30.181.42: icmp_seq=10 ttl=244 time=45.310 ms
64 bytes from 199.30.181.42: icmp_seq=11 ttl=244 time=45.580 ms
64 bytes from 199.30.181.42: icmp_seq=12 ttl=244 time=45.494 ms
64 bytes from 199.30.181.42: icmp_seq=13 ttl=244 time=45.949 ms
```

Figure 3.5: Ping Request to D2L

### 3.3.2 Traceroute

Traceroute is a utility that reports the route (i.e., the specific routers at every hop) between a client computer and a specified destination computer. In each line of the traceroute, the amount of time each hop took is calculated and displayed.

Figure 3.6 shows an example of the traceroute to D2L from an off-campus computer. When we perform a traceroute on the D2L IP from an off-campus machine, the first line represents the off-campus router and the next few lines represent our Internet Service Provider (ISP). Each subsequent line represents a router that is farther away geographically, which helps us determine the location of every router that the traffic traverses before reaching D2L.

```
traceroute to d2l.ucalgary.ca (199.30.181.42), 64 hops max, 52 byte packets
 1  192.168.1.254 (192.168.1.254)  6.405 ms  3.301 ms  3.373 ms
 2  10.139.234.1 (10.139.234.1)  12.802 ms  15.079 ms  12.121 ms
 3  154.11.10.156 (154.11.10.156)  4.552 ms  4.651 ms  3.963 ms
 4  154.11.2.178 (154.11.2.178)  4.162 ms  5.212 ms  4.758 ms
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  207.35.3.210 (207.35.3.210)  45.075 ms  1311.356 ms  45.318 ms
11  * * *
12  ucalgary.desire2learn.com (199.30.181.42)  129.549 ms  172.147 ms  44.706 ms
```

Figure 3.6: Traceroute to D2L from an off-campus computer

As soon as we enter the traceroute command, the software initiates the process of sending a packet, using ICMP (similar to ping). Inside the packet is a special time limit value called the 'time to live' (TTL). It enables traceroute to determine each hop with the help of a Time Exceeded

Message for each TTL value. It increases the time limit value, and retransmits the packet so that it can incrementally reach the next router in its path towards the destination, which in turn returns another Time Exceeded message. This process continues until the final destination router is reached. Traceroute understands when a packet has reached its destination by using a port number that is outside the normal range. Finally, a Port Unreachable message is returned, which enables traceroute to calculate the time for the final hop. As the traceroute progresses, the records are displayed hop by hop. In addition, each hop is measured three times. The asterisk (*) seen in Figure 3.6 indicates a hop that did not respond. The '*' in some columns also signifies that no response was received, and could indicate a packet loss. The format of each line is as follows:

```
Hop Number Domain Name [IP Address] RTT1 RTT2 RTT3
```

When a packet is passed between two routers, it is referred to as a hop. Figure 3.6 shows that it takes 12 hops to reach `ucalgary.desire2learn.com` (the domain name assigned to U of C's D2L server) from our location. RTT1, RTT2, and RTT3 are the round-trip times that it takes for a single packet to get to a hop and back to the client machine (in milliseconds). This is also referred to as latency, and similar numbers can be seen while using ping in Figure 3.5. Traceroute actually sends three packets to each hop and displays each of the times, so that we can estimate the variability of the latency. The domain name, when available, and the IP address often help us determine the location of a router. If this isn't available, only the IP address of the router is displayed, as in Figure 3.6.

A considerable number of traceroutes have been performed throughout this thesis, as discussed in Chapter 4.

## 3.4 Data Processing

Almost 50 GB of compressed log files are generated by Bro every day, including SSL logs, DNS logs, HTTP logs, TCP / UDP connection logs, SMTP logs, etc. The HTTP, SSL, and connection logs are separated into files in an hourly manner. For D2L, the traffic is transferred over

HTTPS, and the bytes transferred are seen in the connection logs. However, the type of files accessed and course names are a mystery, since it is encrypted.

We used bash, `awk`, and Python scripts to perform our analysis. For bash scripting, `grep` and `awk` were used. `awk` is a Linux software tool that conveniently extracts columns of relevant data from a file. Using `grep` in our bash scripts, we can extract rows of data whose fields match our interest. We used bash and `awk` scripts to extract columns from SSL, HTTP, and connection logs. These scripts are shown in Appendix A. Python, a user-friendly programming language, was used to perform logical and arithmetic operations, and some statistical analysis as well. We even used Python with our bash and `awk` scripts to manipulate strings and field values easily as per our requirements. A graph plotting module named 'Matplotlib' in Python was used for drawing our graphs. Several other free Python modules were used for generating useful results.

## 3.5 Ethical considerations

While performing active and passive measurements, a few ethical issues arose. During the packet capture process using Wireshark on the wireless interface, if the 'promiscuous mode' is enabled in our capture options, we could also monitor other traffic packets on the network. That is why we used the 'monitor mode', which captures packets only on a particular channel. Throughout our analysis, the promiscuous mode was turned off to focus on our specific sessions. Also, while performing analysis on some visible content available in our monitor logs, a few devices seemed to reoccur quite frequently, which indicates that a user might be issuing requests from an unencrypted channel. Our analysis is in no way concerned with figuring out the user behind that device, since we do not wish to compromise the privacy of any user throughout our analysis. This thesis is only concerned with aggregate traffic workload measurement for D2L, and to draw inferences based on our findings.

## 3.6 Summary

This chapter showed the methodology for use of the Endace DAG card, and the Bro logging system. Network traffic monitoring is important and helps us to troubleshoot and resolve issues detected with the help of certain tools. We provided a detailed introduction of such tools used throughout this thesis, both for active and passive measurements.

The following two chapters focus on D2L from two perspectives. First, Chapter 4 explores the details of how D2L works, using a microscopic view of client, server, network, and sessions. Later, Chapter 5 discusses general usage of D2L, from a macroscopic point of view. The Bro logging system has collected the data for our analysis throughout the two calender years, 2015 and 2016.

# Chapter 4

# A Microscopic Look at D2L

In this chapter, we take a detailed look at D2L operation at the University of Calgary. We present our analysis of client side, server side, and the network infrastructure in between. Section 4.1 starts with the user view of D2L. Section 4.2 presents the server side of D2L. Section 4.3 highlights the network infrastructure for D2L users. Section 4.4 shows the client side of D2L. Section 4.5 summarizes the entire chapter.

## 4.1 User View

D2L is a Web-based LMS. To access D2L services at the U of C, users first have to enter the D2L URL, which is `d2l.ucalgary.ca`, inside their Web browser (Chrome, Mozilla, IE, Safari, etc). As soon as this URL is entered, users are redirected to a Central Authentication Server (CAS) page at the U of C. After the correct login information is entered, CAS confirms a valid user, and the D2L home page comes up as shown in Figure 4.1.

### 4.1.1 Examples of Roles

I have used D2L as a student and as a Teaching Assistant (TA). All D2L users have roles such as student, TA, or instructor from which to select. Figure 4.1 shows the roles displayed inside D2L, which are the following:

1. `Instructor` shows a user's role as an instructor for a particular term, the courses being taught, and published content.

2. `Student` displays the enrolled courses for a student in a particular term, and grades achieved in each course.

3. `TA` shows the role of a teaching assistant, i.e., the courses and the enrolled students, and provides the option to upload grades with feedback as appropriate.

Instructors and TAs might have more complex roles, but at the time of this research our data set was not vast enough to explore the diversities of these user roles in more detail. Also our data set did not provide enough information to distinguish D2L usage between these categories. Even though our connection logs provide a `uid` field to identify individual connections, characterizing the usage of separate categories is still a challenge because NAT and DHCP complicate the traffic analysis. D2L uses HTTPS so along with encrypted content, the user details are also abstracted from us. However, we highlight the most common activities for each of these user categories.

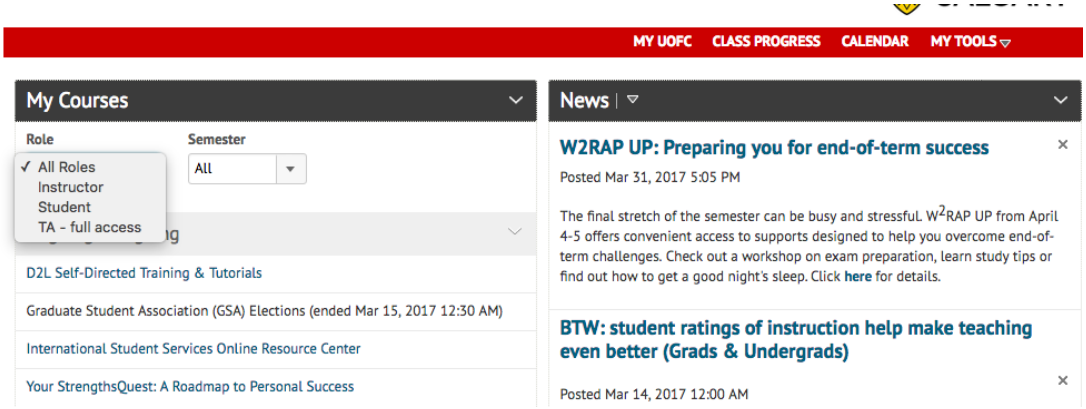



Figure 4.1: The available roles in D2L

### 4.1.2 Example of Student Role

When I use the student role, I can view all of my courses and the academic term for which I was enrolled in that course. Figure 4.2 is an example of the `CPSC 699` course, in which the grade is displayed on the same page where the assignment is submitted. This is done by the TA of that course using the feedback menu. A student could then check their respective feedback on each assignment. Typically students are involved in the following activities inside D2L:

- Selecting courses.
- Selecting a course module.

- Selecting a content item within a course.
- Downloading course content or assignment questions.
- Uploading assignments into the course's dropbox.
- Viewing grades or feedback.
- Checking course progress and comparing their standings with the rest of the class (lowest, average, and highest marks in the distribution).
- Check instructor / TA announcements, and course / assignment deadlines.

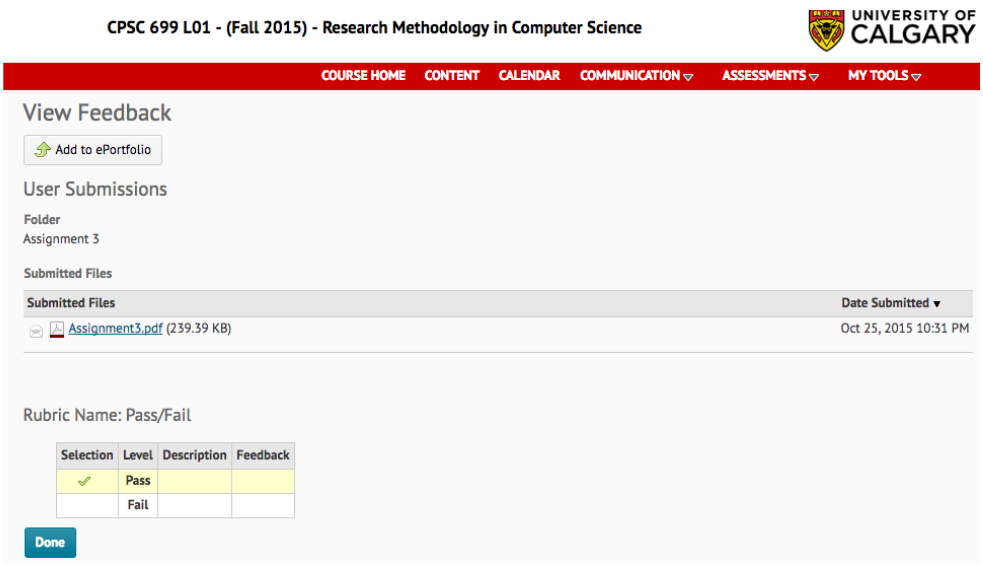



Figure 4.2: Student accessing assignment grade

Table 4.1 shows an example of my courses as a student during my period of study at the U of C. `CPSC` denotes the Computer Science department courses and `SENG` denotes the Software Engineering courses. Every department at the U of C has a predefined department ID. `CPSC 601` is a `Special Topics in CPSC` course, for which there are several categories of specializations. `SENG 607` is similar in this regard. In Table 4.1, only the specialization topic names have been

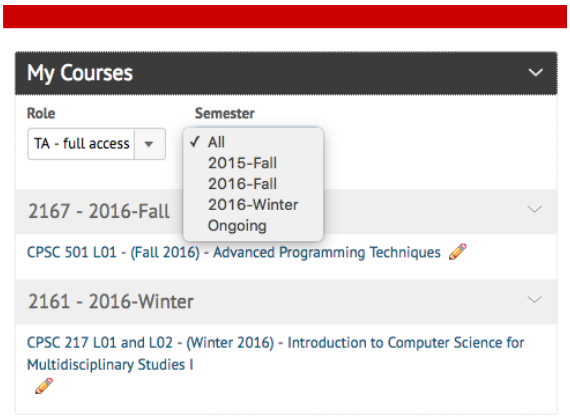



Figure 4.3: Academic session as a TA selection

shown. Some courses at the U of C are distributed over several terms. For example, `CPSC 699` appears as `CPSC 699A` and `CPSC 699B` for the Fall 2015 and Winter 2016 terms, respectively.

Table 4.1: My Courses at the U of C

| Course Code | Course Name | Course Term |
|---|---|---|
| CPSC 601 | System Modeling and Simulation | Fall 2015 |
| CPSC 626 | Network System Security | Fall 2015 |
| CPSC 699A | Research Methodology in CPSC | Fall 2015 |
| CPSC 601 | Network Coding | Winter 2016 |
| CPSC 625 | Principles of Computer Security | Winter 2016 |
| CPSC 699B | Research Methodology in CPSC | Winter 2016 |
| SENG 607 | Analytical Software Project Management | Fall 2016 |

### 4.1.3 Example of TA Role

As a TA, I have multiple options in D2L. First, all the courses for which I am a TA are shown. In addition, I could also select the specific semester in which I was a TA from the `Semester` drop down list. Figure 4.3 illustrates the semester options, which displays only the academic terms in which I was a TA.

Typical activities of a TA include:

- Editing course content.
- Adding customized sections.

- Downloading course content or assignment questions.
- Downloading the assignments submitted by students.
- Grading assignments.
- Importing grades as a CSV file.
- Editing grades.
- Viewing class performance statistics and posting notices.
- Checking instructor / TA announcements, and course / assignment deadlines.
- Posting supplementary course notes along with the instructor's notes to help students gain a better insight on the course subject.

Figure 4.4 and Figure 4.5 illustrate how grades are entered, and how my course content is published for CPSC 501. Other TAs can post their own content similarly.

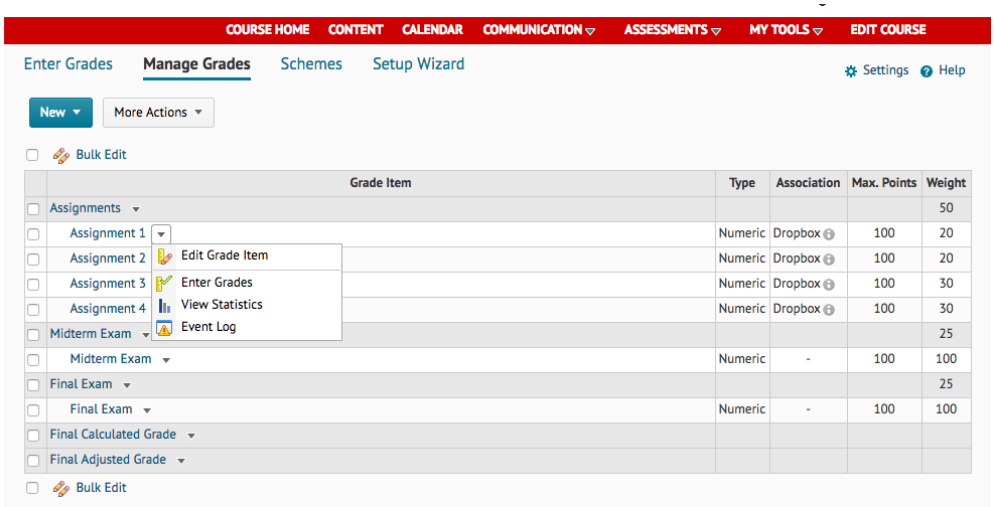



Figure 4.4: Entering grades

Table 4.2 shows an example of the courses for which I was a TA. `CPSC 217` was for first-year or second-year undergraduate students. `CPSC 501` was for final-year undergraduates.

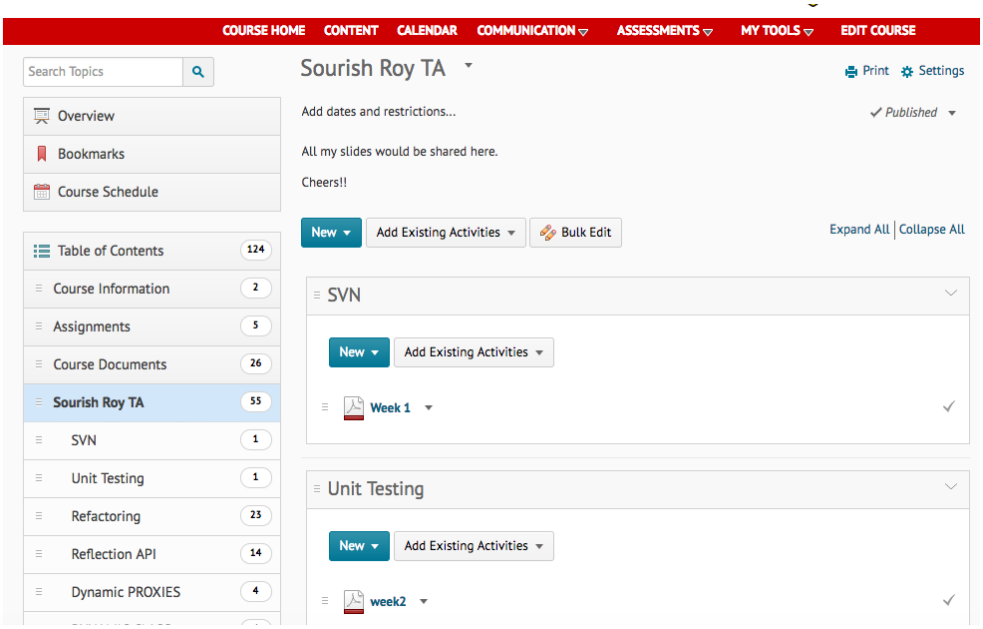



Figure 4.5: Uploaded Course content as a TA

Table 4.2: Courses assisted at the U of C as TA

| Course Code | Course Name | Course Term |
|---|---|---|
| CPSC 217 | Introduction to Computer Science for Multidisciplinary Studies I | Fall 2015 |
| CPSC 217 | Introduction to Computer Science for Multidisciplinary Studies I | Winter 2016 |
| CPSC 501 | Advanced Programming Techniques | Fall 2016 |

### 4.1.4 Example of an Instructor Role

Instructors and TAs have similar privileges. There are almost 40,000 active D2L users at the U of C. It is up to the instructors to activate and set up their respective D2L accounts. Initially an instructor sets up D2L with a single section for every course, and might also have multiple sections. The course grades are entered by instructors / TAs in various ways. It could be entered directly through D2L, through Excel spreadsheets, Google docs, or via a D2L service named Live Grades.

## 4.2 Server View

### 4.2.1 Server Infrastructure and Software

D2L uses Microsoft's Internet Information Server (IIS) version 7.5 as its Web server. We used the `curl -I` command on a Linux command line to determine this information. See Figure 4.6 for the curl output. The `-I` is a curl option, which prints only the server's HTTP response headers. The server name is confirmed by several other resources with tools like BuiltWith®, HTTP Trace, and even by the Taylor Institute IT staff.

```
Sourishs-MacBook-Air:~ roysourish$ curl -I https://d2l.ucalgary.ca/
HTTP/1.1 302 Found
Cache-Control: private
Content-Length: 127
Content-Type: text/html; charset=utf-8
Location: /d2l/login
Server: Microsoft-IIS/7.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
X-XSS-Protection: 0
Date: Mon, 19 Jun 2017 09:51:33 GMT
```

Figure 4.6: D2L Server Response Headers

D2L uses a Virtual Private Network (VPN) cloud whose primary data center is located in Toronto. D2L is extremely reliable, since even during the U of C's internal network outages, D2L services were showing 100% uptime with no security issues whatsoever. The course Web pages are actually stored in the D2L server, which is located in Toronto and hosted by the Bell Canada Data Center. D2L Corporation is presently working on migrating their services to a cloud computing platform called Amazon Web Services (AWS) [22].

The Taylor Institute for Teaching and Learning (TITL) server is hosted by `elearn.ucalgary.ca` on an Apache Web Server. The Taylor Institute runs this server and the U of C's IT data center hosts it. The `ucalgary.blogs.ca` site runs on the same Virtual Machine (VM) with the same IP address, which is 136.159.208.23.

All student grades are exported to PeopleSoft, which is part of the U of C's Enterprise Resource Planning (ERP) system.

D2L assigns courses with integer values that are referred to as `org unit ID`. D2L's Integrated Learning Platform (ILP) uses a role-based permission system. It uses the Valence Learning Framework APIs, which are based on an ID/Key-based authentication system and provide REST-like services. This ID/Key-based authentication system binds every API in a format that ties the action attempted with an LMS user and a corresponding organizational unit [50].

The APIs assign URLs based on resources. D2L provides its users with the following resources, which are visible in most URLs accessed:

1. 'le' stands for learning environment. This is terminology used by BrightSpace, and contains all course-related data in this category. The general URL format is : `/d2l/api/le/(version)/(orgUnitId)/content/topics/(topicId)` or `/d2l/api/le/(version)/(orgUnitId)/content/modules/(moduleId)`.

2. 'lp' stands for learning platform. It consists of all the tools that D2L provides under the `Edit Course` option. The format of the URL is : `/d2l/api/lp/(version)/tools/org/(toolId)`. These URLs also help in retrieving institution level information for the tools. In the example URL `/d2l/lp/auth/login/login.d2l`, the 'lp' also represents widgets and other User Interface (UI) design components of a particular HTML 5 page.

3. 'lms' provides competency-based learning options with grades, chats, dropbox, and other options. Typical URLs under this resource category look like: `/yourLMS.ca/d2l/api/lms/dropbox/admin/folders_manage.d2l?ou=orgUnitId` for dropbox, and `/yourLMS.ca/d2l/api/lms/grades/admin/manage/gradeslist.d2l?ou=orgUnitId` for grades.

4. 'im' provides Instant Messaging services for sending and receiving text messages

to other D2L users or classmates. It also allows connections with Adobe Connect to provide audio/video conferencing services. The skeleton URL under this category might look like:
`/yourLMS.ca/d2l/api/im/onlinerooms/roomlist.d2l?ou=orgUnitId`. Third-party software integration into D2L is a challenge, but D2L itself provides robust and functional services.

The URLs are scoped by `orgUnitId` and in some cases `OrgID`, which denotes the course offering, and the distinct tools available for use within courses. The APIs have version numbers based on their releases or modifications. The OrgIDs are primary keys generated by D2L and used internally by the organization. If these courses were generated by U of C instructors, then they are mapped to PeopleSoft and these keys are unique for U of C. For instance, McGill University would have a different set of OrgIDs for their courses. These OrgIDs are assigned contiguously to courses, but not all of the courses are active, and some might be archived. The most important fact is that none of these keys are reused locally. The U of C's IT staff maintain a database where these numbers correspond to courses in the PeopleSoft system.

### 4.2.2 A Real D2L URL example

Each of the courses created for instructors follow a certain naming pattern: Semester / Coursecode / LectureNumber, for example, Winter2016 / Chem / L01 /. In the network monitor logs, a typical URL for D2L content looks like the following:
`/d2l/le/content/122357/viewContent/1880383/View`

In the network monitor logs, these courses appear as six digit numbers (the number of digits could differ in another institute). In our example, '122357' is the course OrgID. The OrgID is visible in the URL, as shown in our example, followed by an action, and finally a ModuleID / TopicID, and another action. In our network logs, these course IDs are often followed by a 7-digit number, such as '1880383' in our example, which are the module or topic IDs.

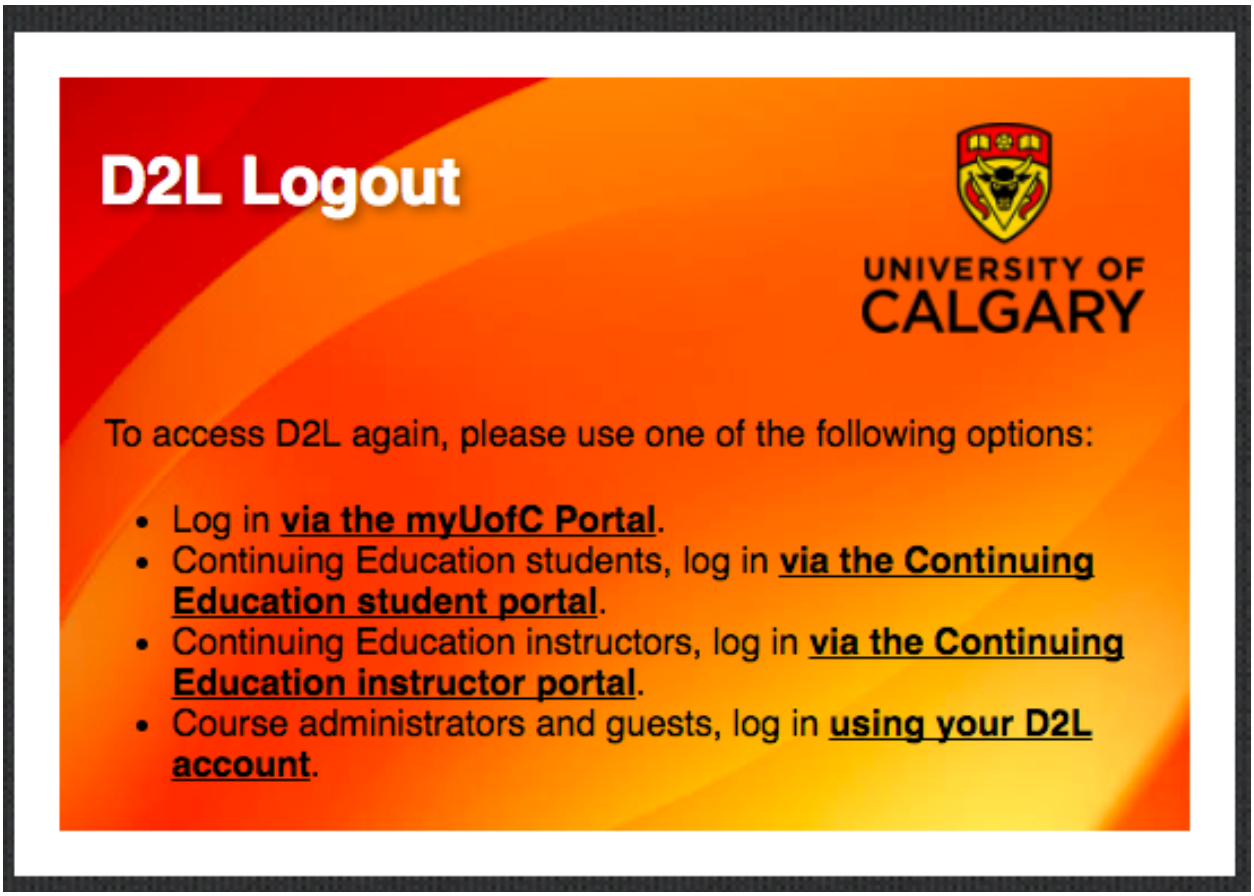



Figure 4.7: D2L or CAS Log-out Page

Within a D2L session, URLs are not visible in our logs, since D2L uses HTTPS. However, we know where they came from through the referer URLs in our network logs. Many people keep their CAS tabs open, and more than 90% of people log back into D2L through the CAS logout page, which also happens to be the D2L logout page. Figure 4.7 shows a D2L / CAS logout page. In addition, we have also noticed a lot of referrals from the Bing search engine, with people searching for 'D2L' or 'UCalgary'. After their D2L session, users tend to do other activities, in which case the referer is now the D2L page. They visit Web sites like Amazon, the New York Times, and other locations, which are visible in our network logs as the post-D2L referer URLs. Referer URL analysis is performed in Chapter 5.

### 4.2.3 Page Structure, Object Types and Page Load Times

We used the GTMetrix tool to perform this analysis. This tool was previously discussed in Chapter 3. The results show that the D2L page loading time for the first time is 7.8 seconds and for subsequent logins the average page load time is 1.2 seconds. The total page size is 2.52 Mega bytes (MB) on average. The course page of D2L uses Microsoft's ASP.net framework. The front end document standards include HTML5, JavaScript, and the X-XSS-Protection HTTP header to protect the Web site against some categories of Cross-Site Scripting (XSS) attacks. D2L uses UTF-8 as the encoding style for its Web pages. This analysis provides a better understanding of the general page layout and page load times.

### 4.2.4 D2L IP address space

The D2L IP address assigned to the University of Calgary is 199.30.181.42. Using a tool named TCPIPUTILS.com, which helps us geolocate an IP, we determined that the main D2L server is located in Kitchener, Ontario, Canada.

The IP address space for D2L includes 199.30.176.0/21 and 174.90.126.0/23. This IP address space is utilized by a variety of schools, colleges, universities, and other educational institutions all across Canada. A few well-known institutions on the 174.90.126.0/23 subnet are shown in Table 4.3. Also, other than the University of Calgary, there are several institutions that belong to the 199.30.176.0/21 subnet, as shown in Table 4.4. Outside Canada, there are several other universities on the D2L IP address space. An example of one such university in the USA is the University of Arizona, with an IP address of 199.30.177.155.

### 4.2.5 Cognate infrastructure

**Taylor Institute for Teaching and Learning(TITL) Web server**

The TITL Web server hosts `elearn.ucalgary.ca`. During our D2L session analysis, we observed another frequently occurring IP in our logs, 136.159.208.23. We performed a reverse DNS

Table 4.3: Example Universities on the 174.90.126.0/23 subnet

| University Name | IP |
|---|---|
| Lakeland College | 174.90.126.240 |
| Laurentian University | 174.90.126.160 |
| McMaster University | 174.90.126.152 |
| SAIT | 174.90.126.214 |
| University of Manitoba | 174.90.127.76 |
| University of Waterloo | 174.90.127.21 |
| Wilfrid Laurier University | 174.90.126.184 |

Table 4.4: Example Universities on the 199.30.176.0/21 subnet

| University Name | IP |
|---|---|
| British Columbia Institute of Tech | 199.30.177.52 |
| Calgary Catholic School District | 199.30.177.161 |
| Camosun College | 199.30.179.117 |
| Edmonton Catholic District School | 199.30.179.192 |
| McGill University | 199.30.180.35 |
| Queen's University | 199.30.179.126 |
| Ryerson University | 199.30.179.117 |
| University of Calgary | 199.30.181.42 |
| University of Guelph | 199.30.179.239 |
| University of Windsor | 199.30.179.176 |

lookup on this IP address and discovered that the host name was `learningtechnologies.ucalgary.ca`. The TITL Web server hosts multiple services all with the same IP (136.159.208.23). In addition, there are multiple domains hosted on subnet 136.159.208.0/24. The number of domains hosted by the IP 136.159.208.23 is 30. The TITL server and related services are all maintained by the U of C's IT department. According to a TITL staff member, the Web server access logs are archived in a PIWIK server. Notable domains hosted by 136.159.208.23 are `learningtechnologies.ucalgary.ca`, `wiki.ucalgary.ca`, `ucalgaryblogs.ca`, `openlogicproject.org`, and `ssharp.org`.

The `elearn.ucalgary.ca` Web site provides information regarding how to use teaching platforms, learning management systems, Web conferencing tools, and the student forums available at the University of Calgary. It also provides usage statistics for these tools, as shown in Table 4.5.

Table 4.5: Number of Instructors and Students on the elearn platforms

| Platform | Instructors | Students |
|---|---|---|
| D2L | 2,624 | 331,855 |
| Top Hat | 147 | 7,960 |
| Adobe Connect Meeting | 176 | 918 |
| ucalgaryblogs.ca | — | 151 |

Two more links available on the d2l.ucalgary.ca home page share the same IP address 136.159.37.76. These links are https://password.ucalgary.ca/ and https://acctman.ucalgary.ca/register. These sites typically help in resetting forgotten passwords or to register new users who are accessing D2L for the first time.

**Central Authentication Service (CAS)**

The University of Calgary is an Autonomous System (AS) within the hierarchical Internet network architecture. It uses the Border Gateway Protocol (BGP) to exchange routing information with other ASs. The public Autonomous System Number (ASN) is AS 33091.

In an AS like the U of C, there are multiple applications that need to be authenticated, and a central authentication server is useful in such a scenario.

The CAS URL is https://cas.ucalgary.ca, with IP address 136.159.96.51. This IP address is one of the most frequently observed IPs in our network monitor logs. This is because every user who wishes to access D2L must perform a secure login. They are redirected to the CAS home page before they can enter their respective accounts inside d2l.ucalgary.ca. A login database is maintained on campus by the CAS server.

CAS is a third-party Web application used by other applications at the U of C to authenticate users. At the U of C, CAS uses a single sign-on feature to make it convenient for users to migrate from one authenticated application to another. By other applications, we mean the myUofC portal, Continuing Education portal, and the D2L login page. However, unlike most authentication services, CAS is not simply a password-validation software. D2L never gains access to our passwords, which are stored on the CAS server. When we have not yet established a session with D2L, our browser is redirected to CAS's login page at https://cas.ucalgary.ca/cas/login.

We are then authenticated by CAS. If an incorrect user-name/password combination is entered, we must remain at the CAS page shown in Figure 4.7, and are provided with options to choose password reset or contact UCIT for assistance. When a correct password is entered, our browser returns to the D2L application. It could return to any application service hosted by the university, but CAS knows that we wish to use D2L because when we were initially redirected to CAS, a service parameter was supplied, for example,

`https://cas.ucalgary.ca/cas/login?service=https%3a%2f%2fd2l.ucalgary.ca%2f d2l%2fcustom %2fcas&ca.ucalgary.authent.ucid=true`

CAS looks at the service portion of the URL and understands that the user intends to use D2L. The final connection between CAS and D2L occurs through a socket connection opened by D2L, where the latter retrieves information from the CAS server. This completes the authentication process.

CAS is not a session management service, since it does not provide any mechanism to keep track of users once the authentication is successful. Since CAS is not a single sign-off facility, any user that logs out would still have access to D2L, since D2L keeps a persistent session with the user and it times out only after a certain period of inactivity, thus logging out the user. For example, even after logging out from `https://cas.ucalgary.ca/cas/logout`, a user does not necessarily log out from D2L. The logout process is complete only if users end a session through their D2L account itself.

For D2L interactions, the TITL server by default stores user logout event logs and generates the Central Authentication Service (CAS) logout page. The CAS server uses Shibboleth [56], which connects U of C users to applications both within and between other third-party organizations. Shibboleth provides Single Sign-On capabilities. It allows D2L to make informed authorization decisions for users to access their secure online resources.

The CAS logout page is created by the U of C. After a certain period of inactivity, the CAS page times out and logs out a user automatically. The D2L entry and exit, however, are separate from CAS's entry and exit points. There are a total of three CAS servers at the U of C. D2L stores

cookies on the user's computer via the Web browser.

### Domain Name Servers

Conceptually, the Domain Name Service (DNS) is the Internet's own phone book, with a directory of domain names and corresponding IP addresses. The details of DNS are specified elsewhere [44]. When students or faculty access D2L, they could do so in several ways: searching for the Web site from their browsers; entering the URL directly (d2l.ucalgary.ca); or using the `myucalgary.ca` portal page to select the D2L option. These options work in pretty much the same way. The DNS converts the domain name `d2l.ucalgary.ca` into an IP address (199.30.181.42).

The Web server used by D2L has a static IP, which provides a consistent point of contact for University of Calgary's D2L.

There are four main authoritative Domain Name Servers (DNS) used by D2L, namely:

- au1pbindex001.au1.int.d2l
- au2pbindex001.au1.int.d2l
- ca1pbindex001.tor01.desire2learn.d2l
- ca3pbindex001.ca3.int.d2l

There is one non-authoritative DNS namely

- 42.181.30.199.in-addr.arpa

The name of this non-authoritative DNS server is `ucalgary.desire2learn.com.`

### Content Delivery Network and Caching

There is lots of browser-level caching for the D2L Web site. Caching is a way of locally storing reusable components to allow faster subsequent requests. The page load time of a particular D2L content page is significantly improved due to browser-level caching. Whenever a browser loads a

D2L Web page for the first time, it downloads information related to the Web page. Upon future visits to the Web page, the page loads more quickly since the browser has already downloaded some of the HTML, CSS, JavaScript elements, JSON, and images. These files are saved locally in our browsers. We have observed that this helps reduce the page load time when browsing multiple course pages of the D2L Web site, since it avoids downloading the same objects multiple times.

The browser even has the ability to store user name and passwords when logging in to the D2L system. However, a few browsers like Google Chrome, Firefox, and Safari ask their users if they wish to save their passwords.

Brightspace by D2L uses a content delivery network (CDN), which is known as the Brightspace cloud content delivery network. This Brightspace cloud CDN works like a traditional CDN by deploying a system of distributed servers in locations around the globe, and delivers Web content based on the geographical location of the user. This cloud CDN creates copies of the pages of the D2L Web site using a network of geographically dispersed servers, thereby caching the contents of the page and redirecting the request from D2L's originating server to a CDN server that is close to the user. This is how D2L accelerates its delivery of static content.

We have noticed that the average load time of the D2L home page when accessed for the first time is approximately 3.71 seconds. After the data is cached, the load time for every subsequent visit to the D2L site is reduced to less than 1 second.

**Cookies**

The D2L server stores a cookie in our computers every time we visit the Web site. When we load the D2L site, our Web browsers send the cookie back to the D2L server every time we visit it. The Web site receives notification of our previous activity in the site by our browsers. These cookies expire after their lifespan, defined by their creators, is over. Such cookies generally store information needed for short periods of time.

The D2L Web site uses these cookies to track information regarding which D2L page the user

visited, their preferences, the number of visits, when and which buttons and banners they clicked, and several other bits of information. With the help of these user-specific cookies, the Web site presents customized forms. A few of the Web sites that save cookies in our Google Chrome Web browser are shown in Appendix B.

## 4.3 Network

### 4.3.1 Typical Internet path for D2L users

We have performed experiments using active measurement tools like `traceroute` (OS X) and `tracert` (Windows) to understand how a client machine accesses D2L over the Internet.

1. Version 1: The Client PC is on campus and has a statically-assigned public IP

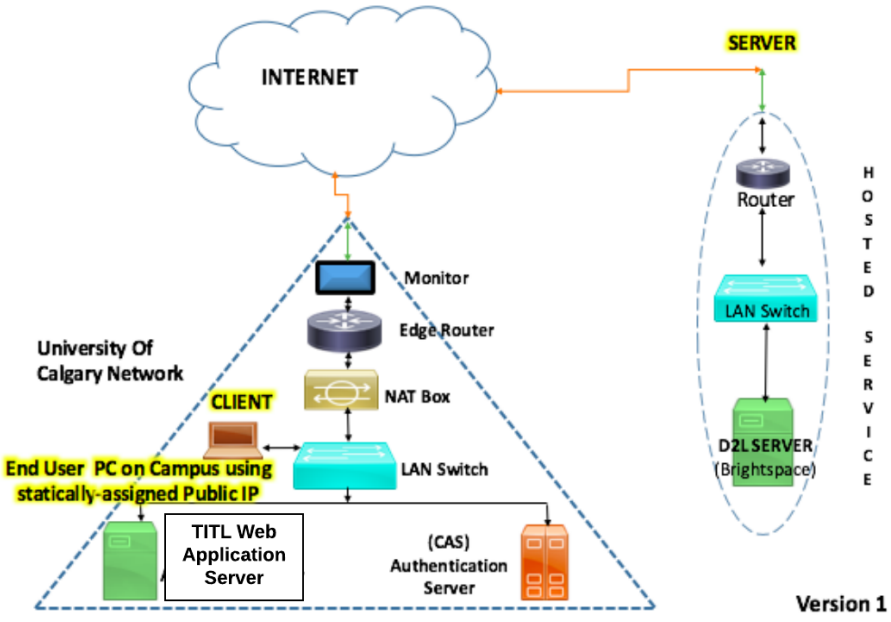



Figure 4.8: Network path to D2L for On-Campus Ethernet Users

The public IP of my office desktop machine is 136.159.16.20, and all other client machines on the same floor of the ICT building share the same public IP address. This indicates that a special type of NAT, namely Port Address Translation (PAT),

is occurring. In this scenario, initially a `tracert` was performed from a Windows desktop to the D2L IP. We used Wireshark to passively monitor our session, which helped us gain some valuable insights. The TITL server has an IP address of 136.159.208.23, and the CAS server has an IP address of 136.159.96.51. Both of these participate in establishing and terminating a session with D2L. The detailed session analysis is performed later in this chapter.

Figure 4.8 shows how the on-campus client PC accesses D2L. In Figure 4.8, the hosted service in Toronto is indicated by the oval, and the campus network in Calgary is enclosed by a triangle.

Initially, after visiting the D2L Web site, users have to login through CAS (Central Authentication Server). The Layer 2 switch redirects traffic towards the CAS server. The D2L user's login activities, such as timestamps, bytes transferred, source IP, destination IP, and referer URLs can be seen in the monitor logs. The traceroutes indicate that the hosted service is located in Toronto. Specifically, `desire2learn.ip4.torontointernetexchange.net` has a public IP 206.108.34.170, owned by the Toronto hosted service. Thus, before network traffic can reach `d2l.ucalgary.ca`, it has to go through Toronto.

In version 1, with PAT, many client PCs can access the Internet using the same public IP. The IP is the same, but the port numbers are different for the user machines. PAT translates internal IP addresses to a public Internet address.

2. Version 2: The wireless Client PC is on campus

   Figure 4.10 shows how a wireless device on campus accesses D2L.

   The client accesses D2L using WiFi wireless access points in this case. The traceroute in Figure 4.9 illustrates the flow of traffic to D2L. Also, from the traceroute in Figure 4.9, we can see the total number of hops required to reach D2L. Line 9 (`clgr2rtr2.canarie.ca`) and line 10 (`wnpg1rtr2.canarie.ca`) show the exis-

```
traceroute 199.30.181.42
traceroute to 199.30.181.42 (199.30.181.42), 64 hops max, 52 byte packets
 1  10.13.68.1 (10.13.68.1)  1.887 ms  1.738 ms  1.689 ms
 2  * * *
 3  10.16.122.1 (10.16.122.1)  2.522 ms  3.734 ms  1.975 ms
 4  10.16.122.4 (10.16.122.4)  1.908 ms  1.803 ms  1.806 ms
 5  10.16.121.1 (10.16.121.1)  2.363 ms  5.754 ms  2.520 ms
 6  10.16.242.4 (10.16.242.4)  2.577 ms  2.649 ms  3.675 ms
 7  h66-244-233-17.bigpipeinc.com (66.244.233.17)  4.435 ms  2.643 ms  3.313 ms
 8  h208-118-103-166.bigpipeinc.com (208.118.103.166)  3.345 ms  2.658 ms  4.036 ms
 9  clgr2rtr2.canarie.ca (199.212.24.66)  3.310 ms  4.527 ms  4.034 ms
10  wnpg1rtr2.canarie.ca (205.189.33.199)  38.686 ms  38.515 ms  38.584 ms
11  peer-as6509.br01.yyz1.tfbnw.net (103.4.99.3)  39.673 ms  38.460 ms  38.860 ms
12  desire2learn.ip4.torontointernetxchange.net (206.108.34.184)  39.758 ms  40.710 ms  41.422 ms
13  172.17.1.34 (172.17.1.34)  39.280  ms
    172.17.1.30 (172.17.1.30)  40.967 ms
    172.17.1.34 (172.17.1.34)  40.945 ms
14  ucalgary.desire2learn.com (199.30.181.42)  40.388 ms  40.820 ms  40.224 ms
```

Figure 4.9: Traceroute from a CPSC wireless PC to the D2L server IP

tence of the CANARIE network, which was originally an acronym for CAnadian Network for the Advancement of Research, Industry, and Education. It connects several provincial and territorial partner networks. This is a 100 gigabit per second digital research infrastructure utilized by Canada's research and education communities. Most Canadian universities utilize this service provided by the Government of Canada.

There is almost a 35 millisecond (ms) latency between Calgary (3.310 ms) and Winnipeg (38.686 ms), and another 2 ms latency to reach Toronto (40.388 ms). While logging in, the CAS server, which is located in our university, authenticates a client. The user and destination IPs are stored in our monitor logs, and the request

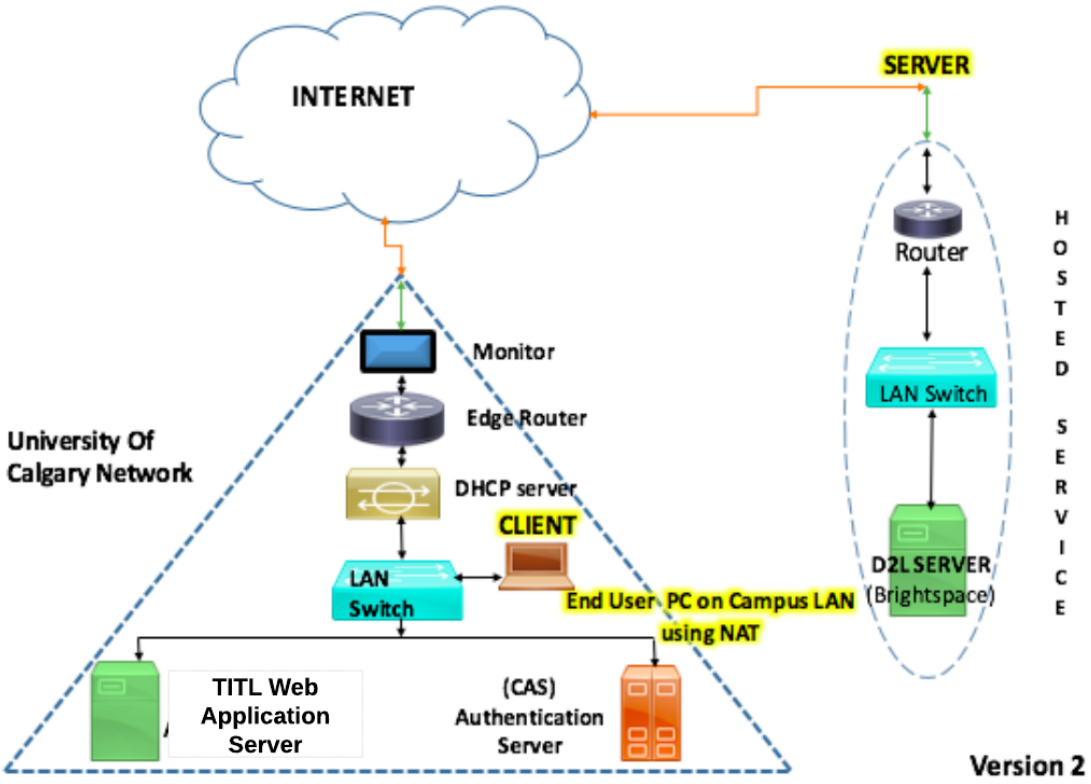



Figure 4.10: Network path to D2L for on-campus Wireless users

to reach the D2L server is again sent to the hosted service. So this traffic entering and returning is reflected in the monitor logs.

In this scenario, the Dynamic Host Configuration Protocol (DHCP) is used to allocate an IP from a pool of IPs. The IP addresses of the DHCP servers can be obtained using the `ipconfig/all` command via the command prompt in a Windows machine. Every user in this case has a different public IP address. These IP addresses are selected from the pool of available IPs managed by the DHCP server and assigned to the users.

3. Version 3: Off-campus client PC with a statically-assigned public IP address

   Figure 4.12 illustrates how off-campus users access D2L.

   In this case, we are using Shaw as the Internet Service Provider (ISP), and want to

know the Internet path for off campus users to access D2L. A similar `traceroute`

```
traceroute d2l.ucalgary.ca
traceroute to d2l.ucalgary.ca (199.30.181.42), 64 hops max, 52 byte packets
 1  192.168.1.1 (192.168.1.1)  7.172 ms  7.089 ms  10.466 ms
 2  174.0.224.1 (174.0.224.1)  19.851 ms  13.106 ms  19.403 ms
 3  198-48-144-1.cpe.pppoe.ca (198.48.144.1)  30.519 ms  21.105 ms  19.497 ms
 4  ge-1-0-0-0-agg01-van.teksavvy.com (76.10.191.57)  62.227 ms
    ge-1-0-7-0-agg01-van.teksavvy.com (76.10.191.109)  25.612 ms
    xe-1-3-1-806-agg01-van.teksavvy.com (76.10.191.141)  42.390 ms
 5  ae0-10-bdr01-van.teksavvy.com (76.10.191.1)  28.991 ms  29.763 ms  56.249 ms
 6  v704.core1.yvr1.he.net (66.160.158.249)  40.325 ms  29.817 ms  28.588 ms
 7  100ge10-2.core1.yyc1.he.net (184.105.64.114)  43.899 ms  72.650 ms  40.024 ms
 8  100ge10-2.core1.ywg1.he.net (184.105.222.97)  109.969 ms  76.444 ms  87.970 ms
 9  100ge6-2.core1.tor1.he.net (184.105.64.101)  82.642 ms  78.409 ms  141.555 ms
10  desire2learn.ip4.torontointernetxchange.net (206.108.34.184)  84.076 ms  71.872 ms  81.662 ms
11  * * *
12  ucalgary.desire2learn.com (199.30.181.42)  95.194 ms  72.797 ms  77.763 ms
```

Figure 4.11: Traceroute from an an off-campus PC to the D2L server IP

analysis has been done here (see Figure 4.11). The latency jump is 7.172 ms for the first hop compared to 1.887 ms for on-campus users. At the time of this research we observe that the latency is more than double for off-campus users compared to on-campus users. We also notice a Vancouver hop (YVR) in line 6 of our traceroute before the packets return to Calgary (YYC) in line 7, indicating that one of the ISP's main routers is in Vancouver. This particular hop is absent from our previous on-campus traceroutes. We also notice a significant 66 ms latency between line 7 and line 8 where it reaches Winnipeg (YWG), before the traffic finally reaches Toronto in line 9. We observe that the traceroute doesn't show any hop through the CANARIE network, since commercial Internet traffic is routed differently. In Figure 4.12, source NATing is taking place from my laptop to a public IP, thus allowing a private IP address to access a public network.

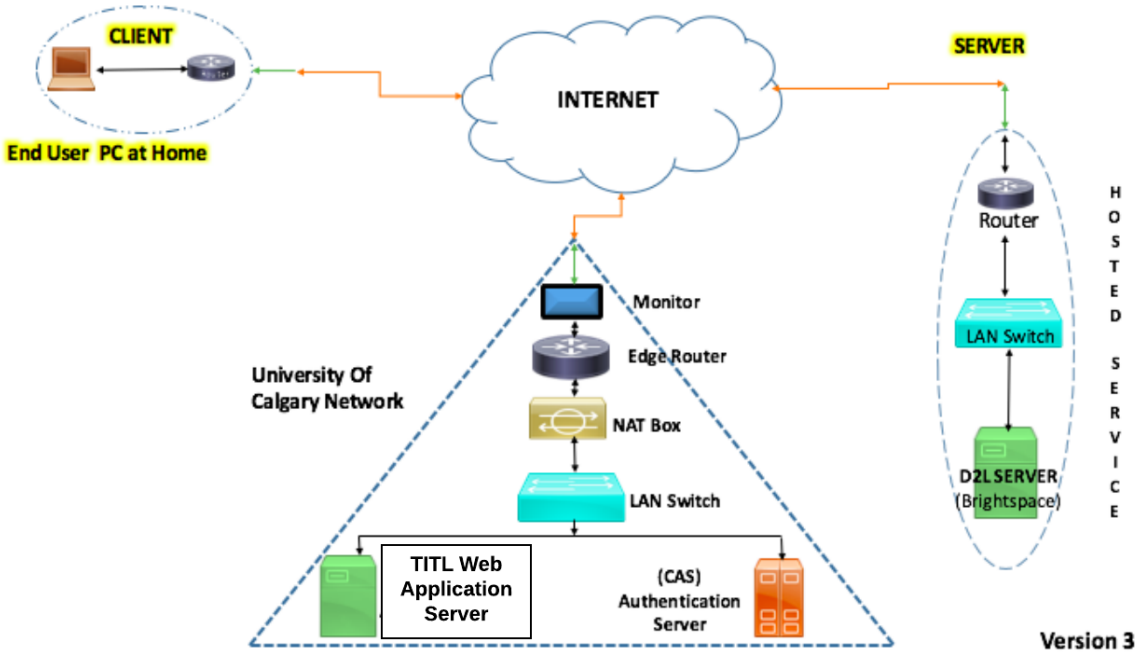



Figure 4.12: Network path to D2L for off campus users

### 4.3.2 Impacts of Network Address Translation (NAT)

We perform an analysis on NAT to check the entries in the monitor logs. The Computer Science department uses NAT for most of their computers. Essentially, these NAT devices allow a single device (NAT-enabled router or firewall) to act as an Internet gateway for the client (faculty, students, staff) machines, which are connected through a LAN.

The NAT device translates our private network IP addresses into a public IP address on the NAT-enabled gateway device. Thus, NAT hides the rest of our network from external users, allowing our whole network to appear as a single device to the Internet. This makes our network more secure, since all Internet communications are handled by the NAT device. This form of security is often called firewall security (when our NAT device is a firewall). These IP translations take place inside the firewall (NAT-enabled devices) through a NAT table that maps between private and public IPs. Additionally, NAT conserves the use of the limited IPv4 public addresses used by our university.

For a large campus like the University of Calgary, the private address ranges are 10.0.0.0/8,

172.16.0.0/12, and 192.168.0.0/16. The IP address for our Computer Science department's (CPSC) NAT is 136.159.16.20 and its name is `ms-studentUNIX-NAT0.cs.ucalgary.ca`. There are several types of NAT based on the type of users, departments, and campus buildings, as will be discussed in Chapter 5.

Figure 4.13 provides an overview of NAT in our campus, and how the Internet sees our network. It conserves the number of public addresses used within an organization, and provides stricter control of access to resources on both sides of the firewall.

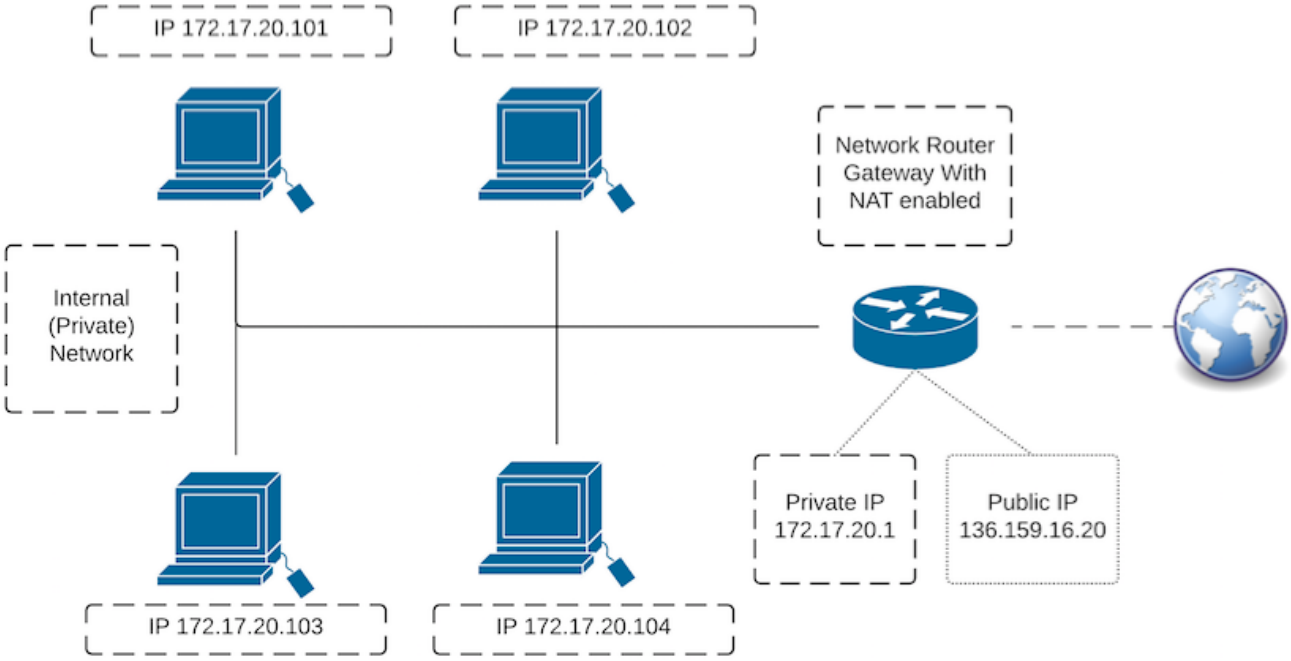



Figure 4.13: A private network at the University of Calgary using NAT to connect to the Internet

## 4.4 Client Side

### 4.4.1 D2L session structure and protocols used

A typical user needs to first login with their respective account login details (user name and password) to ensure a secure login. D2L also uses a single sign on (SSO), which authenticates an end user. As discussed earlier in the chapter, this is achieved via CAS (Central Authentication Server). The user can then access the modules within the D2L environment, which include the D2L home page, course homepage, course content, etc.

**Wireshark and Monitor Log traces**

To study the D2L session structure, we used our monitor logs and software-based measurement tools like Wireshark and Bro. Wireshark helped us in capturing packets by passively sniffing a traffic session in real time without generating any additional probe traffic. Bro helped us in analyzing these captured packets to understand the session structure. We used three different browsers for our experiments, namely Google Chrome, Mozilla Firefox, and Safari. The session structure results were very similar for all of the above mentioned browsers, and thus we present results for a Chrome browser only. These experiments were also performed in wireless and wired LAN environments, as well as different operating systems like OS X and Windows 8. The results are discussed in detail here.

In this section, we show a few real example user sessions in D2L. We have performed several Wireshark experiments, to monitor multiple D2L sessions. The trace shows the session structure, and how packets are sent and received by the client. We notice that a series of parallel and persistent connections are established in the process.

Persistent connections use a single TCP connection to transmit and receive one or more HTTP requests and responses. Persistent connections do not open a new connection every time a request is seen, thus reducing the total number of connections.

Initially, we anticipated that the parallel connections are dependent only on the type of browser

used, but with the help of the experiments shown below, it becomes clear that these connections are not only browser dependent, but are also dependent on user activity. The following experiments help us get a clear understanding of the session structure and the established connections.

Each of the following session diagrams shows the timestamp in millisecond resolution according to their order of appearance in our logs, Wireshark, and Bro. They show the respective port numbers and the types of connections seen, between the client, CAS, TITL, and the D2L server.

1. An on-campus file download session where the client PC is a desktop and has a wired Ethernet connection :

   Throughout our session experiments the majority of connections seen are TCP. Very few UDP connections are seen, and are not included in the session diagrams. The sessions comprise of parallel HTTPS connections until the very end, where a few persistent HTTP connections are seen, which shed light upon the D2L resources stored locally on-campus.

   During this two-minute experiment, performed on June 30, 2016, a session diagram of 34 seconds is shown, where a file of size 177 Kilobytes (KB) was downloaded from a D2L page. In Figure 4.14, first a user logs in, and two HTTPS connections are established with CAS from port 53498. Then six parallel HTTPS connections are established with D2L, indicating the file download process through secure connections. Another HTTPS connection with D2L (port 53489) is seen after termination of a connection with TITL. This indicates the beginning of the logout process. Next, we observe two pairs of persistent connections with D2L. These are GET requests for `https://d2l.ucalgary.ca`, which receive 302 status code responses. Following these persistent connections,are a pair of parallel HTTP connections with D2L. Three HTTP connections with TITL are seen at the end of this session, which indicates a proper log out process. Several FIN connections seen throughout the session indicate proper closing of connections.

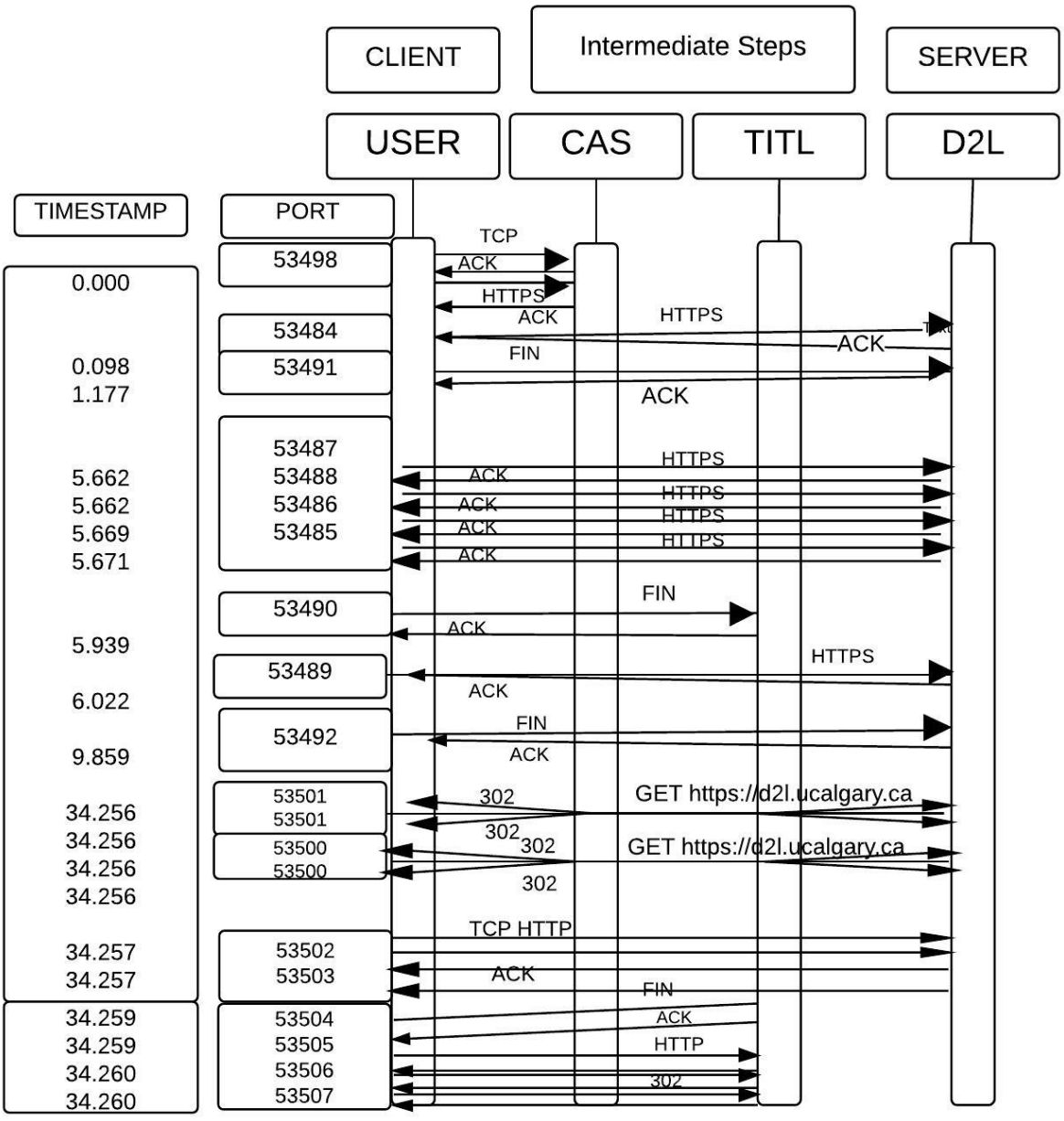



Figure 4.14: D2L session with wired Ethernet where a file is downloaded

We now relate this session to our monitor view, and show these connections in the connection, SSL, and HTTP logs. The connection log in Figure 4.15 shows all the port numbers involved throughout the session between 53484 to 53503, and even the type of TCP connection (SSL or HTTP) as the session progresses.

```
1 1467330654.859900      CKXf9o4AL6J22nI6z3     136.159.16.20   53484   199.30.181.42   443     tcp     ssl     155.704539      23133   97772
 RSTR    T       F       0       ShADadr 54      25307   144     103540  (empty)
1 1467330654.860155      Cbw9Fm4I99LAIg8qAh     136.159.16.20   53486   199.30.181.42   443     tcp     ssl     155.704494      9688    24191
 RSTR    T       F       0       ShADadr 23      10622   45      25999   (empty)
1 1467330654.860159      CpIXHcl7jTGg9ZZz1      136.159.16.20   53485   199.30.181.42   443     tcp     ssl     152.229457      11067   36192
 RSTR    T       F       0       ShADadr 30      12281   65      38800   (empty)
1 1467330654.860168      CPcbqU3rIeuWkE10el     136.159.16.20   53487   199.30.181.42   443     tcp     ssl     170.878203      18375   227497
RSTR    T       F       0       ShADadr 55      20589   229     236665  (empty)
1 1467330654.860185      CdsAqTGDbQ1ziruPa      136.159.16.20   53488   199.30.181.42   443     tcp     ssl     150.865304      14213   375312
RSTR    T       F       0       ShADadr 78      17347   335     388720  (empty)
1 1467330654.860400      CXDLXo30KpQoXnmvld     136.159.16.20   53489   199.30.181.42   443     tcp     ssl     150.846427      8633    24990
 RSTR    T       F       0       ShADadr 24      9607    45      26798   (empty)
1 1467330697.557215      CPgd5k19rfUQOP7ib9     136.159.16.20   53500   199.30.181.42   80      tcp     http    300.255577      1840    274SF
 T       F       0       ShADdaFf        13      2378    11      722     (empty)
1 1467330697.557434      Cf5cxa3gDFTjMHqvj5     136.159.16.20   53501   199.30.181.42   80      tcp     http    300.255412      1850    274SF
 T       F       0       ShADdaFf        13      2388    11      722     (empty)
1 1467330697.557690      CRXjUz42MLoZ3TpwBi     136.159.16.20   53502   199.30.181.42   80      tcp     -       20.042029       0       0  SF
 T       F       0       ShAFaf  4       172     3       128     (empty)
1 1467330697.557693      CnfsCzd9yUJHzk6w       136.159.16.20   53503   199.30.181.42   80      tcp     -       20.042336       0       0  SF
 T       F       0       ShAFaf  4       172     3       128     (empty)
```

Figure 4.15: A D2L session as reflected in the Connection Logs

The SSL logs in Figure 4.16 show the secure HTTP (HTTPS) connections with destination port 443, along with the encryption keys and message digests.

The HTTP log in Figure 4.17 shows the logout page, which indicates the end of a session. It displays the files stored locally on-campus, for example the logos, CSS files, and images in '.png' or '.jpg' format. It also shows the user agent details of the computer from which this session experiment was performed. It shows the exact browser version `Chrome/51.0.2704.103`, and the Windows desktop machine with `Windows NT 6.3; Win64; x64`, which was used during this session experiment.

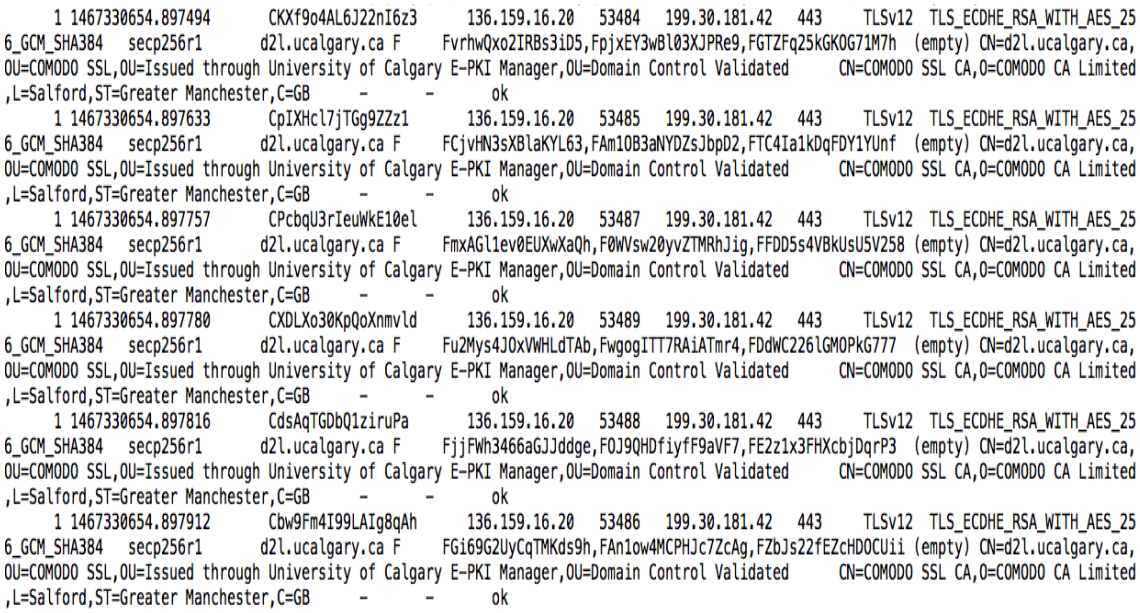



Figure 4.16: A D2L session as reflected in the SSL Logs

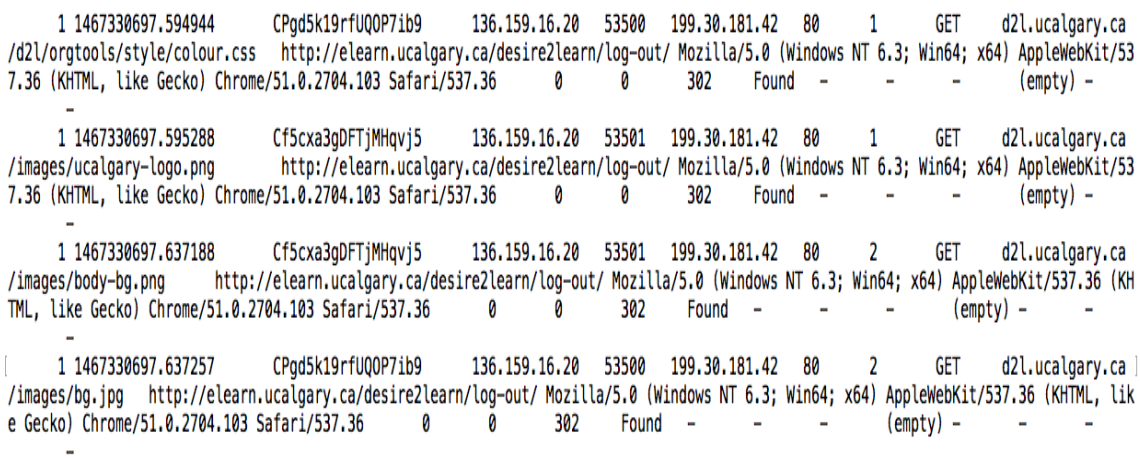



Figure 4.17: A D2L session as reflected in the HTTP Logs

The connection logs show how the SSL connections are established first with D2L, followed by the HTTP connections towards the end. The monitor logs, however, do not clearly shed light upon the intermediate servers involved. For the connections made with CAS or TITL server, the role of Wireshark becomes vital. For example, the initial connection made with the CAS server from port 53498, or the four connections seen towards the end of the session with the TITL Web server, are unclear in our monitor logs. The CAS IP 136.159.96.51 is the same for the logout page of

myUofC portal, continuing education portal, and other U of C applications, which makes it is difficult to understand if it is associated with a D2L session.

The session durations vary slightly, where the monitor log shows 42.698 seconds of session duration, but Bro shows a slightly shorter duration of 34.260. It should be noted that the experiment was performed for two minutes.

The SSL logs clearly relate to our session diagram, which shows exactly six HTTPS connections made with D2L. The HTTP logs show two pairs of GET requests to `https://d2lucalgary.ca` like the ones in our diagram from ports 53500 and 53501, which get redirected with 302 status codes to the D2L log-out page, `http://elearn.ucalgary.ca/desire2learn/log-out/`. This logout page URL contains `elearn.ucalgary.ca`, which is the URL of the TITL Web server, which further confirms that the logout process occurs locally inside the U of C campus. For the next few session diagrams, we show only the D2L session structure in various scenarios.

2. An on-campus D2L login/logout session where the Client PC is a desktop and has a wired Ethernet connection :

   In this session experiment (performed on April 18, 2017), shown in Figure 4.18, we can see that at first, there are six parallel connections with CAS as soon as we log in to D2L. This signifies the login process to D2L. The login / logout steps in this case were performed in six seconds. The port numbers are between 49485 and 49490. Similarly, six more parallel connections are again established, but this time, with D2L. After five seconds, three parallel connections are made with the TITL Web server. The port numbers in this case are 49549, 49550, and 49551, which are monotonically increasing. Finally, two pairs of persistent connections with the TITL Web server are seen at the very end of the session. Ports 49546 and 49547 create persistent connections where visible HTTP information is exchanged.

These are mainly the CAS/D2L logout page components and various images like the University of Calgary logo, and other JavaScript files, stored locally on campus in the TITL server and displayed on the logout Web page. The last few connections with the TITL Web server signifies the log-out process.

3. An on campus D2L login/logout session where the client PC is a wireless device :

   This analysis on April 19, 2017 comprises a similar login/logout session with D2L. Figure 4.19 illustrates this scenario. In contrast to our first session example, four parallel TCP connections are established and terminated with D2L first. The port numbers are between 59491 and 59488. This event is followed by six parallel connections with port numbers between 59493 and 59498 at the same timestamp, and is similar to what we witnessed during our previous analysis. After this, two parallel HTTP connections arise with the TITL Web server, and then two more HTTP/1.1 connections are seen. These receive responses with status codes of 301 (redirect) and 304 (not modified), respectively. Towards the end of the session, a pair of persistent HTTP/1.0 connections are created with D2L, which responds with a 302 (moved temporarily) status code.

4. An off-campus D2L login/logout session where the client PC is a wireless device :

   This experiment was performed on August 5, 2016. Figure 4.20 shows a D2L session when a user is off campus. This session experiment in Figure 4.20 is very similar to our previous analysis. Both sessions involve wireless devices and WiFi connections. The only difference is that towards the end of this session, when HTTP/1.1 connections are established with the TITL Web server, two replies are seen with status codes 301 (redirect) and 200 (OK), which suggests that the user's request was properly received, deciphered, and successfully accepted. Towards the end of the session, we see a pair of persistent HTTP/1.0 connections with D2L, and

a 302 reply. A series of HTTP connections with the TITL server are seen before the session terminates.

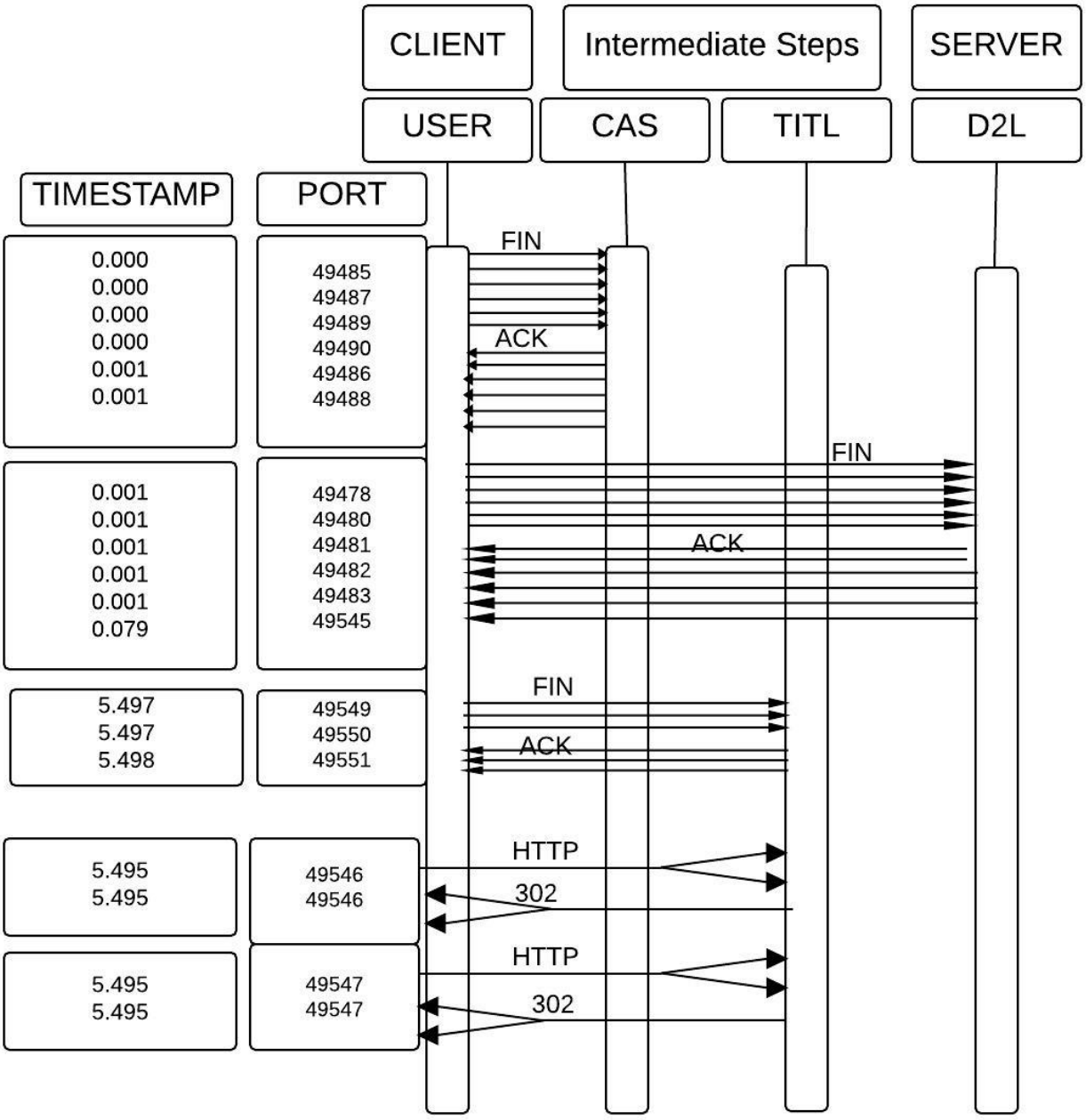



Figure 4.18: D2L login/logout session with an on-campus wired Ethernet device

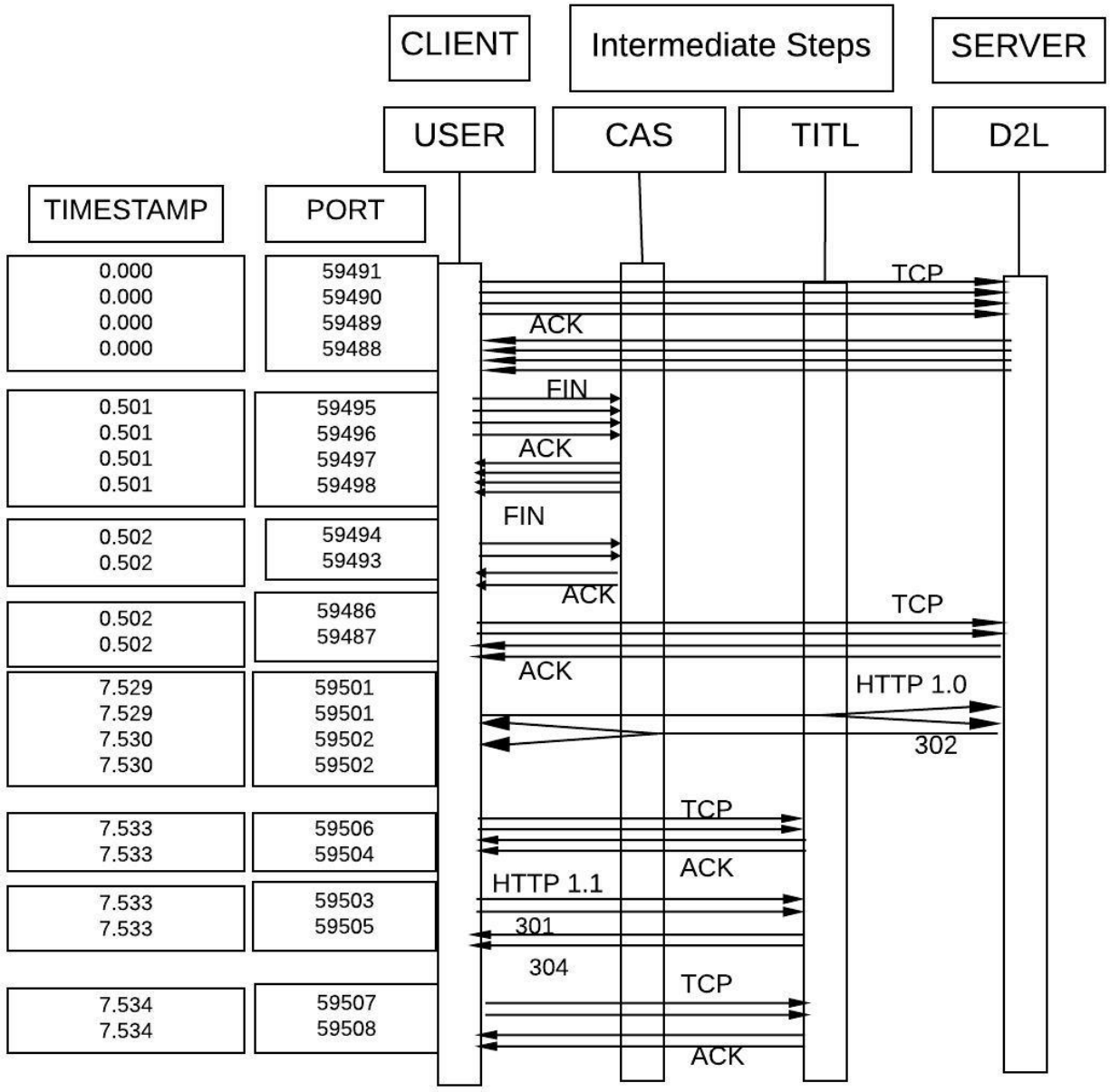



Figure 4.19: D2L login/logout session with an on-campus wireless device

5. An on-campus file upload session where the client PC is a desktop and has a wired Ethernet connection :

   This five-minute session experiment is a scenario where a file is uploaded into D2L. This experiment was performed on April 18, 2017, and the three-minute session is shown in Figure 4.21. The same file for the previous experiment with a size

of 177 KB was uploaded into D2L. In this case, five parallel connections were established with ports from 49873 to 49877. This was followed by five parallel connections with D2L. After two seconds, we see a final connection with CAS, which is followed by five more parallel connections with D2L between 11 and 13 seconds. A pair of HTTP/1.1 connections with the TITL Web server are observed towards the end of the session. Finally, the connections are terminated by three pairs of persistent connections from ports 49889, 49890, and 49891 at the same timestamp with D2L.

The parallel connections do not depend solely on the type of browser. These experiments were performed multiple times, during various times of the day and year, and with three distinct browsers: Google Chrome, Mozilla Firefox, and Safari. These experiments yielded the same results each time even though the maximum number of permissible HTTP/1.1 persistent connections per device vary from browser to browser with 4 for Safari and 6 for Firefox and Chrome [57]. The browsers enhance performance by sending out GET requests for inline images, and they do not wait for the initial GET requests to finish. These GETs are thus processed in parallel. The results discussed in this section are with a Google Chrome browser, and we observed that the type of user activity (browsing, file uploads, and downloads) also play a role in the total number of connections seen.

Also, we compared the Wireshark traces with our monitor logs. During this analysis, we realized that the session activities are first reflected in the connection logs, then in the SSL logs, and finally in the HTTP logs. The HTTP logs show us the persistent connection information, which happen to be the previously discussed inline images. The connection logs display the start and end timestamps of sessions. It starts showing trace information before the timestamps in the SSL logs and finishes just before the trace is transferred to the HTTP logs. Thus, the SSL logs have timestamps just after the session timestamps in the connection logs.

In this section we perform D2L session analysis in five scenarios. We notice that the under-

lying session structure is similar for the on-campus wired and wireless login/logout sessions in Figure 4.18 and Figure 4.19. Figure 4.20 is our final off-campus login/logout session where a large number of connections are seen with CAS, which suggest a proper authentication process for off-campus devices with the CAS server. These three are the standard ways for a user to begin and close a session with D2L. The main reason for us to discuss user sessions while downloading and uploading a file is to illustrate how the TCP parallel connections with the D2L server increase during these two scenarios compared to a login/logout session. This also highlights the idiosyncrasies involved in a D2L session. This increase in parallel connections with D2L is evident from Figure 4.14 and Figure 4.21. The session diagrams also show that the intermediate servers CAS and TITL play an important role during a D2L session. Also, the port numbers are mostly NAT generated and seem to be random. The reason for this is that the Bro logs are not actually in timestamp order. Specifically, the log entries get written when the connection completes i.e., they represent the closing of connections and not the starts. The starts could be arbitrarily far in the past.

## 4.5 Summary

In this chapter, we addressed our first research question regarding how D2L works. Specifically, we examined D2L from five major perspectives: user view, server view, network view, client view, and session characteristics. We discussed the server software and technologies involved, the DNS infrastructure, IP address space, Content Delivery Network, and caching. We geolocated the D2L IP, performed several traceroutes, and determined the role of NAT and DHCP. We discussed the typical Internet path for on-campus and off-campus users. Finally, we performed several experiments to understand the D2L session characteristics with the help of Wireshark, and how parallel and persistent connections reduce the total number of connections.

In the next chapter, we investigate the characteristics of D2L traffic during our observation period, using a macroscopic point of view.

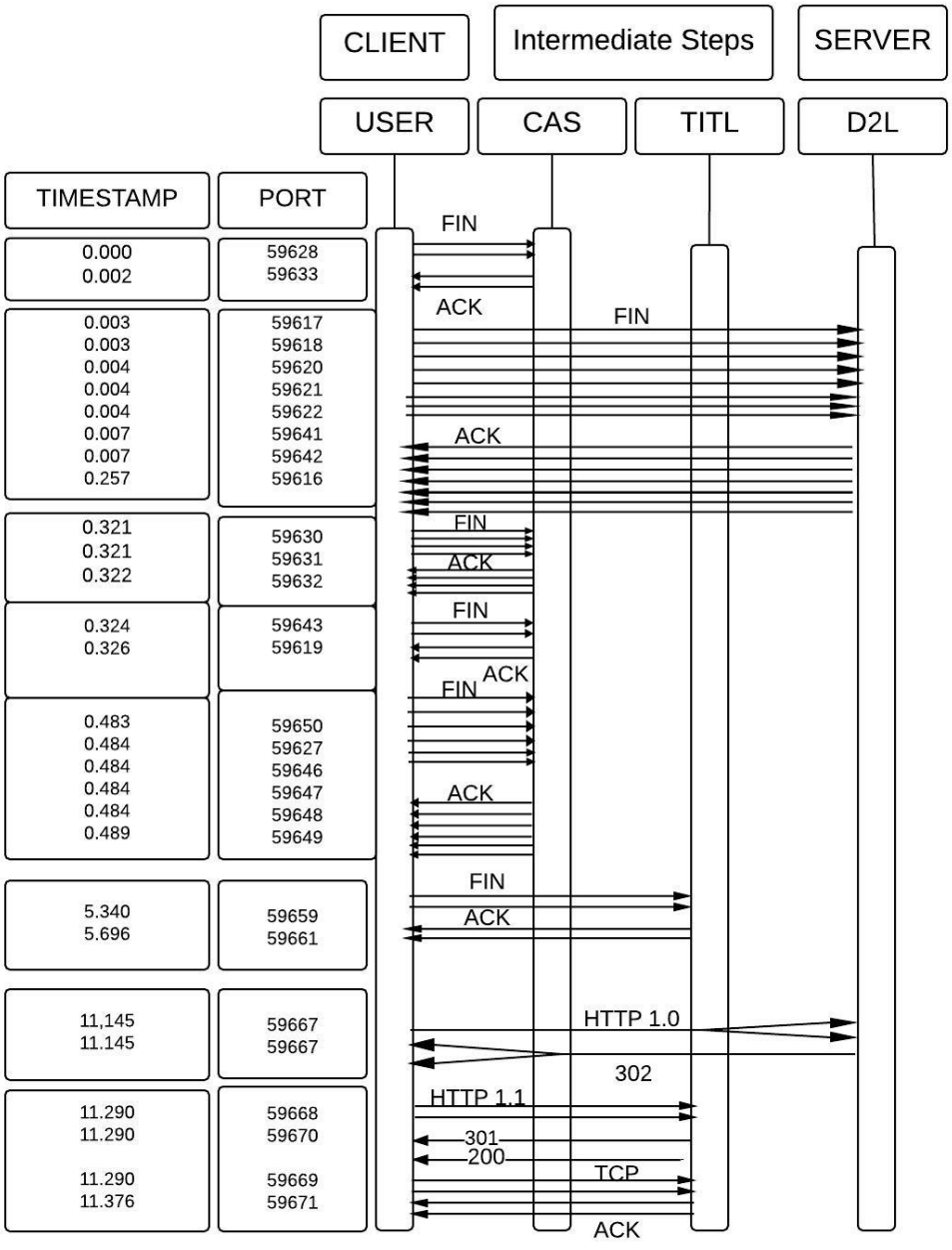



Figure 4.20: D2L login/logout session with an off campus wireless device

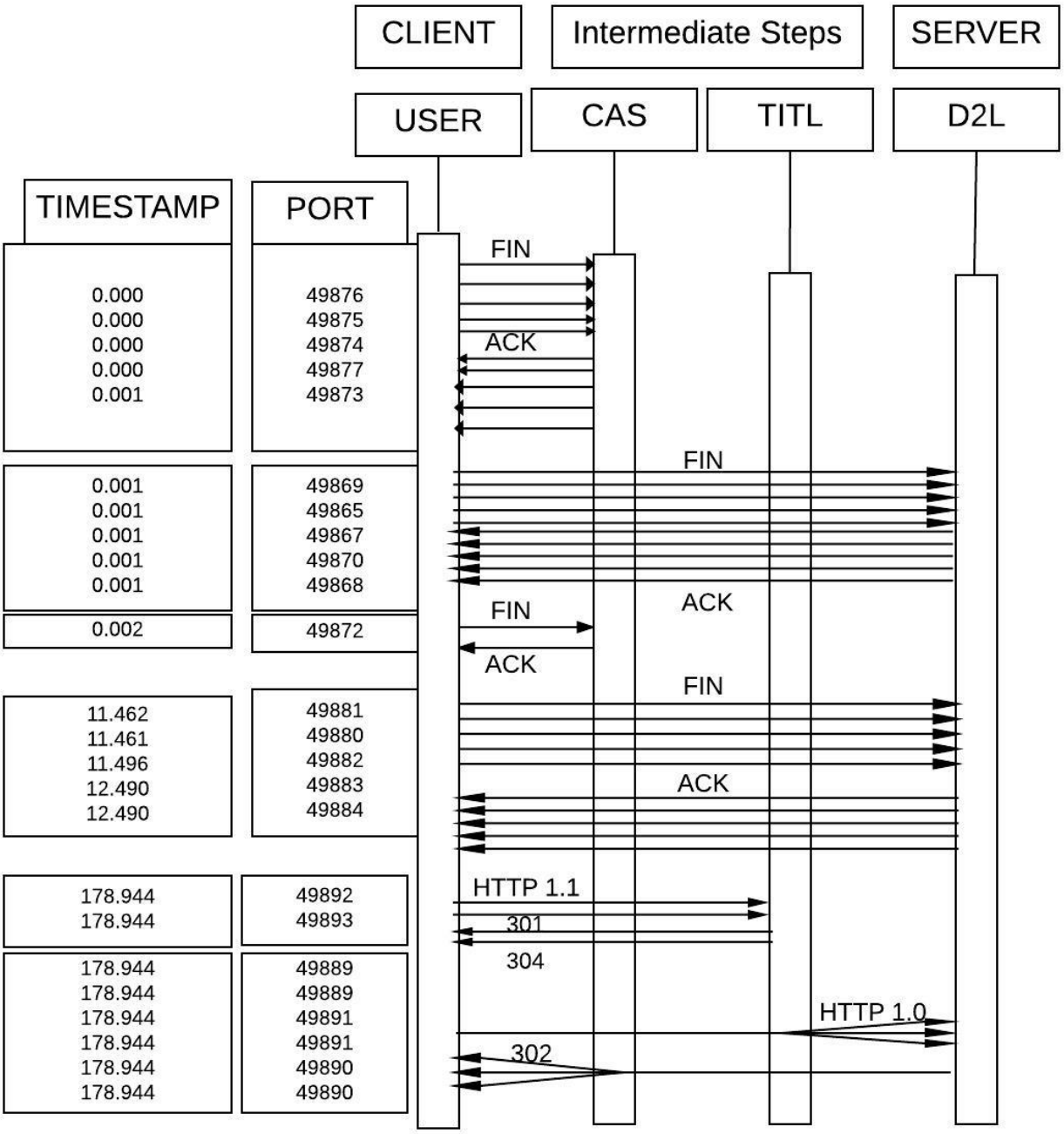



Figure 4.21: D2L session with wired Ethernet where a file is uploaded

# Chapter 5

# A Macroscopic Look at D2L

In this chapter, we perform an in-depth workload study of D2L traffic from a macroscopic perspective. Section 5.1 analyzes the HTTP and HTTPS traffic, including the total number of requests on a daily, weekly, and monthly basis, as well as the typical semester and yearly HTTPS patterns. Section 5.2 compares two full academic years (2015 and 2016) to understand the change in D2L usage over time. Section 5.3 presents an IP analysis. Section 5.4 illustrates the various user agents seen. Section 5.5 highlights session-related analysis. Section 5.6 shows the HTTP methods and response codes seen. Section 5.7 provides a D2L URL popularity profile based on the referer URLs in the monitor logs. Finally, Section 5.8 summarizes the chapter.

## 5.1 Overview of D2L Traffic Volume

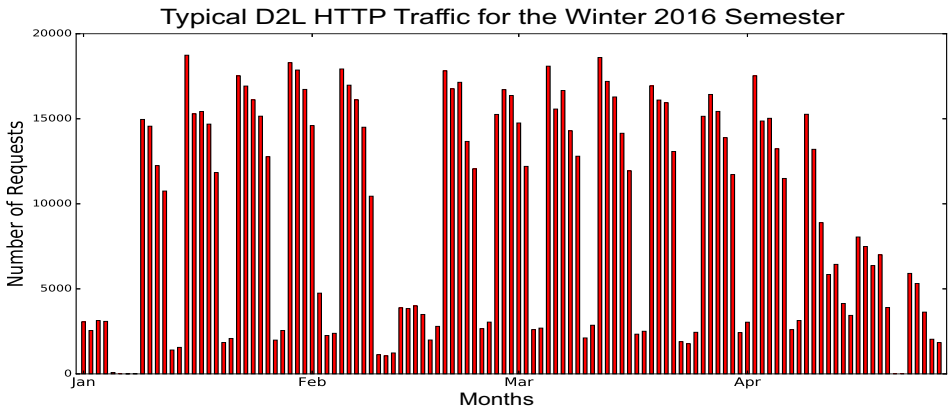
For studying the D2L traffic volume, we count the individual requests made to the D2L IP (199.30.181.42) in our hourly monitor logs, i.e., whenever we see a D2L IP as the destination IP in our monitor, we count it as a D2L request. In the HTTP and SSL logs, the destination IP is obtained from the `id.resp_h` field. We analyze these requests to obtain the yearly D2L request distribution, and then narrow it down to monthly, weekly, and hourly D2L traffic distributions. This analysis does not focus on the D2L requests made by unique IPs or user-agents. Instead, we focus on the total number of requests made to D2L. We are counting every line of the monitor log that has the destination D2L IP, indicating a request made to the D2L server. There could be multiple requests made to the D2L server in a single session. Our monitor logs a request made to a server as a separate line at every timestamp. The yearly patterns are discussed later in this section.

Our analysis of HTTP traffic was performed using the HTTP logs, and the HTTPS traffic was analyzed using the SSL logs. We searched for these connections made with the server IP of interest

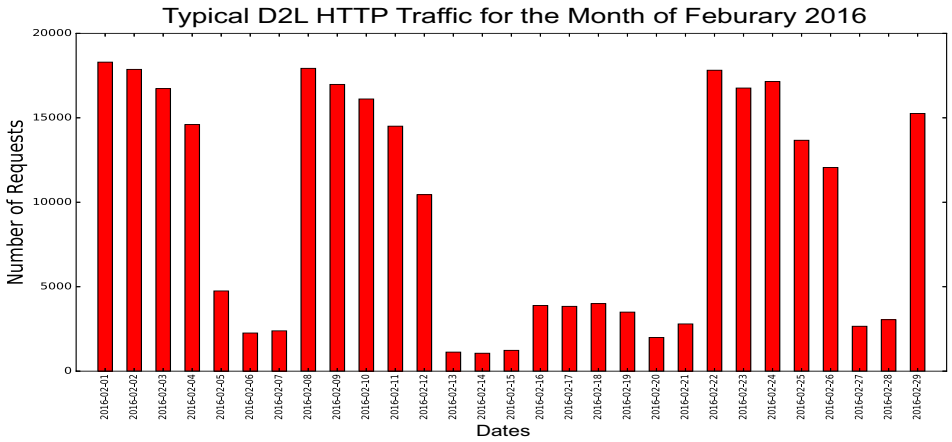
(199.30.181.42) in our monitor. The SSL logs help us in identifying the secure connections made with D2L, but the disadvantage is the difficulty in figuring out traffic volume information, since the respective bytes transferred are not listed there. During our investigation of HTTPS connections, we observed that the application-level details are minimal. There are no request-response pairs, URIs, or header information in our SSL logs. The lack of headers makes it challenging to gain an in-depth overview of the type of content accessed (videos, files, text documents, etc).

We first study the typical semester, monthly, and weekly patterns for HTTP traffic in Figure 5.1, and then for HTTPS traffic in Figure 5.2. We start with a coarse granularity view of the semester pattern, to show a summary of D2L requests made from a broader perspective. We gradually refine the level of granularity to show the monthly and weekly D2L request patterns, and at the finest level of granularity, we show the number of D2L requests on an hourly basis. For this analysis, our observation period is between January 2016 to April 2016 (Winter 2016 term). The purpose of this analysis is to understand the statistics for semester-wide, monthly, daily, and hourly D2L usage.

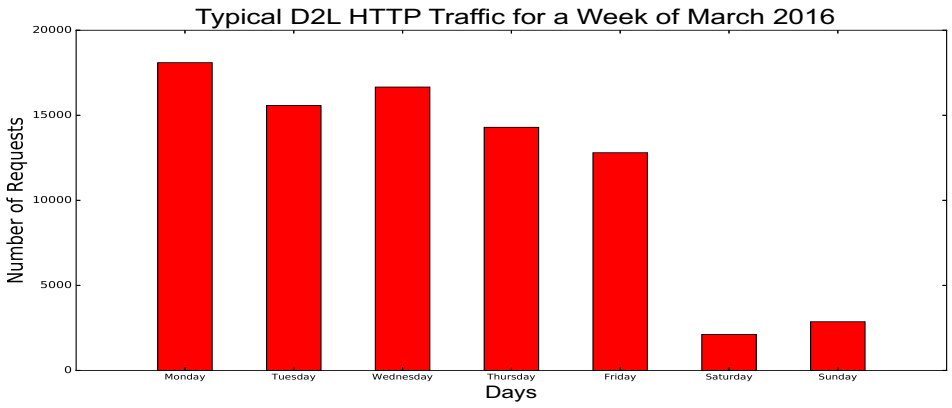
Over the entire Winter 2016 term, a broad overview of the D2L traffic can be seen in Figure 5.1a and Figure 5.2a. The low traffic towards the beginning of the year in early January is because the Winter 2016 term started on Monday, January 4, 2016. It generally takes a couple of weeks for a course curriculum and student activities to get up to speed. Also the Block Week was towards the beginning of the year, thus D2L activity was minimal during that period of time. As soon as lectures began on January 11, the D2L traffic started rising. The traffic patterns continue like this into February, including a dip during Reading Week break. This D2L traffic pattern varies for every month throughout the year, and there is no general monthly pattern. During the Reading Week break between February 14 and February 21 (middle of February) a reason for the low traffic seen could be due to the fact that generally no new content or updates are published on D2L during holidays, and thus students do not check D2L as often.




(a) D2L HTTP traffic pattern for Winter 2016 Semester




(b) D2L HTTP traffic pattern for February 2016




(c) D2L HTTP traffic pattern for March 7-13, 2016

Figure 5.1: Typical Semester, Monthly, and Weekly HTTP D2L traffic patterns

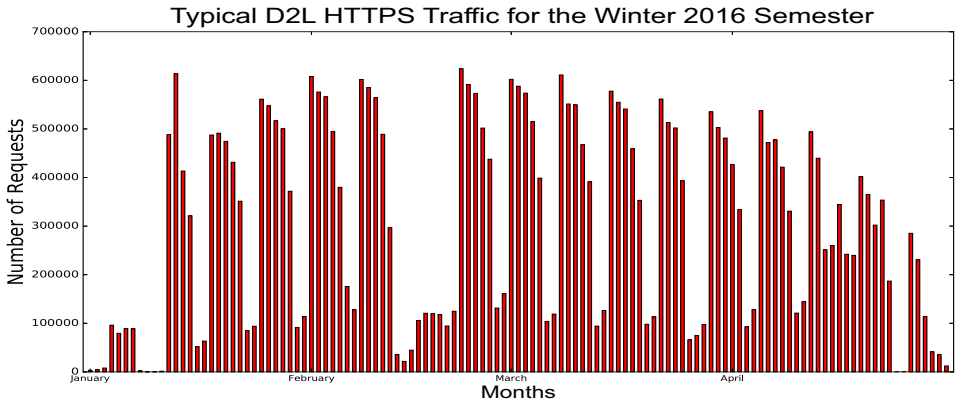
Figure 5.1b and Figure 5.2b illustrate the monthly D2L request patterns in February 2016. During the Reading Week, low levels of network traffic are seen for both HTTP and HTTPS.

The weekly HTTP and HTTPS traffic patterns are shown in Figure 5.1c and Figure 5.2c. These graphs show evidence of a human-generated traffic pattern as opposed to machine-generated behaviour by proxy servers or Web robots. People tend to generate more traffic during the weekdays, and D2L activities decrease as we approach the weekends. We show a week's traffic that starts from Monday (March 7, 2016), which tends to be a busy work day, and ends on Sunday (March 13, 2016). The D2L traffic is generally very low on weekends, i.e., on Saturdays and Sundays.

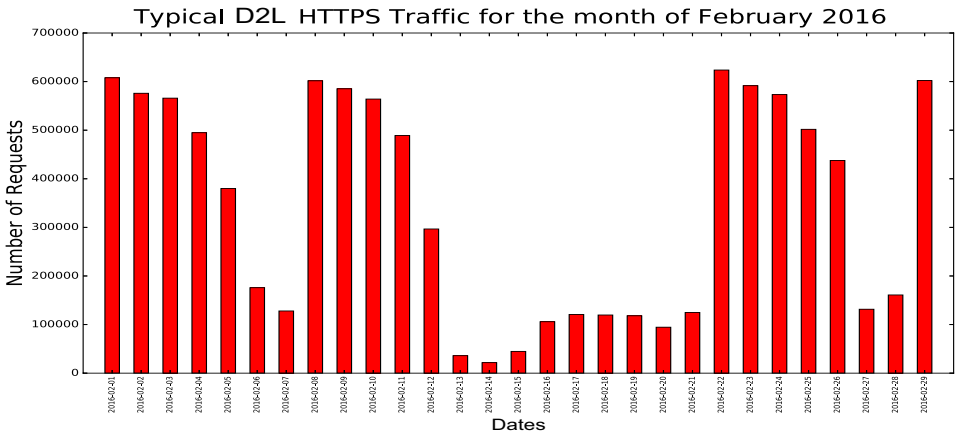
The typical D2L hourly traffic patterns for both HTTP and HTTPS are shown next. This analysis helps us understand the peak traffic hours, and also the hour of a day with least traffic. We focus on the per-hour HTTP and HTTPS D2L traffic for a few specific situations, namely a normal university week day, a weekend, a holiday, a Block Week day, and one day during the final examination period. Classes ended on April 13, 2016, after which the final examinations started. Table 5.1 details the important dates in the academic calendar.

Table 5.1: Winter 2016 dates of significance

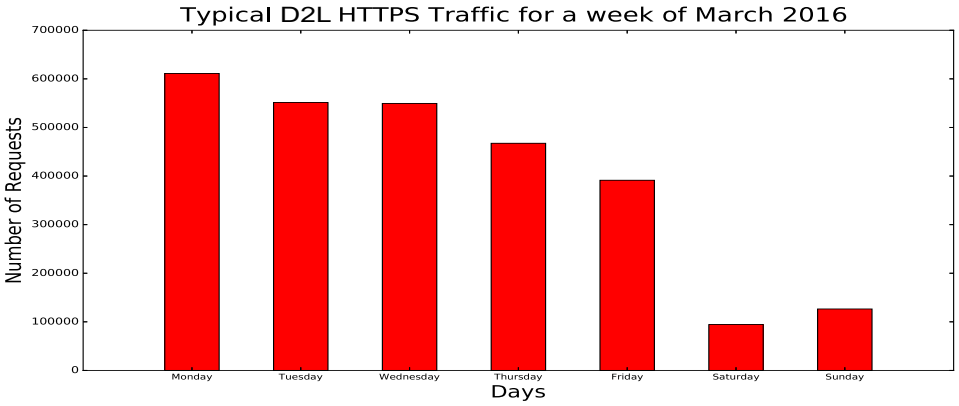
| Dates | Activities |
|---|---|
| January 4 | Beginning of Winter 2016 term |
| January 4 - January 8 | Block Week |
| January 11 | Beginning of lectures |
| February 14 - February 21 | Reading week |
| February 15 | Family Day Holiday |
| March 25 | Easter Holiday |
| April 13 | Beginning of Final Examinations |




(a) D2L HTTPS traffic pattern for Winter 2016 Semester




(b) D2L HTTPS traffic pattern for February 2016




(c) D2L HTTPS traffic pattern for March 7-13, 2016

Figure 5.2: Typical Semester, Monthly, and Weekly HTTPS traffic patterns

We perform hourly HTTP and HTTPS D2L traffic analysis, based on the dates in Table 5.2. We choose these dates to check how D2L usage changes based on normal dates, and specific U of C curriculum dates.

Table 5.2: Dates chosen for D2L HTTP and HTTPS traffic analysis

| Dates | Activities | Figure |
|---|---|---|
| Thursday, January 7, 2016 | Block Week Day | Figure 5.3 |
| Monday, February 15, 2016 | Holiday (Family Day) | Figure 5.4 |
| Friday, March 4, 2016 | Normal Week Day | Figure 5.5 |
| Saturday, March 26, 2016 | Weekend Day | Figure 5.6 |
| Tuesday, April 19, 2016 | A day from the Final Examinations period | Figure 5.7 |

There are five dates mentioned in Table 5.2, and we contrast the HTTP / HTTPS D2L hourly patterns by placing them in the same figures, and setting the vertical axis to logarithmic scales, which produces visible results for both HTTP and HTTPS D2L traffic.

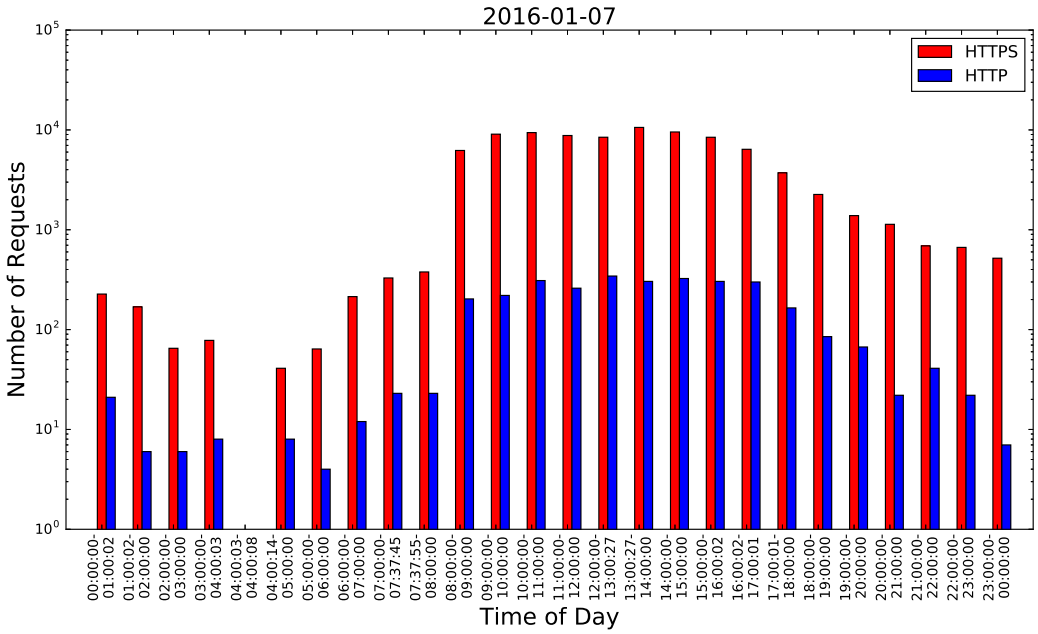



Figure 5.3: Block Week Day

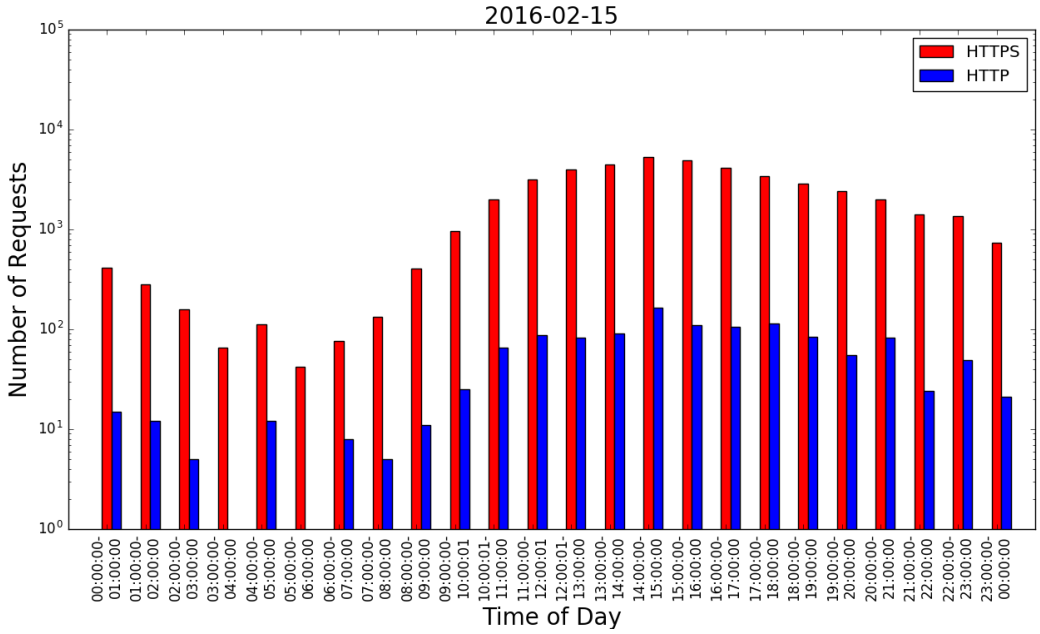



Figure 5.4: Holiday (Family Day)

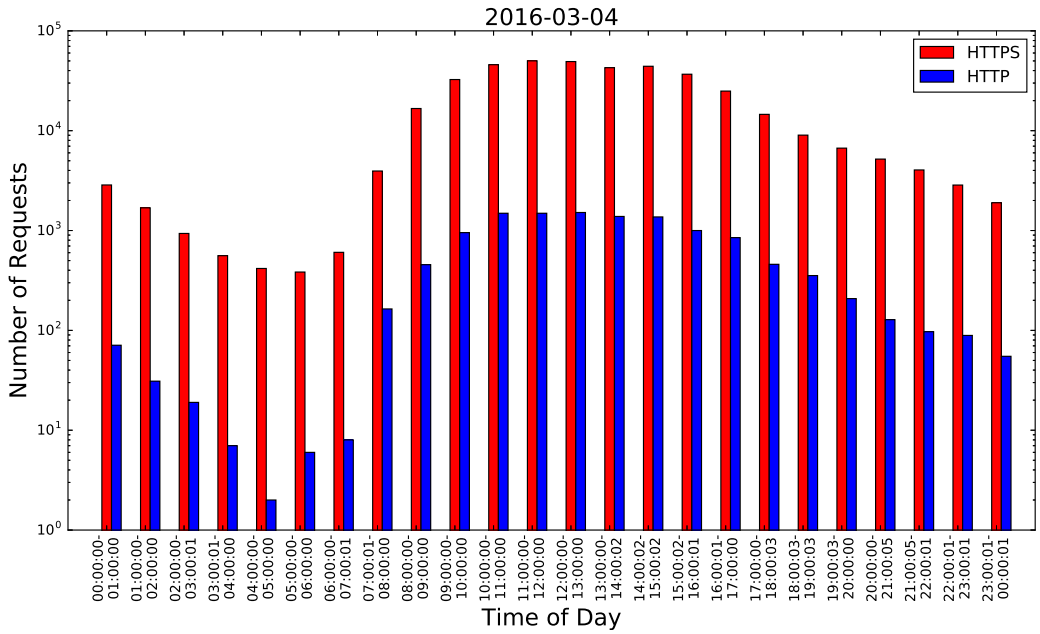



Figure 5.5: Normal Week Day

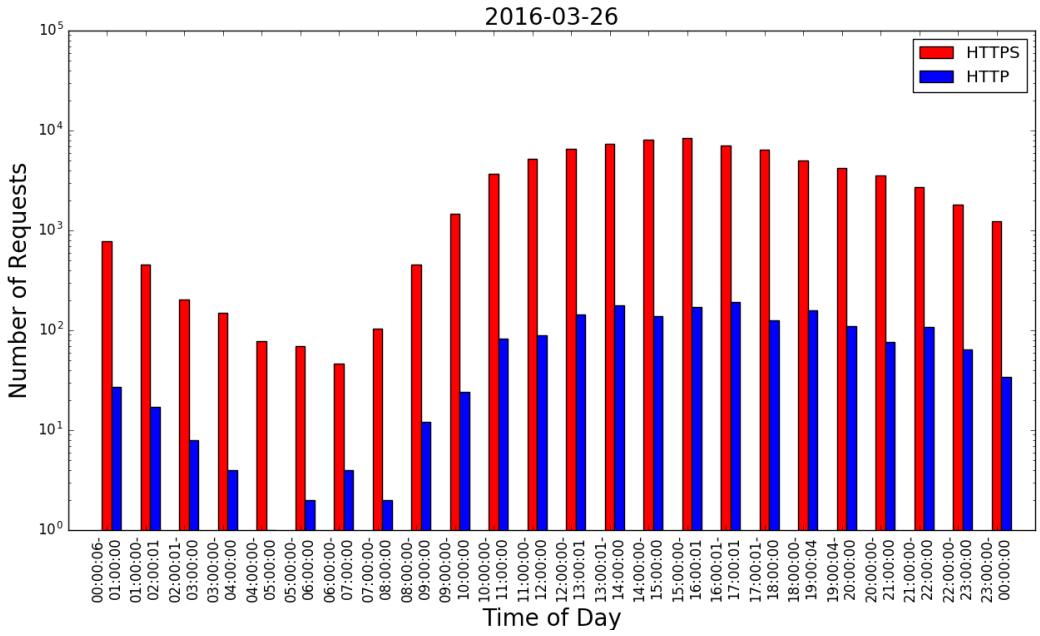



Figure 5.6: Weekend Day

The normal weekday helps us understand the typical diurnal pattern recurring almost every day, other than the special dates mentioned in Table 5.1. Figure 5.5 shows that D2L traffic is light in the early mornings, and then is fairly strong from 9:00 am to 5:00 pm, before tapering off gradually in the evening. This pattern is consistent in all five graphs.

During a normal week day in Figure 5.5, the maximum HTTPS and HTTP traffic seen per hour was (50,000 requests) almost five times the maximum number of requests per hour (10,000 requests) on a Block Week day in Figure 5.3. The lower volume is because classes had not started yet for the Winter 2016 term, as mentioned earlier.

Weekends and holidays have very low network traffic for D2L, with 5000-8000 HTTPS requests per hour in Figure 5.6 and Figure 5.4, and up to 175 D2L HTTP requests per hour. This in-depth analysis provides us with a detailed overview of the D2L traffic volume during specific periods of the academic semester.

**Relation between HTTP and HTTPS traffic patterns**

In general, the HTTPS traffic for D2L exceeds that of HTTP by an order of magnitude, as

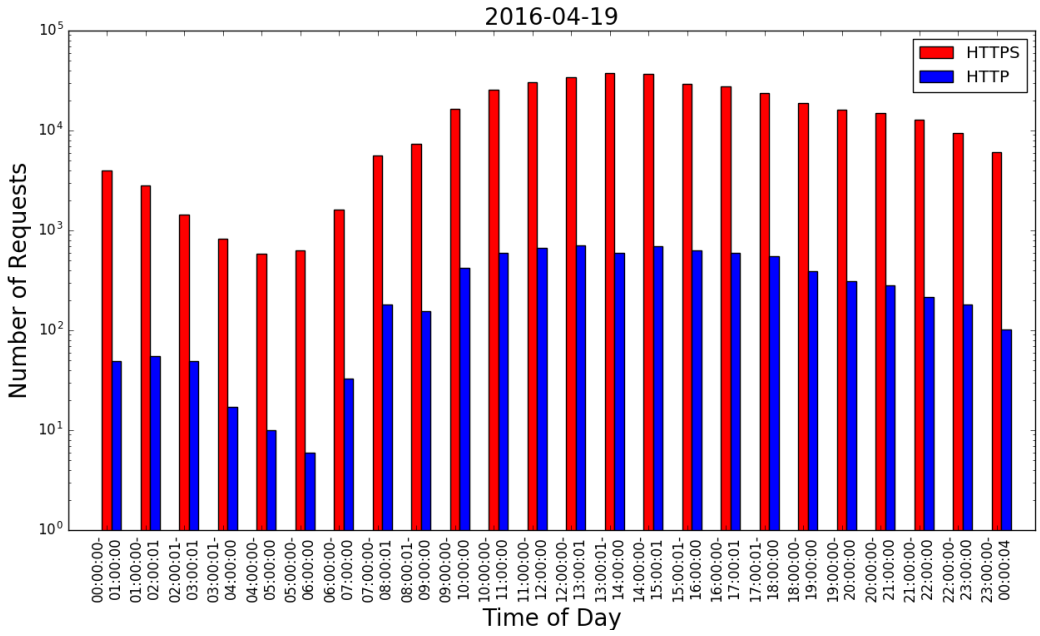



Figure 5.7: A day from the Final Examinations period

evident from the five graphs. The reason for this was explained earlier in the session diagrams for D2L in Section 4.4.1. We see a few HTTP requests at the start and end of every session, showing various image files, CSS files, D2L logo, and other JavaScript format files. Also, the HTTP requests are typically indicative of a logout process, whereas D2L transactions take place via HTTPS throughout the session, which accounts for the larger HTTPS traffic count. This in turn explains the larger number of D2L HTTPS entries seen in our monitor's SSL logs.

If we look at the patterns carefully in Figure 5.1a and Figure 5.2a, we see a similar pattern for both HTTP and HTTPS, with the difference only in the number of requests made. The reason that we see any HTTP activity at all is because a user logs out of D2L. During the logout process, every user gets redirected to the logout page on the TITL Web server. Since the transactions with the TITL Web server take place over HTTP, the files accessed during the logout process are visible. Our session diagrams in Figure 4.14, Figure 4.18, Figure 4.19, and Figure 4.20 show the HTTP requests made to the TITL server towards the end of each of our session experiments.

In Figure 5.1b, the maximum number of HTTP requests are seen on Mondays. The total of approximately 18,000 D2L requests per day is much less than the total number of secure connec-

tions made to D2L. Similarly, in Figure 5.2b the maximum number of HTTPS requests are seen on Mondays, with a total of almost 600,000 D2L requests per day. This clearly indicates that most of the connections made with D2L are over secure HTTP. On average, there are about 30 HTTPS transactions for every HTTP transaction seen.

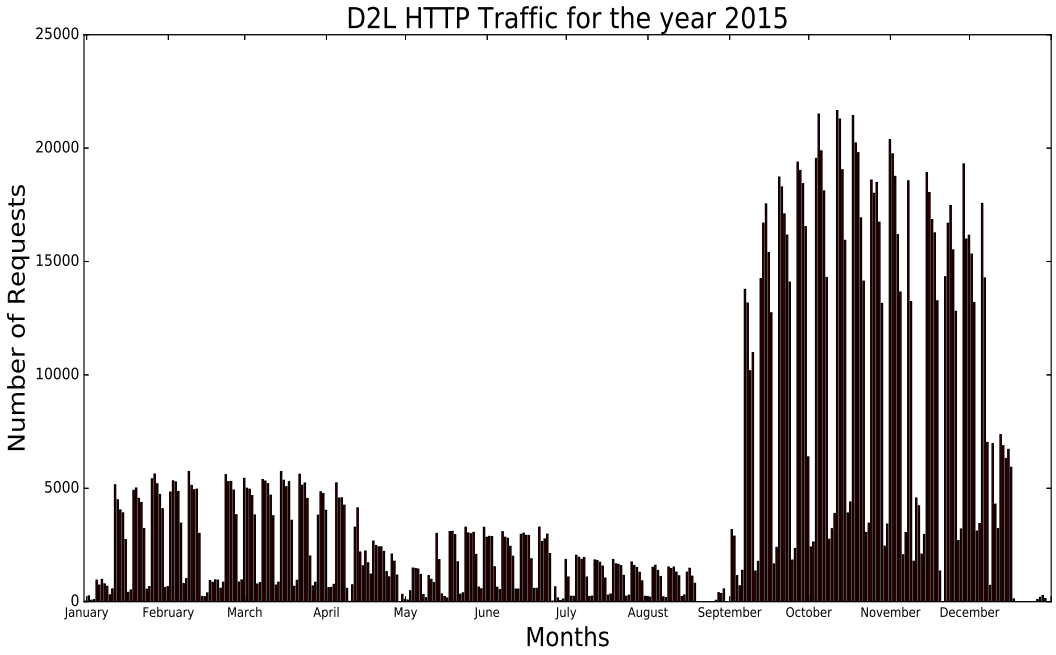
## 5.2 Longitudinal look at D2L traffic volumes

In this section, we compare HTTP and HTTPS D2L traffic patterns for 2015 and 2016. This analysis would help us understand if there was an uptake in D2L usage.

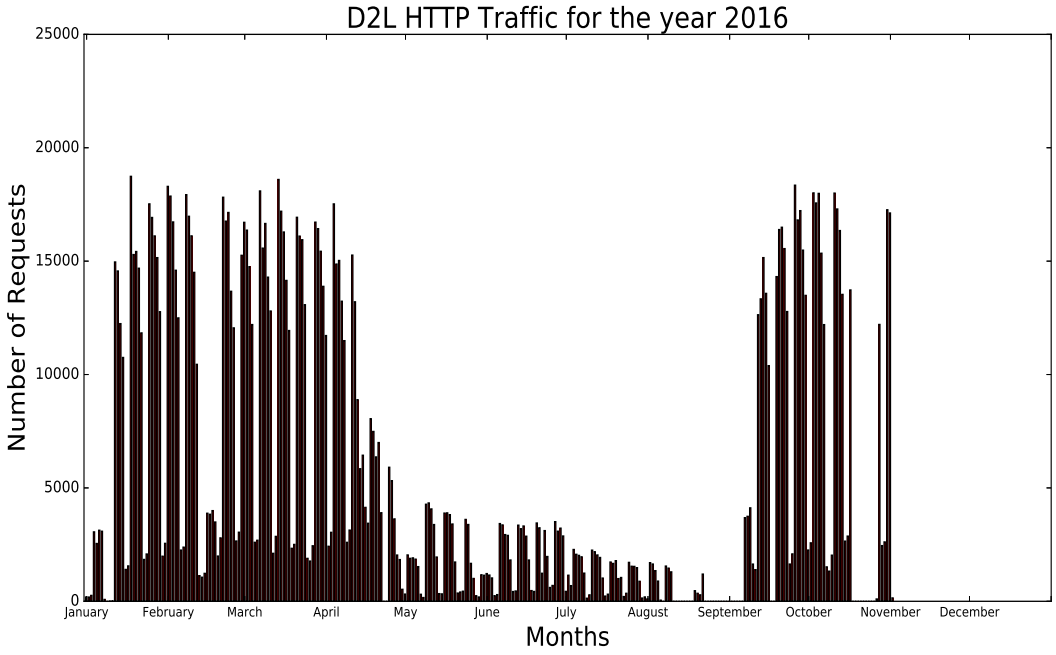
In Figure 5.8a, the Winter, Spring and Summer terms of 2015 have very low HTTP traffic, with a maximum of 5000 requests per day in the Winter term and 4000 requests per day in Spring/Summer 2015. The D2L logout URL `http://elearn.ucalgary.ca/desire2learn/log-out/` is seen in our monitor's HTTP logs every time a user logs out. Further analysis clarifies that towards the beginning of 2015 there was a different D2L logout process in place since this logout URL did not show up even once in the HTTP monitor logs during the Winter 2015 term. Also, during this period the larger HTTPS traffic seen compared to 2016 Winter term HTTPS traffic could be because the logouts happened directly through D2L using HTTPS. The HTTP requests show a maximum of 20,000 requests per day during the Fall 2015 term.

Figure 5.8b for HTTP traffic shows that the maximum number of requests are 18,000 per day during the Winter and Fall terms of 2016. There is low traffic of less than 5000 requests per day during the Spring / Summer terms, when the number of students enrolled in D2L courses was much lower compared to the Fall / Winter terms. For the year 2016, there was a network outage in the middle of November 2016, following which our monitor logs were disrupted until February 2017.

In contrast to the HTTP traffic for Winter 2015, the HTTPS traffic in Figure 5.9a shows a larger number of D2L requests. The maximum number of requests was 780,000 per day for the Winter 2015 term, then the traffic declines significantly during the Spring/Summer terms with about 100,000 requests per day.

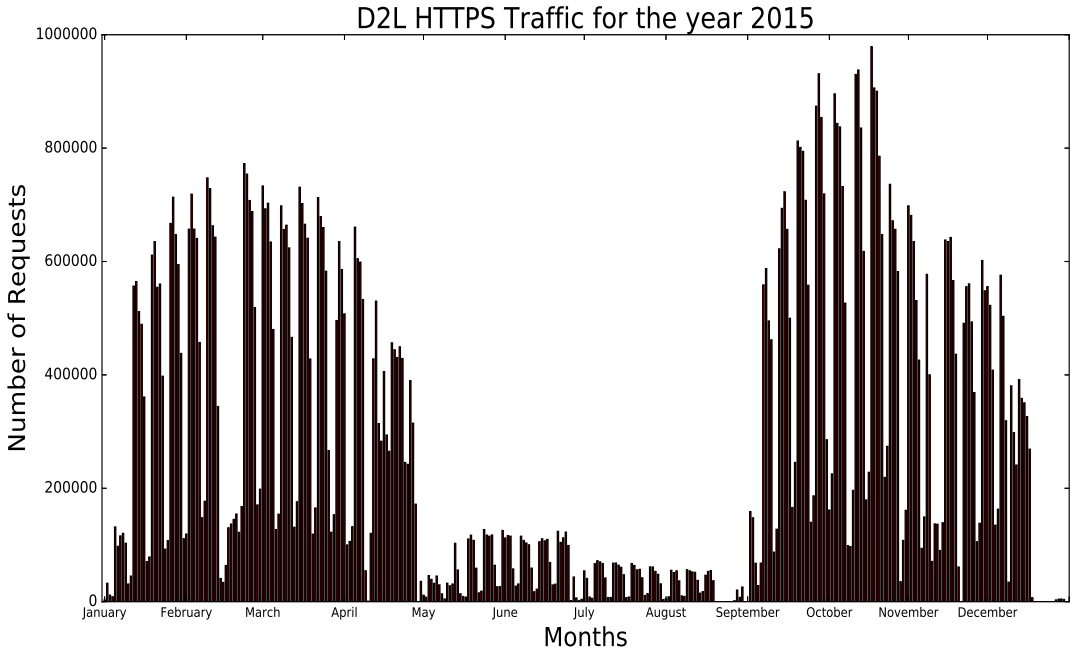



(a) 2015

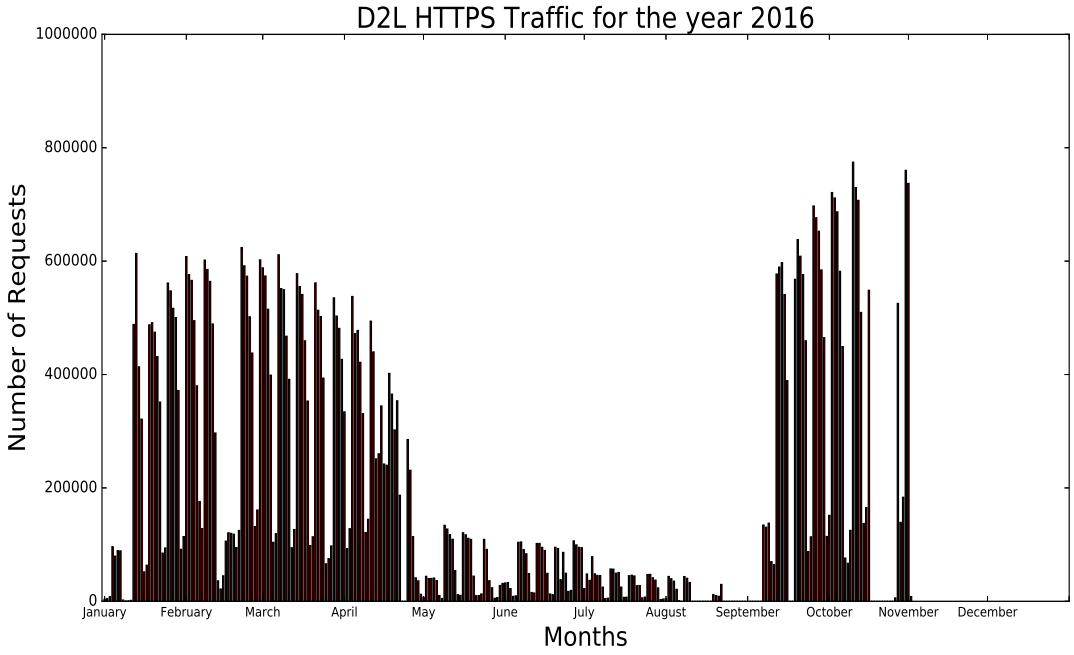



(b) 2016

Figure 5.8: D2L HTTP traffic pattern for two years




(a) 2015




(b) 2016

Figure 5.9: D2L HTTPS traffic pattern for two years

The volume increases again towards the start of the Fall 2015 term, with a maximum of almost 1,000,000 requests per day. The traffic gradually lowers towards the end of the year.

For the year 2016, in Figure 5.9b, the Winter term shows high traffic with a maximum of 600,000 requests per day, which is lower than the Fall 2016 term, which has a maximum of almost 800,000 requests per day. The Spring / Summer terms have very low traffic, with an average of less than 100,000 requests per day.

With the help of this analysis, we see that Fall term is the largest contributor for D2L traffic for both years. Also, the 2016 HTTPS traffic for the Fall term is less than the 2015 HTTPS traffic, which shows a slight decrease in the number of D2L requests for 2016. Due to U of C network outages, the traffic is only visible until early November 2016, which makes it difficult to comment on the D2L usage for Fall 2016. Another reason could be that there were fewer students enrolled in D2L courses in 2016 compared to 2015, there were lesser D2L courses offered, or even due to the frequent U of C network upgrades throughout 2016.

## 5.3 IP analysis

The IP analysis shows which IPs show up more frequently in our monitors. Based on our analysis in Chapter 4, we anticipate that wireless IPs are generated mostly by DHCP servers.

During our 4 months of observation, over 10,000 unique IP addresses visited D2L. The top 10 IPs, along with their counts, are shown in Table 5.3. These IPs were obtained from the source IP information `id.orig_h` in our HTTP logs, which have a destination IP `id.resp_h` of 199.30.181.42 (D2L IP). This analysis is for the Winter 2016 term.

Table 5.4 shows the HTTPS count of the top IPs seen throughout the Winter 2016 term. 11,179 unique IPs accessed D2L through HTTPS during these four months.

**IP frequency-rank analysis**

A frequency-rank plot is the distribution of frequency by rank, in decreasing order of frequency, where the source data are from a frequency distribution.

Table 5.3: Top HTTP IPs requesting D2L (Winter 2016)

| Rank | IP | Count |
|---|---|---|
| 1 | 136.159.49.121 | 40,544 |
| 2 | 136.159.49.123 | 38,915 |
| 3 | 136.159.49.126 | 38,889 |
| 4 | 136.159.49.124 | 38,616 |
| 5 | 136.159.49.117 | 38,458 |
| 6 | 136.159.49.119 | 38,427 |
| 7 | 136.159.49.114 | 38,237 |
| 8 | 136.159.49.115 | 37,763 |
| 9 | 136.159.49.120 | 37,658 |
| 10 | 136.159.49.122 | 37,308 |

We performed IP frequency-rank analysis for HTTP connections with D2L, as reflected in the HTTP logs in our monitor. The results show that the top 10 most popular IPs are the same as the IPs seen for HTTPS connections, which suggests that these IPs were generated from a pool of available IPs.

Figure 5.10 shows an HTTP IP frequency-rank analysis for one semester (Winter 2016), one month (March 2016), one week (March 1-7, 2016), and one day (March 22, 2016). Over these four months, the number of unique HTTP IPs was 7,628.

The IP frequency-rank analysis identifies a visible plateau with the IPs on the 136.159.49.0/24 subnet occupying the higher ranks. Figure 5.11 shows a frequency-rank analysis applied to the HTTPS traffic of the D2L site. These IPs are all responsible for HTTPS connections with D2L as seen in our SSL logs. These IPs were obtained from the source IP information `id.orig_h` in our SSL logs, which have a destination D2L IP in the `id.resp_h` field.

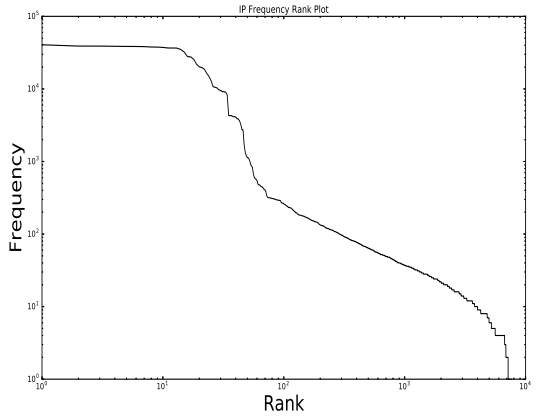
The IP frequency rank graphs in Figure 5.10 and Figure 5.11 show a three-piecewise power-law structure. The first plateau represents only the wireless IPs generated by DHCP servers, the second portion identifies mainly the wired NAT IP subnets, and the final portion shows a human-generated power-law structure towards the tail, most of which are RezNet-Secure IPs. We can see that the top 20 to 50 IP addresses behave quite differently from the others since their number of connections is much higher, which suggests that they might be aggregators using NAT or DHCP. Furthermore,

the graphs are almost flat across these IPs, suggesting that some sort of round-robin load-balancing is being used across this IP infrastructure. The transition into the 'human' part of the plot is much smoother. This analysis shows a clear distinction between human and machine behavior.

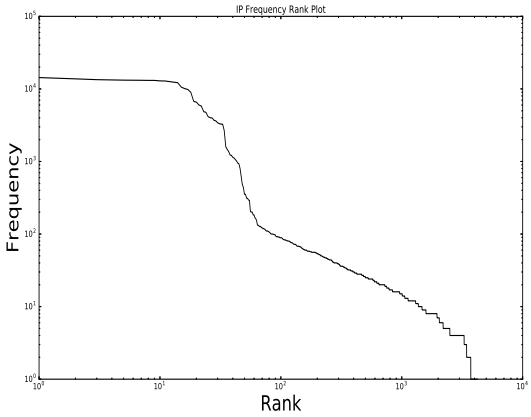
Table 5.4: Top HTTPS IPs requesting D2L (Winter 2016)

| Rank | IP | Count |
|---|---|---|
| 1 | 136.159.49.121 | 1,899,239 |
| 2 | 136.159.49.124 | 1,854,487 |
| 3 | 136.159.49.119 | 1,844,189 |
| 4 | 136.159.49.126 | 1,826,598 |
| 5 | 136.159.49.120 | 1,823,579 |
| 6 | 136.159.49.114 | 1,810,781 |
| 7 | 136.159.49.123 | 1,810,070 |
| 8 | 136.159.49.117 | 1,801,712 |
| 9 | 136.159.49.115 | 1,786,548 |
| 10 | 136.159.49.125 | 1,758,725 |

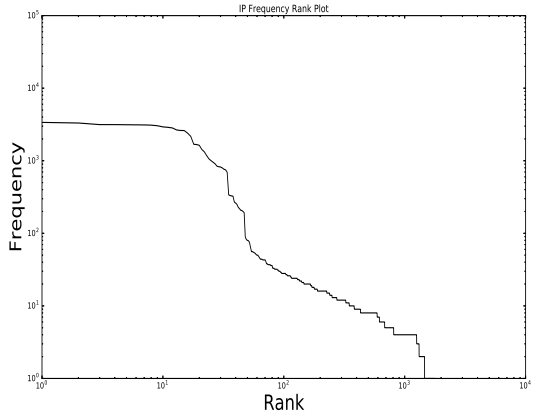
When we connect our portable devices with the available U of C WiFi / wireless access points, for example, `eduroam`, or `airuc-secure`, we receive an IP address from the 136.159.49.0/24 or 136.159.160.0/24 subnets. We believe that the 136.159.49.0/24 and 136.159.160.0/24 subnets have been assigned for most of the on-campus wireless connections generated through DHCP servers. We find 82 and 99 DHCP generated IPs in our HTTP and SSL logs respectively. In comparison only 7 wired NAT generated IPs are seen in each of the HTTP and SSL logs. This indicates that these DHCP servers or NAT-enabled devices provide a pool of available IPs, and since these IPs are reusable, the counts are high. The top IPs for D2L requests are always from the campus WiFi (`airuc-secure`), which indicates the popularity of wireless connectivity inside our campus network infrastructure. The current sizing of IT infrastructure include at least two DHCP servers. For HTTP, the cumulative frequency for NAT generated IPs is 36,724 and for DHCP IPs is 856,682. For HTTPs however, a much higher cumulative frequency for both NAT generated IPs with 831,804 and for DHCP IPs with a value of 31,469,555 is seen.



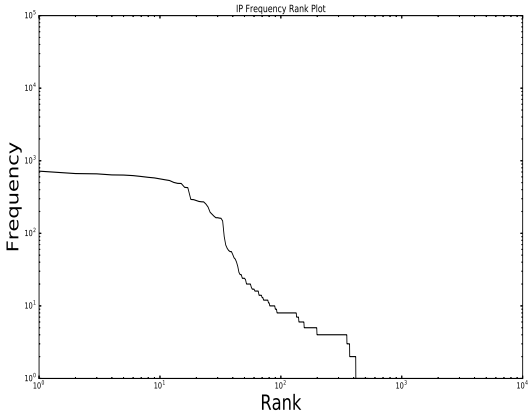

(a) One semester (Winter 2016 Term)




(b) One month (March 2016)



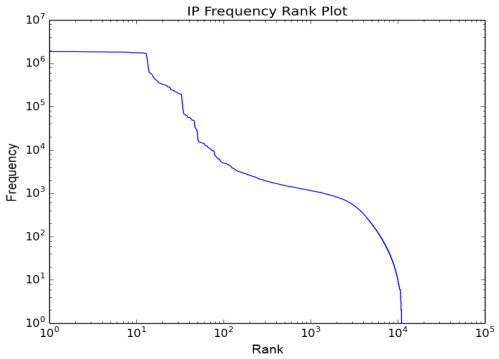

(c) One week (March 1-7, 2016)



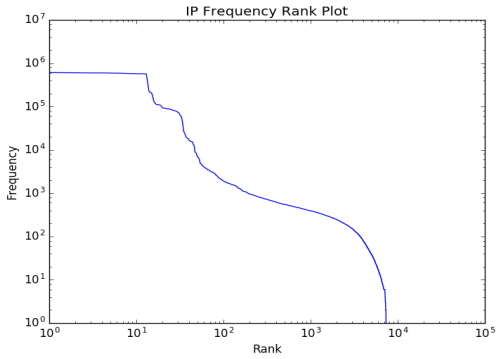

(d) One day (March 22, 2016)

Figure 5.10: HTTP IP Frequency Rank plot

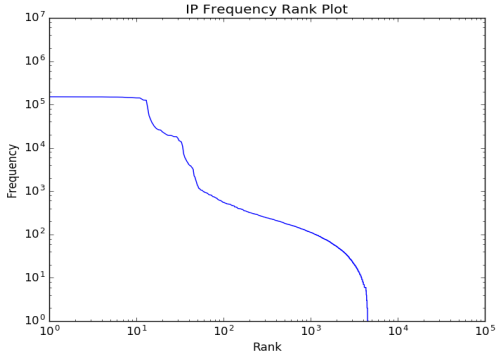
We also see IP addresses from the 136.159.132.0/24 subnet while connecting through `reznet-secure`. These reznet IPs are mostly assigned to user devices connecting through the U of C on-campus residence buildings. These IPs show up towards the tail of the graph.



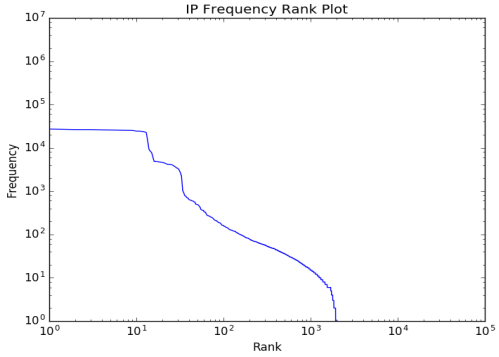

(a) One Semester (Winter 2016 Term)




(b) One month (March 2016)




(c) One week (March 1-7, 2016)




(d) One day (March 22, 2016)

Figure 5.11: HTTPS IP Frequency Rank plot

## 5.4 User Agent Information

The vast majority of users accessing D2L are humans, which makes a user agent analysis worthwhile. The user agent is software that acts on behalf of a user. The user agent request header contains a user-agent string. This user-agent string allows network peers to identify the operating system, browser information, application type, software version, or software vendor of the requesting software user agent. Thus along with identifying our browsers, it provides certain system details to servers hosting the Web sites that we visit.

In this section, we provide a detailed discussion regarding the typical browser, OS, and devices of D2L users. The D2L site is mostly used by students, faculty, and staff. This makes it useful to understand the user statistics for their preferred browsers or Operating System (OS). For gathering user agent and operating system information, we have used an on-line application program

interface, namely 'User Agent String.com' [1], which is primarily a user agent database.

Table 5.5 shows the top 10 most popular user agents for the D2L site over a period of four months (Winter 2016). The top 10 user agents are Mozilla 5.0. Nine of the agents are using the Windows NT operating system. They account for a very low percentage because of the variety of user agents seen in these four months.

Table 5.5: Top 10 user agents for the Winter 2016 term

| User Agents | Requests | Total Pct. |
|---|---|---|
| Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko | 57,113 | 5.37% |
| Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0 | 39,769 | 3.74% |
| Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36 | 25,461 | 2.39% |
| Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36 | 21,622 | 2.03% |
| Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36 | 21,266 | 2.00 % |
| Mozilla/5.0 (Windows NT 6.1; rv:34.0) Gecko/20100101 Firefox/34.0 | 19,081 | 1.79 % |
| Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko | 17,756 | 1.67 % |
| Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/601.4.4 (KHTML, like Gecko) Version/9.0.3 Safari/601.4.4 | 17,471 | 1.64% |
| Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36 | 16,589 | 1.56% |
| Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36 | 15,875 | 1.49% |

Our analysis of the monitor's HTTP log from January to April 2016 shows that 99.89% (1,061,931) of the requests were made using a browser, and only 12 requests were made using a Web crawler. Almost all of the observed user agents belong to the 'browser' category. Thus we did an analysis of the user agents in this category in Figure 5.12.

Among all the browsers, about half of the D2L users (49.3%) use Chrome. Other popular browsers include Firefox, Safari, Internet Explorer and Android Webkit Browser. Table 5.6 shows the top browsers and popular versions of browsers observed in HTTP requests.

[1] `http://www.useragentstring.com/`

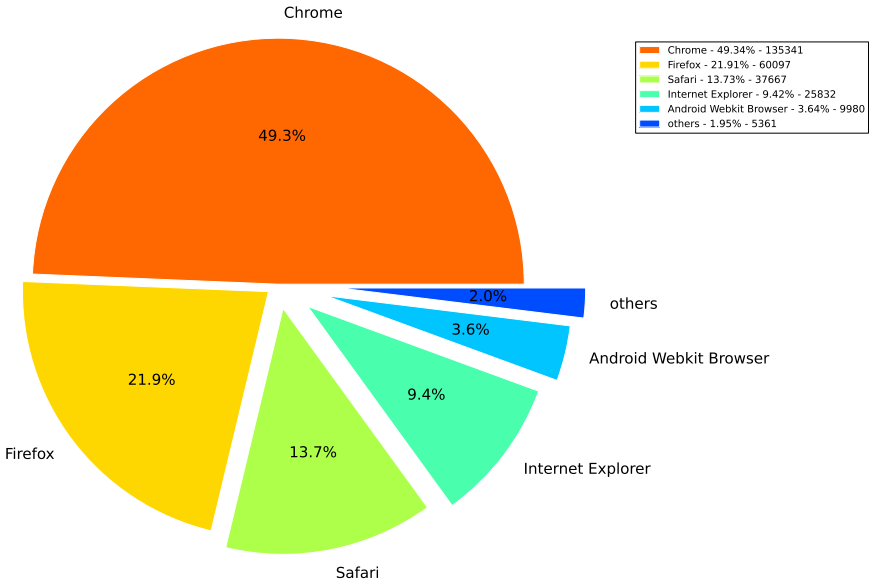



Figure 5.12: Top browsers based on HTTP requests

Figure 5.13 illustrates a classification of the types of operating system for all the user agents. Microsoft Windows is the top operating system, generating more than half (64.5%) of the requests, followed by Macintosh OS X on Apple computers, and the iPhone OS (iOS), which runs on all of Apple's portable mobile devices (iPhone, iPad, and iPod touch). The Macintosh produces 27.2% of the requests, followed by Android (4.3%) and Linux (3.6%).

Figure 5.14 shows the distribution of the most popular operating systems of each type, based on HTTP requests.

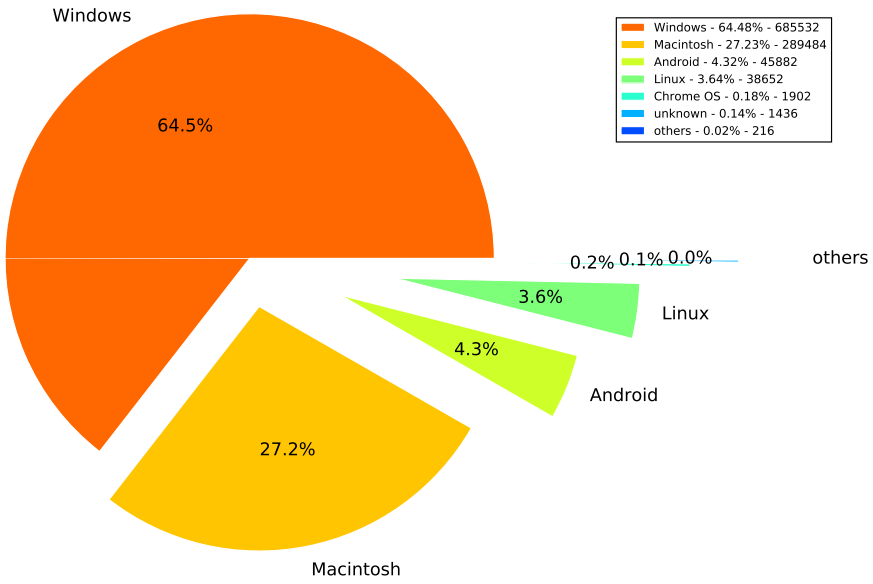



Figure 5.13: Types of OS

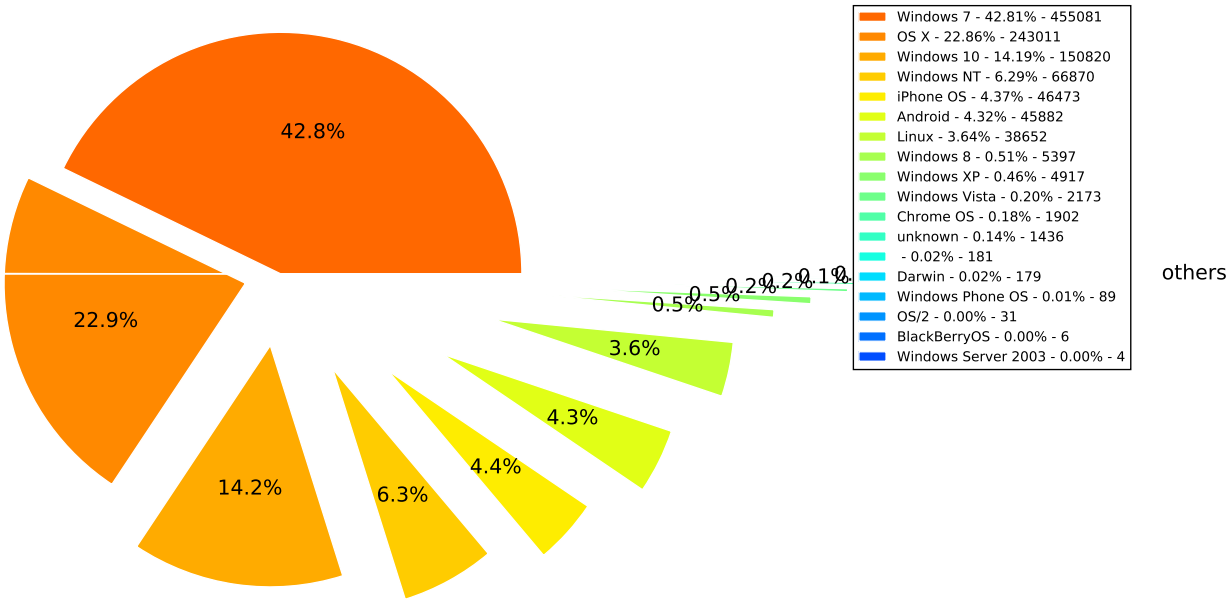



Figure 5.14: Classification of OS types

Each category of the operating systems in Figure 5.14 are further discussed below:

1) For Windows users, 42.8% of the requests were generated by Windows 7, 14.19% by Windows 10, 6.29% by Windows NT, 0.51% by Windows 8, 0.46% by Windows XP, 0.20% by Windows Vista, 0.01% by Windows Phone, and just 4 requests by Windows Server 2003.

2) For Apple, 22.86% of the requests were issued by OS X users, and 4.37% by iOS.

3) The use of the Android OS is minimal with 4.32% of the requests accessing D2L. Amongst others, Chrome OS has 0.18%, Darwin has 0.02% of the total requests, OS/2 has 31 requests and finally BlackBerry OS and Windows Server 2003 have a cumulative total of 10 requests.

Table 5.6: Top 5 Browser Versions

| Browser Version | Reqs | Pct. |
|---|---|---|
| 48.0.2564.116 | 43,209 | 31.93% |
| 48.0.2564.97 | 2,682 | 21.93% |
| 48.0.2564.103 | 26,277 | 19.42% |
| 48.0.2564.109 | 22,830 | 16.87% |
| 47.0.2526.111 | 4,025 | 2.97% |

(a) Chrome (49.3% of total Reqs)

| Browser Version | Reqs | Pct. |
|---|---|---|
| 44.0 | 28,741 | 47.82% |
| 43.0 | 14,842 | 24.70% |
| 34.0 | 5,094 | 8.48% |
| 42.0 | 4,374 | 7.28% |
| 38.0 | 2,861 | 4.76% |

(b) Firefox (21.9% of total Reqs)

| Browser Version | Reqs | Pct. |
|---|---|---|
| 9.0.3 | 12,205 | 32.40% |
| 9.0 | 8,579 | 22.78% |
| 9.0.2 | 3,952 | 10.49% |
| 8.0 | 2,062 | 5.47% |
| 9.0.1 | 1,654 | 4.39% |

(c) Safari (13.7% of total Reqs)

| Browser Version | Reqs | Pct. |
|---|---|---|
| 11.0 | 24,427 | 94.56% |
| 7.0 | 576 | 2.23% |
| 10.0 | 491 | 1.90% |
| 9.0 | 320 | 1.24% |
| 8.0 | 18 | 0.07% |

(d) Internet Explorer (9.4% of total Reqs)

## 5.5 Session-related analysis

Session durations provide more insights on user-level interaction. Session analysis was performed using the `duration` field from our connection logs. We speculate that the 2016 HTTPS traffic volume is slightly less than 2015 HTTPS traffic because of improved browser support for parallel or persistent connections. Although individual user sessions cannot be determined, a general understanding of the session durations becomes possible through this analysis. We tried to

compare the CDF of connection durations for 2015 and 2016 Winter terms. During our four-month comparison period (Winter semester), we observed that the median connection duration with D2L for both years was 51 seconds.

Figure 5.15 shows the Cumulative Distribution Function (CDF) for session durations for the month of February 2015 and February 2016. A CDF graph is useful, since it provides a summary of a large sample of data points. The vertical axis (Y-axis) of any CDF graph shows the probability, and the horizontal axis (X-axis) show what is being measured, which in this case is the connection duration. Our CDF analysis illustrates the probability of session durations less than a specific value. The two CDF lines are almost overlapping with similar median values. 90% of the session durations are less than 200 seconds. Thus, most session durations appear to be in the range of 0-200 seconds for both years.

The session duration CDF shows similar characteristics for both Winter 2015 and Winter 2016 semesters. However, a few large values were observed. The highest session duration value for the Winter 2016 term was 437,865.03 seconds, while that for the Winter 2015 term was 1,365,635.62 seconds. A more in-depth study suggests that the observed session durations are heavy-tailed. The maximum observed duration value for 2016 is however lesser than 2015, suggesting a possible session time-out, following a certain period of inactivity. This timeout feature seems to be used since 2016.

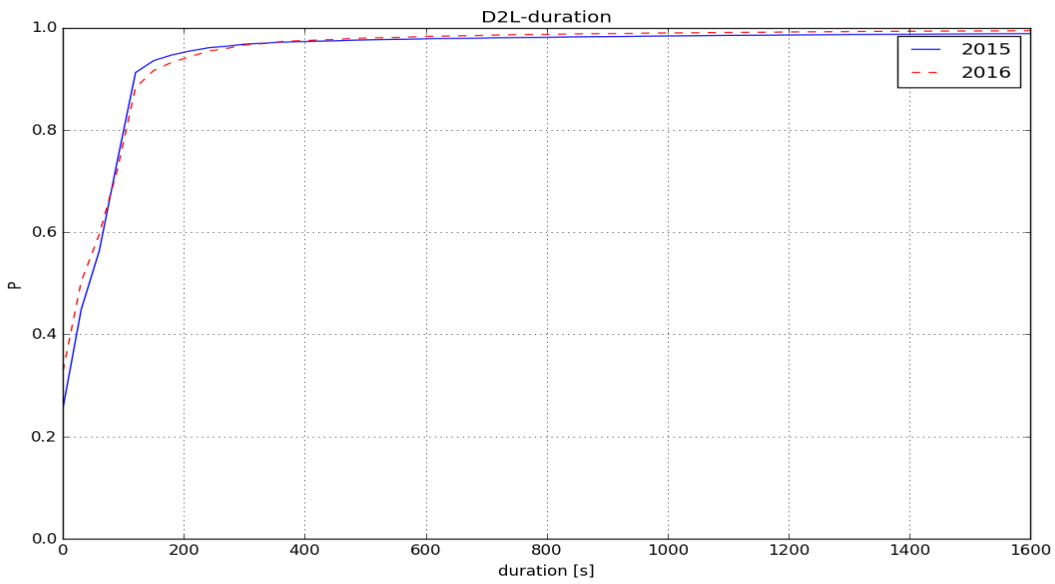



Figure 5.15: Connection Duration CDF for Winter 2015 Semester versus Winter 2016 Semester (One Semester)

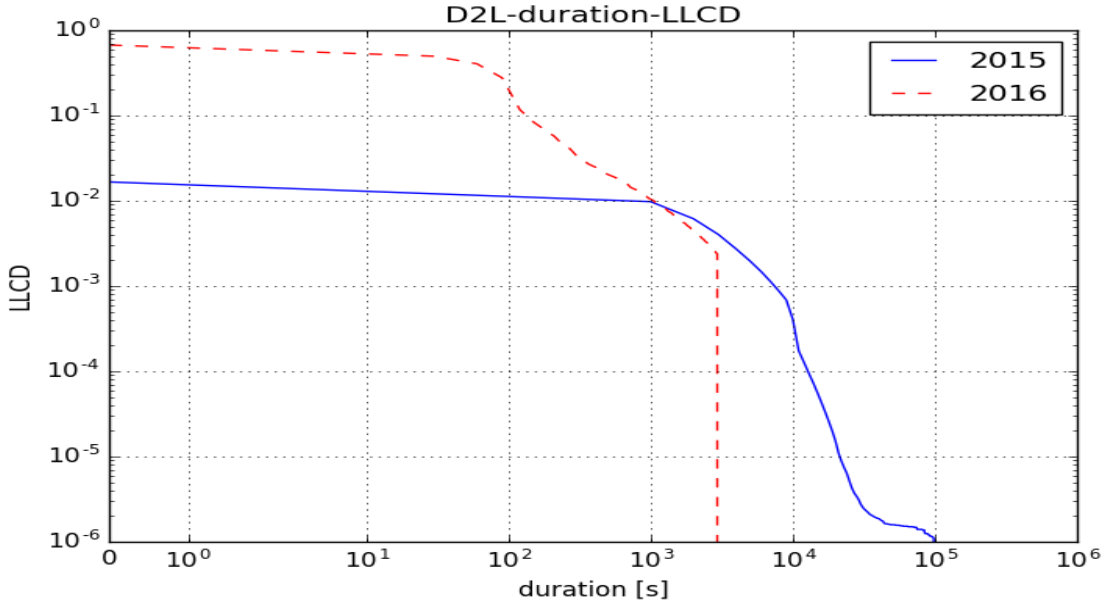



Figure 5.16: Connection Duration LLCD for Winter 2015 Semester versus Winter 2016 Semester (One Semester)

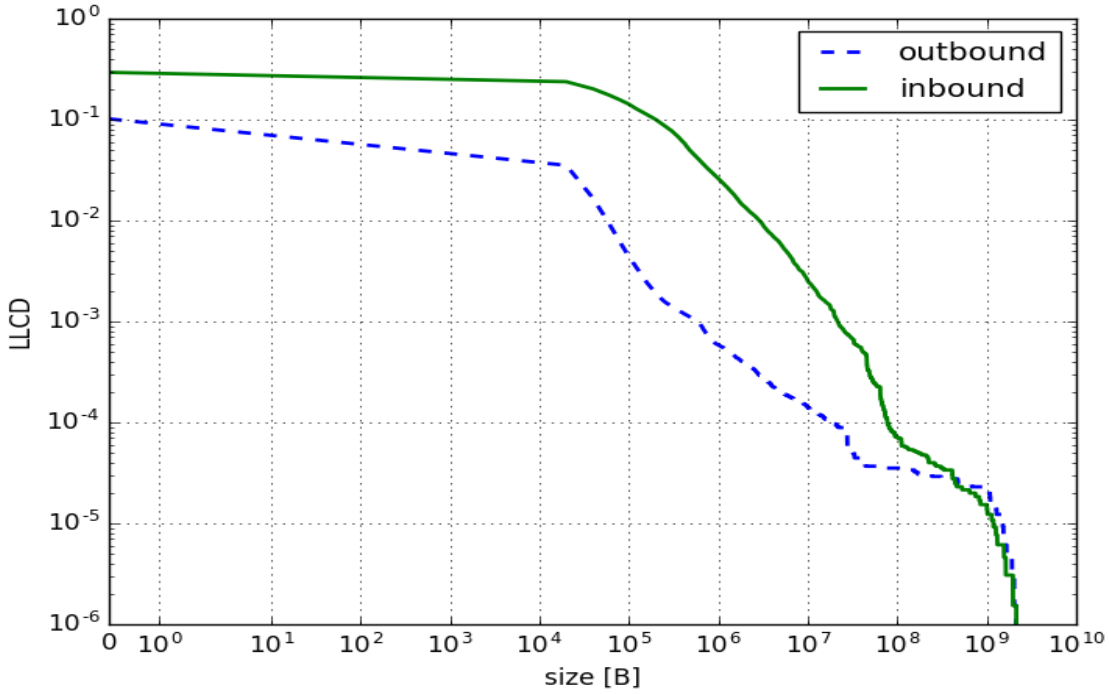
After analyzing the body of our CDF graph we constructed a Complimentary Cumulative Distribution Function (CCDF) graph where we observed that the asymptotic shape of the distribution was hyperbolic suggesting a possible heavy-tailed distribution. A CCDF graph shows us how quickly or slowly the tail of a distribution decays. We see that the tail decays slower than an exponential distribution confirming a heavy-tailed distribution [42]. To understand the heavy-tailed characteristics, we generate a log-log complementary distribution (LLCD) graph next. For brevity we do not show the CCDF graphs.

Figure 5.16 illustrates the heavy-tailed characteristic for session durations using LLCD plots. A heavy-tailed distribution tends to have many outliers (very high values) [59][71]. As the tail gets heavier, the probability of observing one or more very large values increases. Thus we see a few values as high as 380 hours at the very tail of our entire distribution for the year 2015. However, as discussed earlier, the maximum value observed in 2016 is 122 hours (due to the possible timeout value), as is evident from the graph.

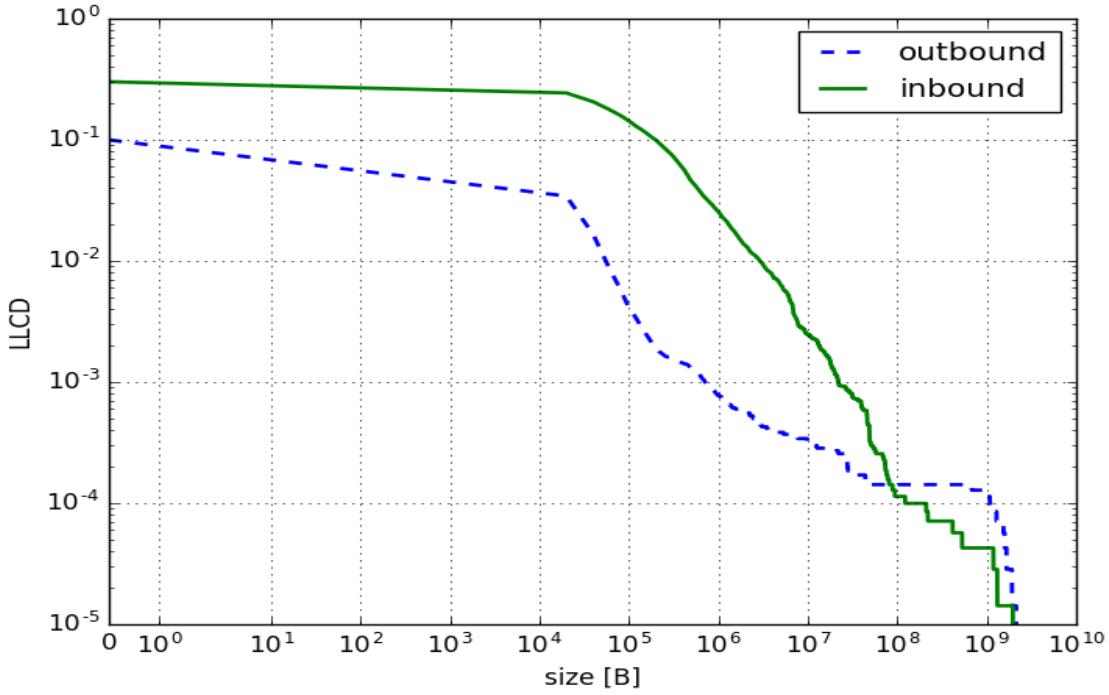
**Inbound and Outbound Traffic Analysis for a single day:**

Figure 5.17a shows the LLCD plot for inbound and outbound data per connection in bytes for Monday, February 1, 2016. A total of 651,770 D2L connections were seen on this single day. The requests made to D2L or files uploaded to D2L are considered as the outbound traffic, and the received bytes (downloaded files, images, videos, etc) are referred to as the inbound traffic. We observed similar inbound and outbound median values of 2.4 KB. The mean inbound data bytes value is 215 KB, and for outbound the mean is 49 KB. The maximum inbound and outbound value seen is 2.1 GB, which is comparable for both as indicated by the tail of the graph.

Further analysis shows that the inbound traffic is usually greater than the outbound traffic at any given hour. Our previous hourly analysis suggests that busy D2L traffic tends to begin before noon. A one-hour analysis between 12:00 pm to 1:00 pm, with 70,272 connections on the same day, is shown in Figure 5.17b.




(a) One Day (February 1, 2016)




(b) One hour (12:00 pm - 1:00 pm, February 1, 2016)

Figure 5.17: Inbound v/s Outbound Data per Connection on Log scales

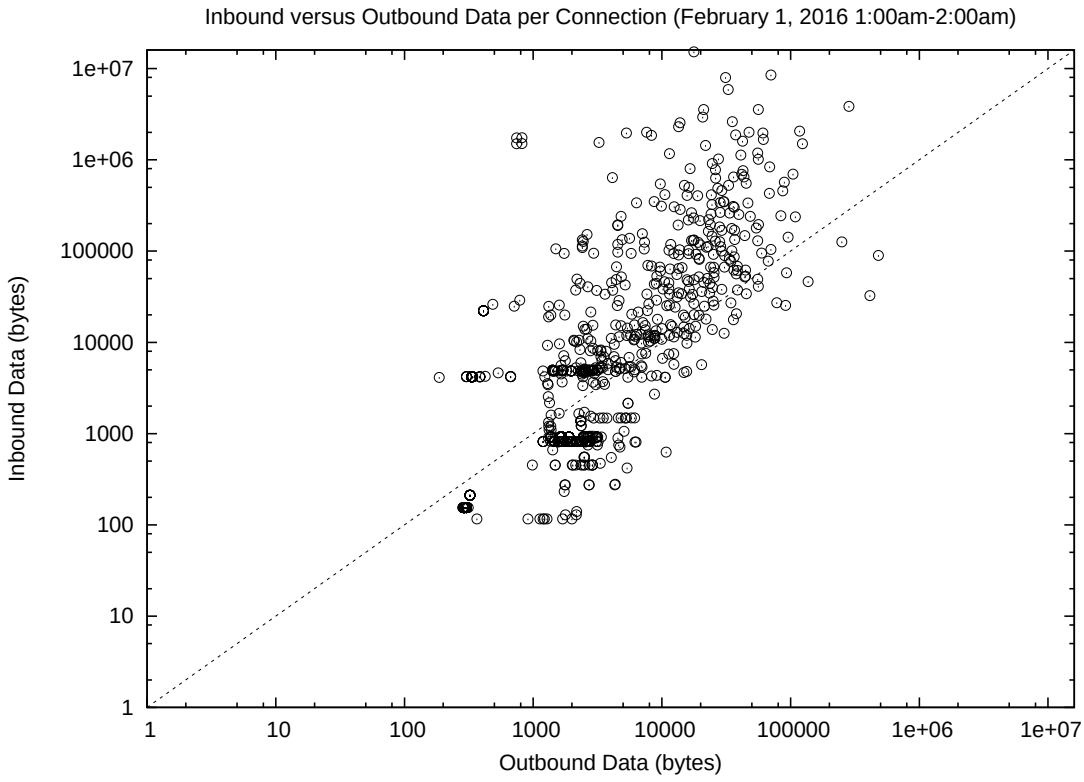



Figure 5.18: Inbound v/s Outbound data per connection on log scales for 1 hour

Here, the largest inbound and outbound data sizes are again comparable, with a value of 2 GB. The inbound and outbound mean data volumes are almost symmetric, with 264 KB and 211 KB, respectively. The inbound median data value is 3.5 KB and the outbound median value is again 2.4 KB. This hour has been selected for analysis to understand the distribution of traffic by focusing on the many connections seen during the day. Selecting this hour helps, since there are an enormous number of connections made with D2L during the day, which helps us understand the graph trends. It helps us look at the inbound and outbound data distribution structure better by further narrowing down our data points.

We show a scatter plot of inbound versus outbound data sizes per connection in log-log scales in Figure 5.18. Each data point is a sum of the TCP header and payload data. We select an hour

during the night time (1:00 am - 2:00 am) when the number of data points are fewer (2,114), and the scatter plot is more understandable. The log scales allow us to see the wide range of data points more clearly. Figure 5.18 gives a good sense of the degree of asymmetry. A y=x line has been added to easily tell which connections are symmetric in data volume and which are not. Almost 60% of the connections are above this identity line, indicating higher inbound data than outbound. However, during busy hours of the day a much higher percentage of inbound traffic is seen.

We also see a few horizontal bands which signify outbound connections of same size. These outbound data sizes comprise mostly of the D2L log-out page components (logos, css files, etc) as reflected in the Bro HTTP logs.

Finally, this analysis suggests that U of C is a net consumer of D2L traffic and not a net producer.

**Throughput Analysis**

The throughput analysis helps us see the data transfer rate, that is, the amount of data that can be carried from one point to another in a given period of time, generally a second. Throughput is one of the factors that affects network application performance. We calculated the Average Data Rate (ADR), which is the size of the transferred file divided by the time duration. We are measuring the ADR since it indicates the average throughput. Figure 5.19 shows the LLCD plot of the ADR in bits per second for one day: February 1, 2016. The average ADR is 500 Kbps, with some data points up to 5 Mbps for inbound connections. A much lower ADR is seen for outbound connections, with the average being 50 Kbps, and a maximum ADR of around 350 Kbps. These low throughput values represent only the average, and not the instantaneous throughput. Specifically, they are calculated from the byte counts and the durations reported in the connection logs, and the duration includes all the TCP connection setup / teardown, and only timeouts used for persistent connections.

## 5.6 HTTP Request Methods and Response Status Codes

This analysis has been performed for one semester (Winter 2016 term). For the HTTP requests, the majority (almost 90%) use the GET method, while we see some HEAD and POST methods in Table 5.7. Table 5.8 shows HTTP status codes seen over the four months duration.

Table 5.7: Top HTTP Methods over 4 months (Winter 2016)

| HTTP METHOD | Count | Pct. |
|---|---|---|
| GET | 1,067,880 | 99.91 % |
| HEAD | 792 | 0.07 % |
| POST | 144 | 0.01 % |
| Others | 1 | 0.00 % |

Table 5.8: Top HTTP Responses over 4 months (Winter 2016)

| Response Code | Count | Pct. |
|---|---|---|
| `302 (Found)` | 1,068,992 | 81.46% |
| `200 (OK)` | 217,783 | 16.60% |
| `204 (No Content)` | 4,309 | 0.33% |
| `503 (Service Unavailable)` | 1,577 | 0.12% |
| `304 (Not Modified)` | 1,354 | 0.1% |
| Others | 5,149 | 1.39% |

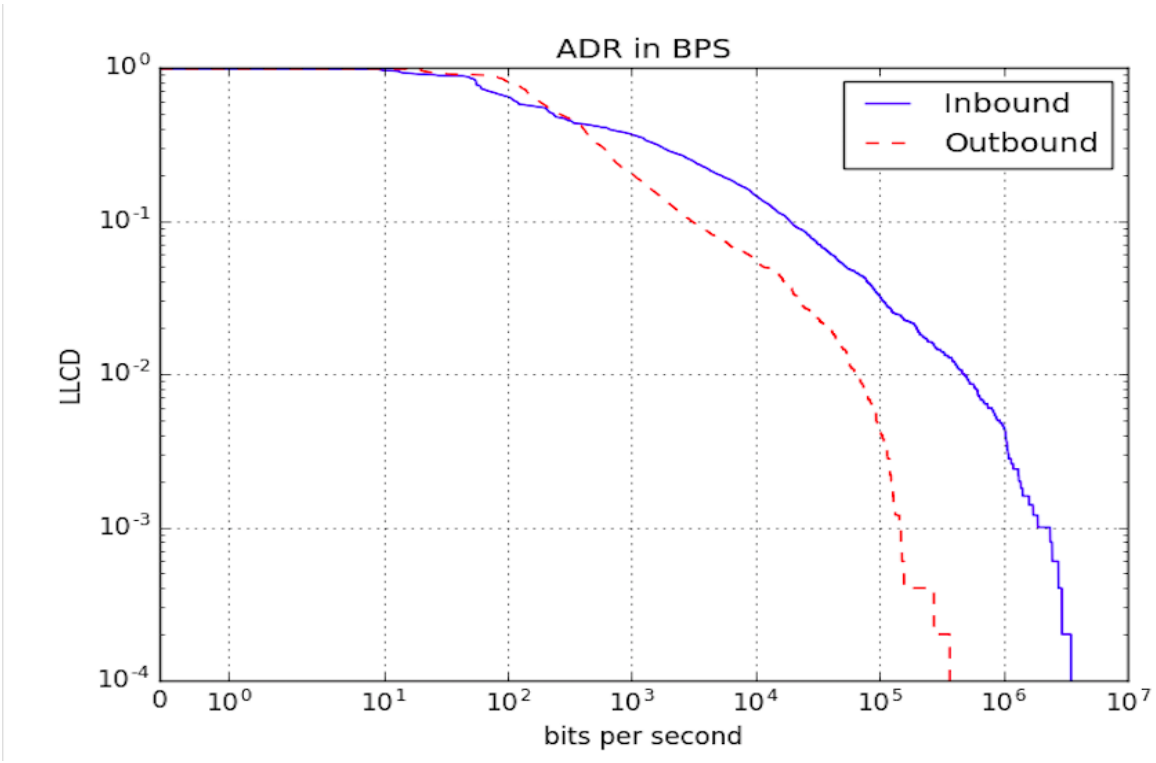



Figure 5.19: ADR (in bits/second) for Inbound versus Outbound Connection

Most of the status codes (over 80%) seen are HTTP `302 Found`. The status codes over the week and weekend periods are similar to our previous HTTP traffic analysis over one semester. Following the HTTP 302 is the HTTP 200 OK, which signifies that a GET, HEAD, POST, or TRACE request has succeeded. Various other types of codes are also seen infrequently. The ones that are seen are 204 (No Content), 503 (Service Unavailable), 304 (Not Modified), 404 (Not Found), 301 (Moved Permanently), 403 (Forbidden), 407 (Proxy Authentication Required), 206 (Partial Content), 414 (Request URI too long), 400 (Bad request), and 303 (See Other). We see a few 304 responses as mentioned in Table 5.8. The 304 response signifies that the server is trying to minimize data transfer when the client already has a cached object [36]. The D2L server tells the client to use that stored version. Thus we would have seen more 304 response codes if there was a CDN node locally placed inside U of C.

## 5.7 URL Popularity Profile

During our investigation of referer URIs, we observed two types of referers: 'pre-D2L referer' step, and the 'post-D2L referer' step. We used the `host`, `uri`, and `referer` fields from our HTTP logs for this analysis. The `host` field helps us view the hosted service Web site, which in this case was the D2L site, `d2l.ucalgary.ca`. The `uri` shows the 'pre-D2L referer URLs', and the `referer` field allows us to view the 'post-D2L referer URLs'.

The 'pre-D2L referer' step shows what people were doing just before starting a new D2L session. Figure 5.20 shows examples of the pre-D2L referer URLs from our HTTP monitor logs. This analysis showed how people re-enter from the logout page, while some others do it via a Bing search engine for U of C D2L. The TITL Web server (`elearn.ucalgary.ca`) and Yahoo search are also seen in our traces.

Figure 5.21a shows the pre-D2L referer URL popularity frequency-rank plot over a period of one semester (Winter 2016).

```
1457680654.281954       CbgHy134fuLtitvJI1      136.159.49.118  60001   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586  0       0
     302      Found   -       -       -       (empty) -       -       -       -       -       -       -
1457680658.528837       CaCBGt2aJnoR6lLW37      136.159.49.117  51727   199.30.181.42   80      1       GET     d2l.ucalgary.ca /       -
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36   0       0       302     Found
 (empty)  -       -       -       -       -       -       -       -
1457680680.053108       CbgHy134fuLtitvJI1      136.159.49.118  60001   199.30.181.42   80      2       GET     d2l.ucalgary.ca /d2l/orgtools/
style/colour.css http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
ko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586        0       0       302     Found   -       -       -       (empty) -       -       -
     -
1457680680.092003       CbgHy134fuLtitvJI1      136.159.49.118  60001   199.30.181.42   80      3       GET     d2l.ucalgary.ca /images/ucalga
ry-logo.png      http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
ko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586        0       0       302     Found   -       -       -       (empty) -       -       -
     -
1457680680.173859       CJgHB73BrA9hyhHPA3      136.159.49.118  60027   199.30.181.42   80      1       GET     d2l.ucalgary.ca /images/body-b
g.png    http://elearn.ucalgary.ca/desire2learn/log-out/ Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
me/46.0.2486.0 Safari/537.36 Edge/13.10586        0       0       302     Found   -       -       -       (empty) -       -       -       -
     -
1457680680.178853       CbgHy134fuLtitvJI1      136.159.49.118  60001   199.30.181.42   80      4       GET     d2l.ucalgary.ca /images/bg.jpg
http://elearn.ucalgary.ca/desire2learn/log-out/  Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.
2486.0 Safari/537.36 Edge/13.10586        0       0       302     Found   -       -       -       (empty) -       -       -       -       -
     -
```

Figure 5.20: Example of Pre-D2L Referers reflected in the HTTP Logs

A total of 1,001 unique URIs were seen in the period of 4 months. The distribution has a power-law structure, but the majority of pre-D2L referer URIs show `http://elearn.ucalgary.ca/desire2learn/log-out/`, the D2L logout URL, which takes

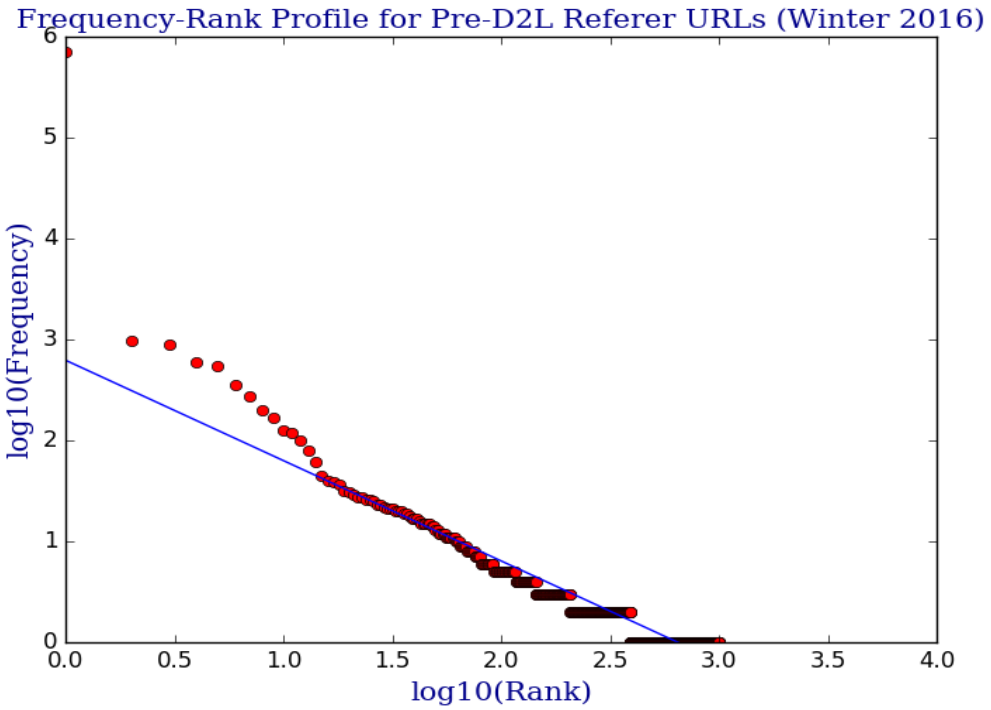
place locally through the TITL Web server. A few other popular pre-D2L referer URI seen are mostly through a Bing search engine page `https://www.bing.com/`, and the U of C IT support Web site `http://www.ucalgary.ca/it/`.

In addition, Wireshark shows the host and referer URIs. Wireshark confirmed our analysis of the 'pre-D2L referer' step and correctly shows that when the host is `d2l.ucalgary.ca`, during the logout process the referrer is `http://elearn.ucalgary.ca/desire2learn/log-out/` as seen above. In addition, Wireshark also shows that this referrer URL actually triggers the `GET` requests for these images, and UCalgary logos. An example of a full request URI for such images looks like `http://d2l.ucalgary.ca/images/ucalgary-logo.png`.

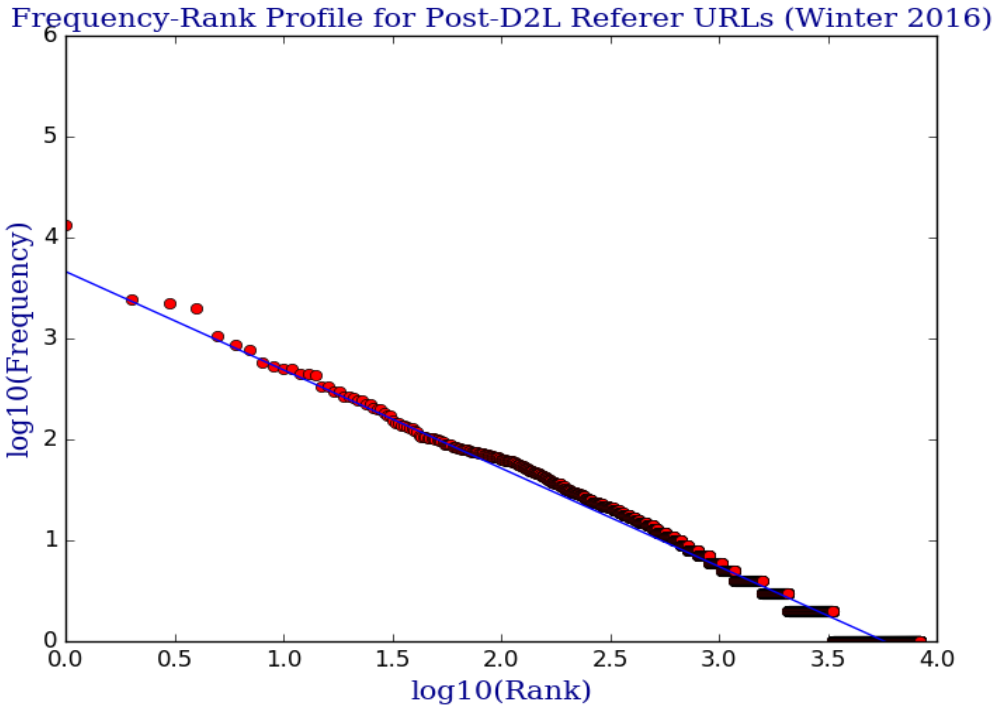
The list of post-D2L referers includes search engines like Bing, Google, Yahoo, QQ mail, several blog sites, online newspaper sites, online shopping sites like Amazon, and various others. The 'post-D2L referer' step shows what people do right after their D2L session, and what page they were on when they left. Figure 5.22 shows some of the post-D2L referers visible in our HTTP logs.

Figure 5.21b shows the post-D2L referrer URL popularity frequency-rank plot over a period of one semester (Winter 2016). A total of 8,418 unique post-D2L referer URIs are seen in the period of 4 months. The graph shows evidence of a Zipf-like (i.e., power-law) popularity distribution, which is often seen in Web traffic [11].

The HTTP pre-D2L referer and post-D2L referer fields provide us with interesting findings on the D2L traffic and shed some light on D2L usage. However, it also tells us that HTTP gives us very limited visibility into D2L activities. A best fit line has been drawn for our URL frequency-rank analysis to see if the graph is linear. The line of best fit in Figure 5.21 helps us identify trends occurring within the dataset. Visually it is evident that it is not strictly linear, but close to it.




(a) Best fit line for Pre-D2L Referer URLs




(b) Best fit line for Post-D2L Referer URLs

Figure 5.21: HTTP Referrer Best Line Fit

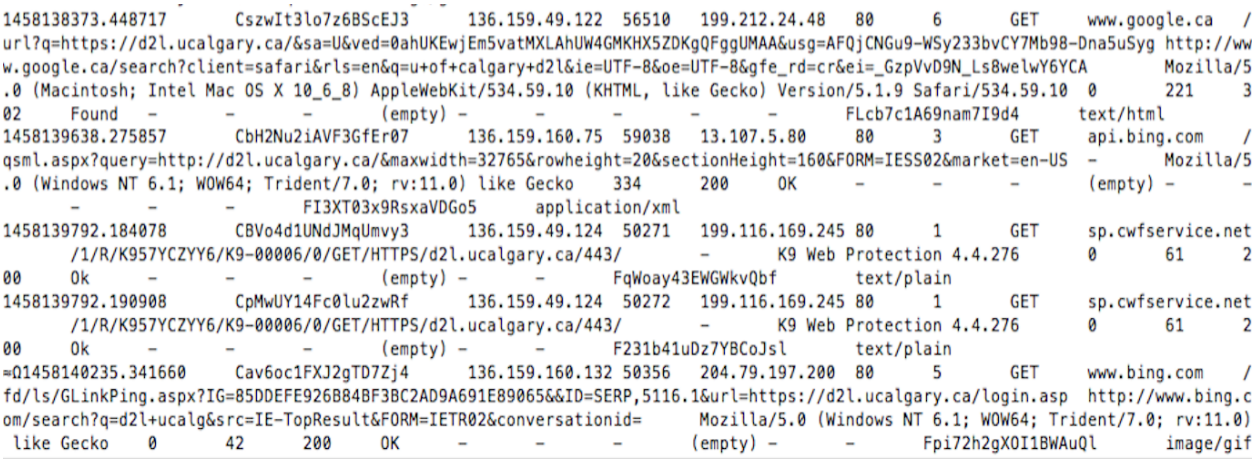



Figure 5.22: Example of Post-D2L Referers reflected in the HTTP Logs

## 5.8 Summary

In this chapter, we sought to understand how D2L is being used by faculty, staff, and students at the U of C. We performed several statistical analysis to understand the typical D2L traffic patterns during the Winter 2016 term. We studied the HTTP and HTTPs IPs and popular subnets accessing D2L, to get a better sense of the role of DHCP server, and NAT from a macroscopic perspective. We discussed the various user agents used, and also characterized the types of browsers and operating systems. We performed session-related analysis, and also discussed throughput. Finally, we concluded this chapter with an in-depth study on the typical URLs for the 'pre-D2L step', and right after logging out as the 'post-D2L step'.

In the next chapter, we further discuss D2L features and provide our recommendations.

# Chapter 6

# Discussion and Recommendations

In this chapter, we analyze TCP dynamics, discuss Web prefetching, and provide recommendations for D2L to improve their site. Finally, we conduct experiments using the `mitmproxy` software to shed light upon encrypted traffic, and provide recommendations for U of C as well.

A recurring theme in this chapter is the adverse impact of network latency on user-perceived performance in D2L. The performance of D2L is affected by the previously seen high RTT values (40 ms). Users spend time waiting for responses from a distant data center (in Toronto). This hampers the responsiveness of the D2L Web site. The bandwidth may not be fully utilized, and the performance of D2L suffers. Moreover, we study if TCP slow start, window size, retransmissions, or packet losses are affecting the TCP throughput for downloads and uploads of large files.

## 6.1 TCP Settings

We have used Wireshark to study the TCP version and settings used by D2L data transfers, in order to understand their performance impacts. Wireshark provides us with various other TCP details like TCP slow start, TCP segment length, maximum segment size, window size, window scaling features, checksum, sequence number, SEQ/ACK analysis, and others.

### 6.1.1 Throughput

In data transmission, network throughput is the amount of data transmitted successfully from one point to another in a given period of time, and is typically measured in bits per second (bps). For our experiments, we measure throughput in Megabits per second (Mbps). To examine the throughput, we performed two Wireshark experiments from an off-campus wireless device. In both experiments, a file with size 3.2 Megabytes (MB) was used.

In the first experiment, a file was downloaded from the available CPSC 501 course materials, taking approximately 31 seconds, and in the second experiment the same file was uploaded, taking almost 67 seconds.

We used the IO feature in Wireshark to generate our results. The `I/O Graph` feature shows us the maximum observed throughput in either direction, from source to destination or vice-versa.

During our first experiment, the highest throughput observed while downloading a file was 14 Mbps. The highest throughput while uploading the same file was 7 Mbps. Similar results were obtained in on-campus experiments (see Table 6.1).

There are two observations here. First, the throughput is low (i.e., much lower than one would expect on a fast network). Second, upload and download are different, almost by a factor of two.

### 6.1.2 TCP Sequence and Acknowledgement Numbers

During the process of data transmission on a TCP connection, the sequence and acknowledgment numbers keep track of the sent data. During a TCP session, each side of the session maintains a 32-bit sequence number, which keeps track of the amount of data it has sent. The sequence number is present inside each transmitted packet, and acknowledged by the opposite host, known as an acknowledgement (ACK) number. The ACK number informs the transmitting host that the sent data was received successfully. A packet is retransmitted if the original packet is lost, or no ACK packet is received in time. A series of three ACK packets for the same sequence number is one indication of a lost packet that requires retransmission.

For this experiment, we downloaded the same file of size 3.2 MB as discussed in Section 6.1.1. To interpret how sequence and ACK numbers work throughout the duration of a TCP session, we can utilize Wireshark's `Flow Graph` utility, which builds a graphical summary of the TCP flow. Each side of the TCP session starts with a value of 0, as the TCP conversation has not yet started. As the packets start to flow, the sequence and ACK number values keep increasing, to track the cumulative bytes transferred. Towards the end of the session, we see a relative sequence number of 3,288,990, which is followed by an ACK number of 3,288,990. This represents the amount of

data that has been transmitted, and received successfully by the end of a full TCP session, in this case a file of 3.2 MB. Also, throughout this particular session, no packet loss was observed, and no packet recovery mechanisms were used.

### 6.1.3 Window Scaling

TCP places certain limits, called windows, on the amount of data that can be in transit between a pair of endpoints at any given period of time. TCP is a reliable protocol, so a TCP sender can send a limited amount of data only, prior to receiving an acknowledgement from a receiver, to make sure that lost segments (if any) can be efficiently retransmitted.

In general, window scaling is used for increasing data transfer efficiency in high-delay and high-bandwidth networks. The TCP window scale option increases the receive window size above its initial maximum value of 65,535 bytes (64 KB). Generally, in most of our Wireshark traces, the Window size value seen for a TCP handshake during a D2L session is 64 KB as indicated by the receiver window. This indicates a very small socket buffer size in use at the user end.

The window scale factor is not set, and no window scaling is used. The window size can be adjusted dynamically by changing the value of the window field in the TCP header, but the scale multiplier remains static for the entire TCP connection. The maximum allowable scale value is 14.

Scaling should be implemented in case of D2L sessions so that more data bytes can be continually transmitted before reaching the window size limit. It provides TCP extensions for high performance networks [37], and improves throughput.

### 6.1.4 TCP Slow start

There are two variables that affect the amount of unacknowledged data that can be sent by a sender. They are the receiver window (RWND), which is advertised by the TCP peer, and the congestion window of the sender (CWND). The details of TCP congestion control algorithms, namely slow start, congestion avoidance, fast retransmit, and fast recovery can be found in [2]. In order to understand the dynamics of a connection-oriented TCP behavior, we use Wireshark's TCP

stream graph. It allows us to visualize the TCP behavior throughout the entire TCP session.

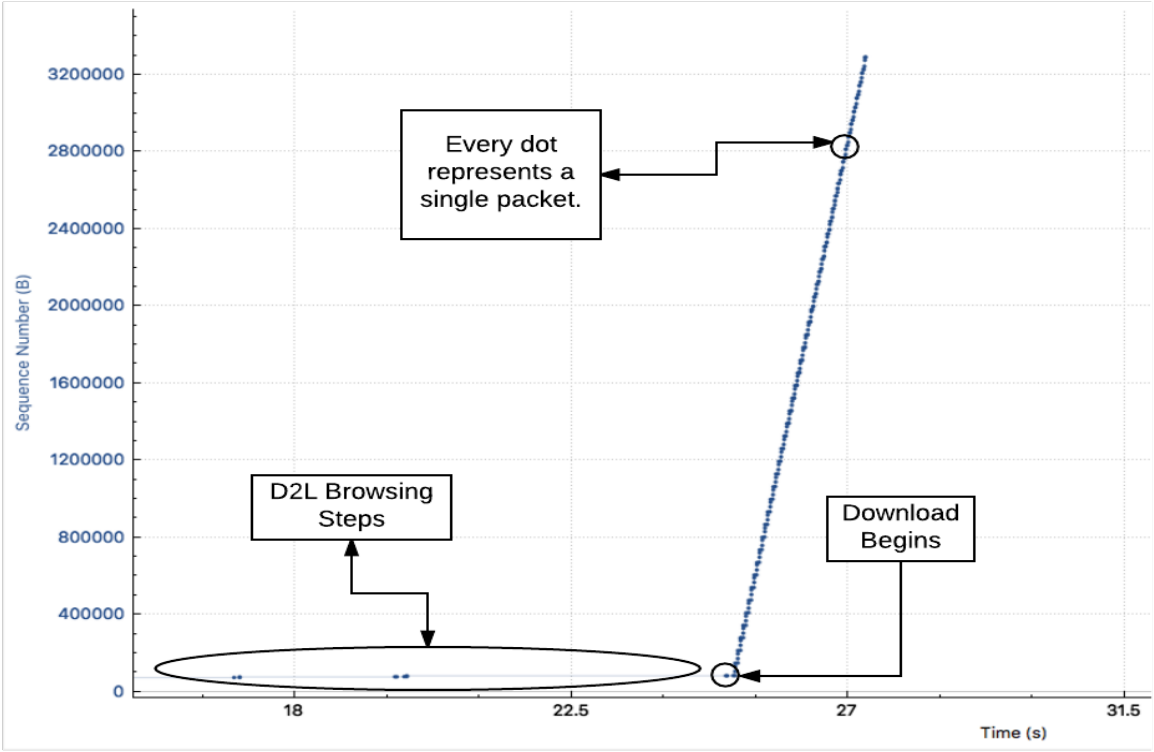
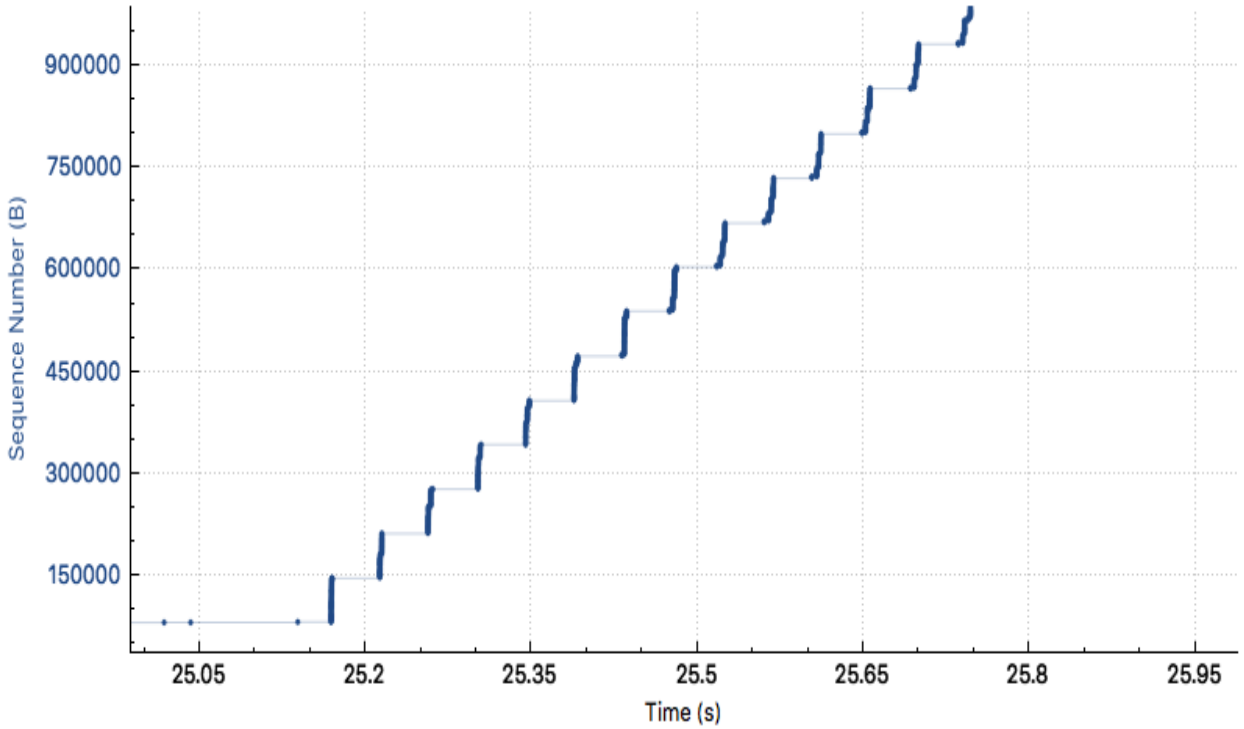
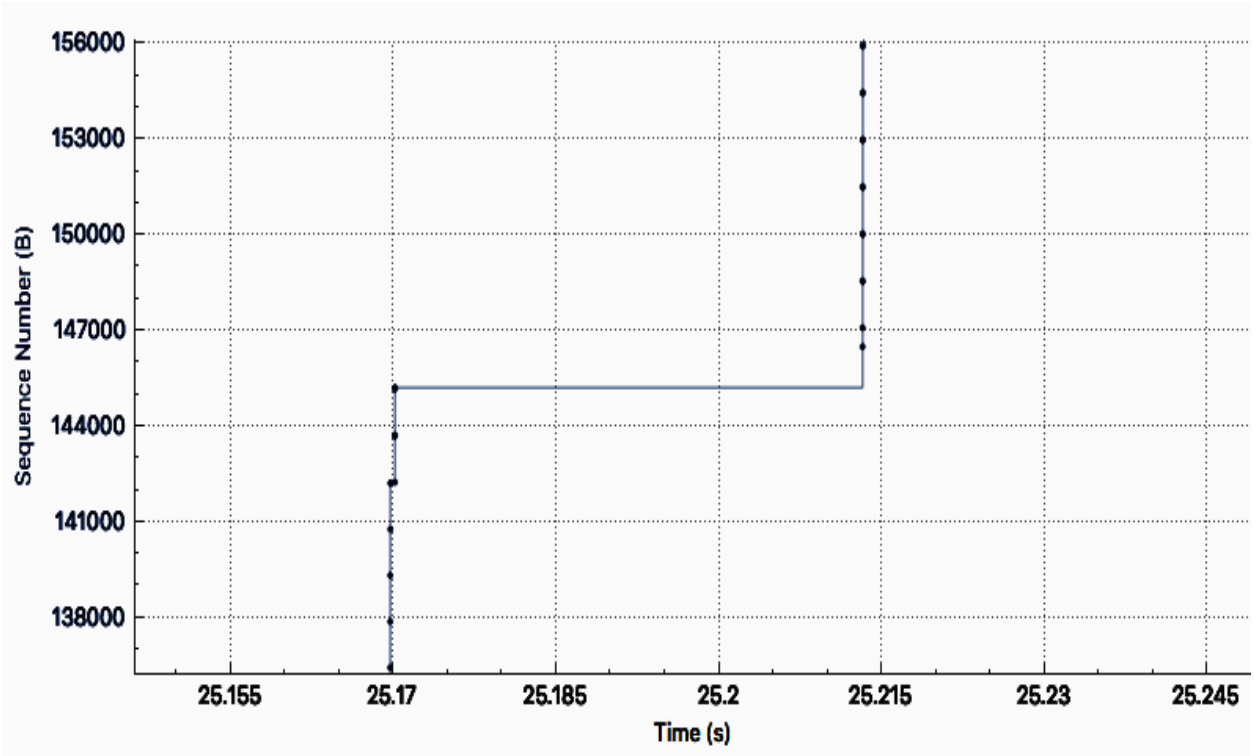



Figure 6.1: TCP throughput during a File Download (Server to Client)

Figure 6.1 shows the throughput generated while an off-campus wireless laptop device downloads a file. It shows the progression of data transfer over time. The horizontal axis represents the time (in seconds) and the vertical axis represents the relative sequence number of each packet. The packets from the D2L server side to the client are shown in Figure 6.1, since a file download is taking place. Every dot in the graph represents a single packet. Initially for the first 24 seconds, a cluster of packets are transmitted roughly every 5 seconds for the first few iterations. These are D2L browsing steps before the download begins. The download happens at 25 seconds and lasts a few seconds, as indicated by the steep vertical slope on the graph. This behavior is consistent for all off-campus wireless downloads performed.




(a) One-second portion of the data transfer




(b) Two data bursts

Figure 6.2: RTT Latency between each data burst

In addition, we zoom in and show a one-second portion of the data transfer. Figure 6.2a shows us the long RTT latency patterns between each data burst of a window of packets. Finally in Figure 6.2b, we zoom in further to calculate the exact RTT values seen.

Figure 6.2b clearly illustrates a RTT latency of 45 ms, between two data bursts. We saw similar high RTT values in our traceroute analysis in Chapter 4.

The `Time-Sequence` graph for the download does not indicate any gaps between sequence numbers, indicating that there is no loss or congestion in the network. The average throughput is 13.5 Mbps (3.2 MB of data in 2.24 seconds).

This experiment shows that D2L data transfers are window-limited. A simple back-of-the-envelope calculation shows that if 64 KB of data are exchanged perfectly every 40 ms, then the average throughput would be about 13 Mbps. This matches closely with the observed performance for downloads.

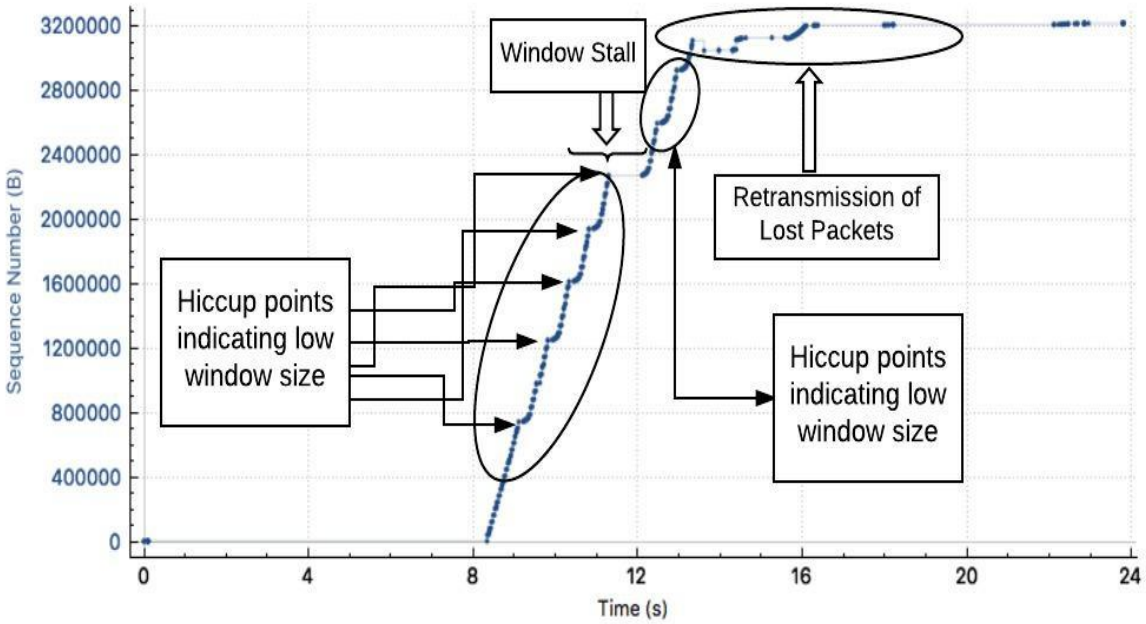



Figure 6.3: TCP throughput during a File Upload (Client to Server)

Next, we take a look at the performance for uploads. The packets from the client to the D2L

server side are shown in Figure 6.3, where the same file of size 3.2 MB is uploaded from an off-campus wireless laptop. The average throughput is 3.4 Mbps, since the data transfer took 8.16 seconds. The behavior discussed below is consistent across all our upload experiments for an off-campus wireless device accessing D2L.

Figure 6.3 reveals an interesting step-ladder pattern while uploading a file. Between 8 and 10 seconds we observe five hiccups, and the Wireshark trace indicates that the sequence numbers are not increasing steadily. These hiccups are due to window size issues. This event is followed by a one second window stall between 11 and 12 seconds. During this one second stall we see that the receiver window size of D2L is almost zero (see Figure 6.4). More hiccups are observed later between 12.5 and 13 seconds. Next we observe four retransmissions of lost packets between 13 and 14 seconds. A series of three duplicate ACKs observed along with these retransmissions signify a packet loss event, and although TCP can recover from this loss, the overall throughput of the connection with D2L drops. The packet losses could occur due to two main reasons. As discussed later in this chapter we speculate that D2L uses an outdated and slow Windows 2008 R2 server. But packets are mainly lost during wireless data transfers, since we did not observe any packet loss while uploading the same file using a wired Ethernet device.

While uploading a file of size 3.2 MB into D2L through a wireless off-campus device we noticed that the advertised receive window size decreases significantly, tends towards zero with a minimum window size of 375, but does not reach zero. At this point the D2L receive window size indicates that it would not be able to receive further information momentarily, and the TCP transmission is halted, until it can process further information in its receive socket buffer. The decrease in the window size is similar to what we would expect from a ‘zero window event’. This low window size value of 375 is observed seven times during the file upload process. Each time it indicates that D2L has fully filled its receive socket buffer.

We initially speculated that the upload performance could be limited by our off-campus Shaw connection speed. But we noticed similar throughput values for the on-campus wireless laptop

setting.

We also generate a graph of the D2L advertised receive window size versus time, so that we have a better understanding of this example transfer. We focus on the window size values between 8 and 16 seconds since the upload process takes place within these timestamps. Figure 6.4 shows how the receiver window size tends to decrease and have very low values (close to zero) on seven occasions. The straight line between 11 and 12 seconds signifies the window stall event. Also we never see a window size exceeding 64 KB, which shows that there is no dynamic resizing of window size during this D2L file transfer process.

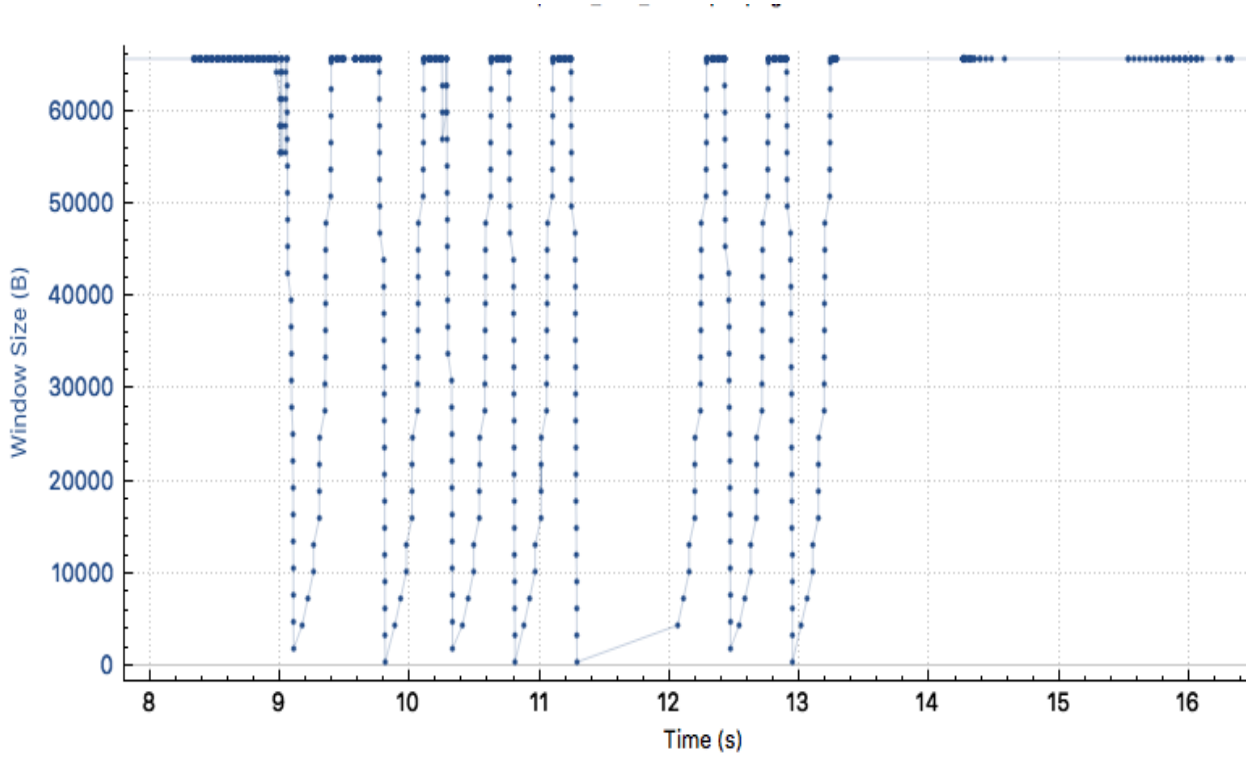



Figure 6.4: Advertised Receive Window Size versus Time

We observed the TCP throughput behavior in several settings, which are shown in Table 6.1. The throughput seen while downloading/uploading a file from a wireless on-campus laptop device is identical to an off-campus wireless download/upload process as shown previously. Surprisingly, when we use a wired on-campus Ethernet desktop device, we see similar throughput results as in case of wireless devices. On a LAN with high bandwidth and extremely low delay, we expected

to see a higher throughput while downloading/uploading. These results indicate that window size limits constrain D2L's throughput performance for U of C users.

Table 6.1: TCP Throughput in various settings

| Setting | OS | Type of Device | Download | Upload |
|---|---|---|---|---|
| On-campus wired | Windows 8 | Desktop | 14 Mbps | 7 Mbps |
| On-campus wireless | Mac OS X | Laptop | 14 Mbps | 7 Mbps |
| Off-campus wireless | Mac OS X | Laptop | 14 Mbps | 7 Mbps |

## 6.2 D2L Observations and Recommendations

D2L should change its socket buffer size to at least 1 MB instead of the default 64 KB receive window size. A higher socket buffer size would improve the data transfer rate. Windows as high as 1 GB are permissible by using a window scaling factor. For this approach, manual tuning is required.

D2L operates in a network environment with large bandwidth and high round-trip times. TCP/IP stack fingerprinting techniques like ettercap [1] revealed the D2L server side OS information. We notice the presence of IIS 7.5 Web server as seen previously in Chapter 4. We speculate that the TCP version used by D2L is Compound TCP (CTCP) [58]. CTCP is a combination of delay-based and loss-based approaches. Enhanced Microsoft IIS 7.5 features are mostly installed in Windows 2008 Web R2 servers (which are outdated). This TCP version permits a maximum window size of 64 KB and generally produces a 50 ms RTT. We have previously seen similar results while digging deeper into our throughput analysis. Using the delay-based component of CTCP the data sending rate is increased rapidly by D2L when the network path is under utilized but retreats gracefully in a busy network. However, the data receiving rate is not necessarily improved as seen by the poor latency values during file uploads to D2L. That is why D2L is especially slow for instructors or TAs who upload content compared to students who mostly download content.

[1] `https://ettercap.github.io/ettercap/`

Although the current version provides reasonable performance for D2L, there is certainly room for improvement. There is no dynamic resizing of the TCP receiver socket buffer size. A TCP version like CUBIC should be implemented to support D2L's high speed network environment. CUBIC provides a cubic function to simplify and enhance window control [33]. We have previously seen high RTT values for D2L. CUBIC provides a window growth mechanism that is independent of RTT, which makes it ideal for both long and short RTT values. This TCP version could improve the instructor-perceived performance of D2L while uploading large files in the form of course content.

There are various other TCP versions that are available and should be considered by D2L. Dynamic resizing of send and receive socket buffer sizes based on the observed characteristics of the network portion could be vital. In high or low packet loss environments Selective Acknowledgment (SACK) could be advantageous.

The D2L configuration should be improved for its geographically dispersed clients. U of C is one of D2L's western-most clients. Thus content caching mechanisms should be used abundantly. Campus level caching (local proxy server), and using a local CDN node would be ideal for improving content delivery. Other latency hiding techniques like prefetching could be used.

**Prefetching**

Since U of C students have access to a few specific courses they are enrolled in, we anticipate that D2L could deploy Web prefetching to obtain course content even before a user requests it, based on the user's pattern of requests. Prefetching helps in optimizing the D2L performance since the content that might be accessed by a user would be downloaded in advance, and thus reduces the latency, and makes D2L a more responsive Web site. In the background, the user's browser caches this content, making it instantly available if the user clicks on a link for that content.

The most important advantages of this prefetching approach are:

1. Improved content loading times for frequently accessed files.

2. If a course has many students, we should get a good idea of what course content

is being accessed and when. If there was better caching at the U of C, then many RTTs could be saved for students. This might be a scenario where prefetching (or presence of local CDN nodes) could actually work well.

3. The browsing experience for D2L makes the users feel that a Web site is locally hosted rather than in Toronto. This makes the Web site more responsive and results in improved user experience.

**CDN Nodes**

The RTT between a client and the D2L server is slow both for on-campus and off-campus users. As discussed in Chapter 4, the total latency observed for off-campus users varies a lot, and is often more than double that of on-campus users. The traceroute experiments revealed that the off-campus traffic has packets traveling through Vancouver before finally reaching the D2L hosted service in Toronto.

These multiple hop counts for both on-campus and off-campus D2L users result in large latencies, which could be greatly reduced by deploying a Content Delivery Node locally at the U of C. Browser-level caching partially solves the problem.

We used a bulk whois resolver[2] to resolve all the IP addresses seen in our Wireshark trace files. We find that D2L uses CDN services like Fastly (CIDR 23.235.33.0/24) and Cloudflare (104.16.16.0/20).

D2L even uses Amazon Web Services (AWS) like Amazon Elastic Cloud Compute i.e., EC2 (52.24.0.0/14) and Amazon Cloudfront (52.84.16.0/21), which is a CDN offered by AWS. These CDN services used by D2L accelerates static cached content delivery and improves page load times. We geolocated these CDN IPs and most of the CDN nodes are located in the United States of America (Texas, California, etc).

While downloading an object within a session, several round trips are required between D2L, the TITL Web server, and user to fetch a single D2L object. During login and logout, a few round

[2] `www.team-cymru.org`

trips become evident through the 302 redirects observed during our D2L session analysis. We feel that the latency would be greatly reduced through the use of local CDN nodes. Also, the number of redirects should be reduced by slightly modifying their back-end APIs, which could help in fetching the content directly from the destination server. This could in turn accelerate the content delivery.

## 6.3 Recommendations for U of C

### 6.3.1 Man in the Middle (MITM) proxy

Since most D2L traffic was encrypted, we could not monitor the internal activities of D2L sessions in either our monitor logs or our Wireshark traces. For example, the files that we were browsing in D2L are not evident.

To obtain insight into this encrypted traffic, we used software named `mitmproxy` [43]. The software intercepts the traffic between a client and a server, and can silently listen to all the HTTP or HTTPS traffic requests made by the user. All browsing data could be sent to `mitmproxy` from our desktop, and `mitmproxy` would in turn forward these requests to the Internet. The Internet would then send back information, which passes through `mitmproxy` before finally reaching the end user. This is how `mitmproxy` acts as a man-in-the-middle (mitm) proxy. The most interesting feature of `mitmproxy` is its ability to decrypt HTTPS traffic and make these packets visible in a human-readable form. The HTTPS traffic is decrypted by `mitmproxy` by installing a certificate on our devices such that the information can be sent to `mitmproxy` in plain-text.

For the purpose of this experiment, `mitmproxy` was installed in a Macbook Air laptop and the Mozilla Firefox browser was configured. Since the sniffed machine and the `mitmproxy` machine are the same device for this experiment, we installed the certificates in the same machine from a Web site named `mitm.it`. We also need to make sure that `mitmproxy` is running in the background terminal. We can install these certificates in any machine by simply sending the certificate to that

device and by making sure that the target machine trusts the certificate.

For this experiment, we updated the `Manual Proxy Configuration` in Mozilla, and set the HTTP and HTTPS proxy IPs to 127.0.0.1 (loop back), and the port numbers to 8080 for both the cases. These settings can be easily altered based on our requirements. For example, if we wish to monitor traffic of a different device, or a desktop device, or even a mobile device (iPhone or Android device), the setting would be slightly different. Further details are available in [43].

```
GET http://d2l.ucalgary.ca/
    ← 302 [no content] 186ms
GET https://d2l.ucalgary.ca/
    ← 302 text/html 127b 108ms
GET https://d2l.ucalgary.ca/d2l/login
    ← 302 text/html 132b 115ms
GET https://d2l.ucalgary.ca/d2l/custom/cas
    ← 302 text/html 243b 94ms
GET https://cas.ucalgary.ca/cas/login?service=https%3a%2f%2fd2l.ucalgary.ca%2fd2l%2fcustom%2fcas&ca.ucalgary.authent.ucid=true
    ← 200 text/html 1.89k 101ms
GET https://cas.ucalgary.ca/cas/css/screen.css
    ← 200 text/css 8.84k 289ms
GET https://cas.ucalgary.ca/cas/images/tile.png
    ← 200 image/png 18.66k 476ms
GET https://cas.ucalgary.ca/cas/js/lib/jquery-1.10.2.min.js
    ← 200 text/javascript 90.92k 486ms
GET https://cas.ucalgary.ca/cas/js/lib/respond.min.js
    ← 200 text/javascript 3.97k 435ms
GET https://cas.ucalgary.ca/cas/js/lib/jquery.placeholder.js
    ← 200 text/javascript 4.96k 386ms
GET https://cas.ucalgary.ca/cas/js/script.js
    ← 200 text/javascript 114b 489ms
GET https://cas.ucalgary.ca/cas/fonts/ProximaNova-Bold-webfont.woff
    ← 200 text/plain 21.71k 467ms
GET https://cas.ucalgary.ca/cas/images/crest.png
    ← 200 image/png 19.09k 315ms
GET https://cas.ucalgary.ca/cas/images/background.jpg
    ← 200 image/jpeg 34.56k 318ms
GET https://cas.ucalgary.ca/favicon.ico
    ← 200 image/vnd.microsoft.icon 1.12k 94ms
```

Figure 6.5: Before logging into D2L

First, we simply enter `d2l.ucalgary.ca` into our browser and see the redirects before logging in. See Figure 6.5 for the exact files that are requested by D2L, and how the D2L traffic is redirected to the CAS server before logging in. Furthermore, through `mitmproxy` we can see the file name, file size, the HTTP status codes, and the full URL as well.

Next, after logging into D2L, we notice that the user name and passwords are not shown during the process. Nonetheless, the HTTPS traffic is immediately visible, along with the exact course name, and file type (pdf, docx, mp3, mp4, etc), including the file size.

We accessed the `CPSC 501` course and viewed a file of size 210.57 KB. The first line shows

the course term as F2016, indicating a Fall 2016 course offering, L01 is the lecture number, and Week%201.pdf indicates the file name. We notice '%20' because there was a blank space character in the original file name (Week 1.pdf). The file size is 210.57K, the file type is application/pdf, and the time taken to load the content is shown as 482 ms. Then inside D2L, we select a drop down menu to download the file by clicking the Download option. The list of available options from the menu are also shown in the URLs that follow, for example creating a new file, editing, deleting, and leaving feedback. The last few lines in Figure 6.6 show that the file was viewed just before it was downloaded, as evident from the URLs. The same file of 210.57K took 247 ms to finish downloading. The download process always has higher throughput as we have seen earlier. This process is faster also because unlike the file upload process (which we discuss next), D2L does not update a file directory structure internally.

```
GET https://d2l.ucalgary.ca/content/enforced/156449-F2016CPSC501L01/Week%201.pdf?d2lSessionVal=lzv7bLYl7i5p2ofNn2hCOC6dw&ou=156449
    ← 200 application/pdf 210.57k 482ms
POST https://col.eum-appdynamics.com/eumcollector/beacons/browser/v1/AD-AAB-AAE-ANG/adrum
     ← 200 text/html [no content] 219ms
GET https://d2l.ucalgary.ca/d2l/img/lp/contextMenu/bg.gif
    ← 200 image/gif 43b 320ms
GET https://d2l.ucalgary.ca/d2l/img/0/Content.Main.infNewFile.gif?v=10.7.2.7112-98
    ← 200 image/gif 220b 487ms
GET https://d2l.ucalgary.ca/d2l/img/0/Shared.Main.actEdit.png?v=10.7.2.7112-98
    ← 200 image/png 302b 421ms
GET https://d2l.ucalgary.ca/d2l/img/0/Content.Main.actMetadata.gif?v=10.7.2.7112-98
    ← 200 image/gif 357b 410ms
GET https://d2l.ucalgary.ca/d2l/img/0/Content.Main.actDelete.png?v=10.7.2.7112-98
    ← 200 image/png 1.03k 572ms
GET https://d2l.ucalgary.ca/d2l/img/0/Content.Main.actLeaveFeedback.gif?v=10.7.2.7112-98
    ← 200 image/gif 362b 469ms
GET https://d2l.ucalgary.ca/d2l/le/content/156449/topics/files/download/2185587/CheckFileTopicInfo?isXhr=true&requestId=2
    ← 200 application/json 384b 139ms
POST https://d2l.ucalgary.ca/d2l/le/content/156449/viewContent/2185587/f76162e0-9d28-4845-85d8-ef2a042c243d/FinalizeView?time=21516
     ← 200 application/json 329b 140ms
GET https://d2l.ucalgary.ca/d2l/le/content/156449/topics/files/download/2185587/DirectFileTopicDownload
    ← 200 application/pdf 210.57k 247ms
```

Figure 6.6: Browsing and Downloading a file from D2L

Our final example shows the same file being uploaded. Unlike the download session, the file size and duration are distributed through a series of URLs. However, the series of POST requests clearly indicate a file being uploaded to the server. See Figure 6.7 for an illustration of the file upload process. It took longer to upload the same file than downloading (612 ms). D2L internally

updates a file directory structure as indicated by `UpdateTreeBrowser` in one of its URLs. The final line indicates the pop up in the activity feed right after new content is uploaded into D2L, as a notification for its users.

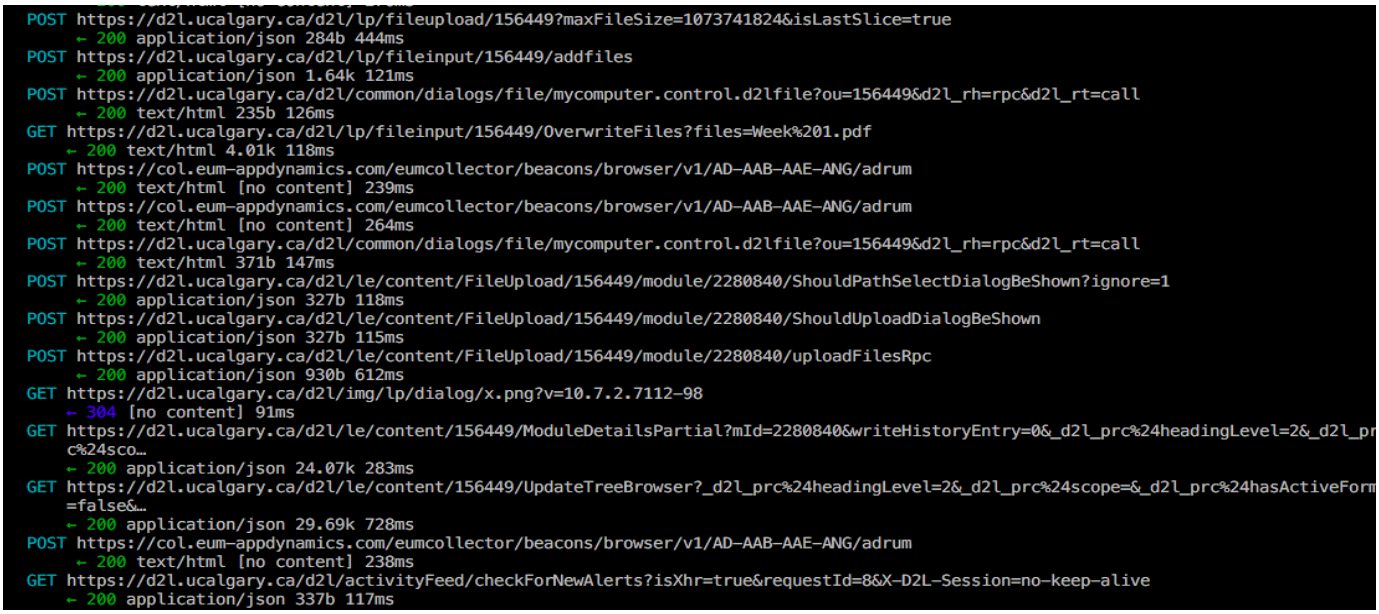



Figure 6.7: Uploading a file into D2L

This `mitmproxy` software could be used by the U of C as a way of checking encrypted traffic for suspicious users by simply installing a certificate and setting up the proxy IP of the target machine as the `mitmproxy` device IP. It could be used for ethical hacking, making campus devices more secure, and getting prepared for malicious cyber attacks. This could be used as an Intrusion Detection System by U of C IT staff and Internet Service Providers (ISP). It could also be deployed on personal computers to perform deeper analysis on the exact URLs, which shed light upon user activities in encrypted or unencrypted Web sites. Also, because of the significant amount of information at a researcher's disposal, more measurement studies can be performed since the type of file, file size, activity durations, and others are clearly visible.

On the other hand, D2L should be aware of this software and should provide instructions to never accept `mitmproxy` certificates or any unknown certificate since it could encourage man-in-the-middle attacks.

### 6.3.2 Other suggestions

Previously we have seen a receiver window size of 64 KB which is fine for LAN networks at the U of C. Improvement for content delivery at the U of C requires a larger socket buffer size, which can be achieved through manual tuning (using a scaling factor) or dynamically by using an improved TCP version.

In Chapter 4, we discussed how D2L uses cookies. However, unencrypted HTTP cookies could be sent even if D2L uses only HTTPS. However, `mitmproxy` also provides us with `Request`, `Response`, and `Detail` fields. Under the `Request` field, we observed that secure cookies are used. This should be constantly monitored by U of C IT services to make sure that non-encrypted cookies are never allowed by blocking them, which could potentially prohibit man-in-the-middle attacks.

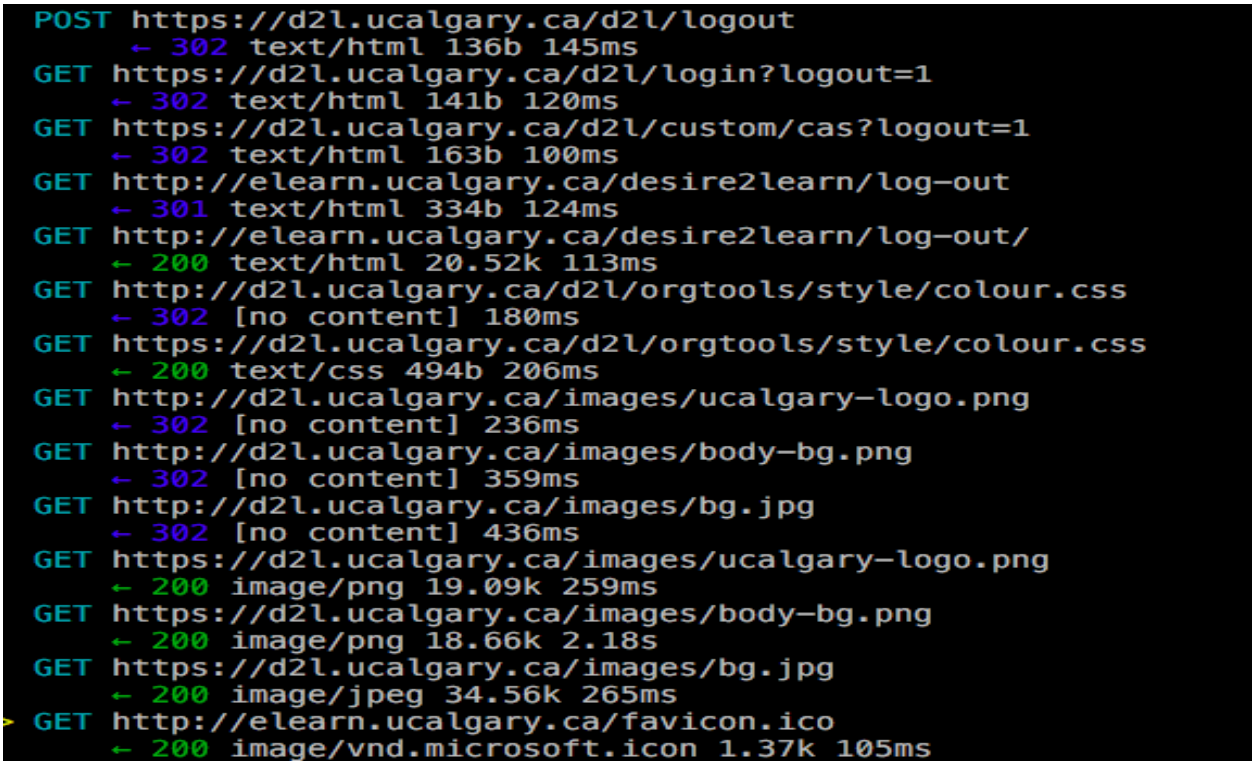



Figure 6.8: Logout from D2L

Also, users can re-enter through the D2L logout page (CAS page) directly, even after logging out. The CAS login page does not even show up if the browser saves passwords.

Finally, we detected multiple redundant redirects during our session analysis. The most notable

ones are first, during the login process in Figure 6.5 where there are four `302` redirects instead of just one to reach the CAS login page. Second, during the logout step in Figure 6.8, we see multiple unnecessary redirects, the most glaring example is the `301` redirect with URL `http://elearn.ucalgary.ca/desire2learn/log-out` before finally reaching the actual logout page at `http://elearn.ucalgary.ca/desire2learn/log-out/`.

These redirects should be reduced, especially for logging into the D2L homepage and while logging out. These redirects affect the D2L browsing experience for the following reasons:

1. Creates unnecessary load on the D2L servers.
2. Triggers one additional HTTP request-response cycle.
3. Adds RTT latency.

## 6.4 Summary

In this chapter, we studied the TCP dynamics of a D2L session when a file is downloaded or uploaded. We measured throughput, TCP Window scaling options, observed TCP slow start, and also examined the existence of congestion control mechanisms. We provided suggestions for D2L regarding CDNs and prefetching. We provided suggestions for the U of C, by performing experiments using the `mitmproxy`, and pointed out how redirects affect network performance.

In the following chapter, we provide conclusions from our measurement study.

# Chapter 7

# Conclusions

This chapter highlights our observations made from the D2L workload characterization study. We start with a thesis summary in Section 7.1. We revisit our research questions in Section 7.2. We provide conclusions in Section 7.3. Finally, Section 7.4 outlines future work.

## 7.1 Thesis Summary

Overall, this thesis presents a thorough analysis of the D2L traffic at the University of Calgary. Our observation period for monitor logs was from Winter 2015 to the Fall 2016 term. We also performed Wireshark and `mitmproxy` experiments. During this study, we investigated how D2L was used on campus. Our measurements were performed from microscopic and macroscopic perspectives, and several active and passive measurement tools were used.

The chapter-by-chapter structure of the thesis was as follows. Chapter 1 introduced the thesis and presented our goals. Chapter 2 introduced background knowledge on computer networking. We discussed Learning Management Systems, and how Content Delivery Networks work. We discussed the Internet, and the application-layer protocols like HTTP and HTTPS. Finally, we provided related work in the area of network traffic measurement. Chapter 3 described how network traffic data was collected. We discussed the Bro logging framework, passive and active measurement tools, data processing, and ethical considerations. Chapter 4 analyzed the D2L traffic from a microscopic point of view, while Chapter 5 illustrated the D2L traffic from a macroscopic perspective. Chapter 6 provided observations and recommendations for D2L and U of C.

## 7.2 Research Questions Revisited

In Chapter 1, we highlighted our main research questions for this thesis. We provided a detailed discussion of our first two research questions (i.e., the technologies D2L uses in the form of various APIs, and how the D2L traffic is distributed) throughout Chapters 4 and 5.

In Chapter 4, we showed the available D2L roles to select from, the back-end API structure used by D2L, the D2L Content Delivery Network, the typical Internet path for D2L users (both for on-campus and off-campus users, as well as for wireless and Ethernet networks). We also discussed the role of NAT, the role of the CAS server, and provided a thorough analysis of the idiosyncrasies involved in D2L session structures, and how D2L uses parallel and persistent connections throughout a session. We discussed D2L from a microscopic point of view in this chapter, to answer our first research question.

In Chapter 5, we provided network usage statistics of HTTP and HTTPS traffic for D2L over a period of two years. We provided details on the client side operations, the user-agent information, and the types of browsers and OS seen for the Winter 2016 semester. The D2L server could use this information to analyze the platforms their users are comfortable with and stop development for old and outdated browsers and OS. We analyzed the IP frequency, and showed the top HTTP and HTTPS IPs requesting D2L. We studied the typical session durations, and the inbound and outbound D2L traffic volume. We also provided an average throughput analysis, and discussed the ADR for inbound and outbound traffic. We also looked at HTTP requests and responses, and studied the pre-D2L referer URLs and post-D2L referer URLs. Chapter 5 answers our second research question about how D2L is being used.

Our third research question concentrates on how to improve D2L performance at U of C. We perform further analysis to understand the TCP dynamics in Chapter 6. We comment on prefetching features that might be used by D2L, and provide various other suggestions for D2L. We conduct additional experiments using `mitmproxy` software to understand encrypted D2L traffic and provide recommendations for U of C to monitor HTTPS traffic. We discuss the prospects of reducing un-

necessary redirects during the login and logout steps for D2L. This chapter answers our third and final research question.

## 7.3 Conclusions

This thesis provides a valuable workload characterization study of the D2L Web site for U of C users. Most of our analysis has been performed on a four-month period, while a few studies are conducted on the full academic years of 2015 and 2016. We believe that sharing this research work with D2L might help improve the poor RTT latency observed by U of C users. Our work highlights the importance of using an appropriate TCP version and we strongly feel that it could improve the D2L performance for all of its users. The results from this research would greatly benefit U of C IT support, TTIL staff, and other relevant on-campus technical staff members. For example, peak network traffic times could be monitored, encrypted traffic could be monitored, network outages could be reduced, and local caching servers could be deployed to improve content delivery times along with page load times. Some supporting experiments were performed in 2017 using Wireshark and `mitmproxy`. We analyzed both HTTP and HTTPS traffic thoroughly.

Our conclusions are presented as follows:

1. Studying D2L is challenging. D2L traffic is encrypted using HTTPS, so there is limited visibility into the content being accessed. Furthermore, the use of NAT and DHCP complicate traffic analysis, and obfuscate user dynamics.

2. D2L is complex. D2L communicates with two intermediate servers (TITL Web server and CAS server) during its user sessions. The D2L entry/exit points are separate from the CAS entry/exit points. It uses parallel and persistent connections throughout its sessions with users.

3. D2L is heavily used. D2L traffic is generated by both on-campus and off-campus users. High inbound traffic is seen in comparison to outbound traffic. The majority

of traffic generated by D2L is over HTTPS, which is evident from our hourly, daily, weekly, monthly, and yearly traffic analysis of D2L.

4. D2L is slow. At the minimum, there is a 40 ms round trip latency for U of C users to access the D2L site. There is no CDN node present inside the U of C campus. Placing a CDN node locally on-campus could greatly improve content delivery. In addition, we observe over one million HTTP `302` redirects over the Winter 2016 term, which should be reduced to lower the load on the U of C servers. The TCP window size is low and does not scale dynamically in case of file uploads or downloads. Thus it limits the full utilization of available bandwidth.

## 7.4 Future Work

Although much analysis was done throughout this thesis, it could still be extended further in several directions. There are adaptive learning mechanisms adopted by D2L to evolve their LMS system into an Integrated Learning System [73], which they refer to as the world's first 'Integrated Learning Platform' [23]. More work understanding these adaptive features, and how they have influenced D2L users could certainly be a topic for future study.

More studies should be performed in understand the off-campus D2L traffic. This would allow us to calculate the relative data volume and perform comparative analysis for on-campus and off-campus D2L usage.

Throughout our analysis we observe heavy-tailed characteristics. The tail of the distribution could be studied further using the tail-estimator approach using the `aest` tool as discussed in [19]. The estimator is available as `C` source code.

Further studies could be performed in understanding the TCP dynamics. User experience analysis could be accomplished by understanding home versus on-campus D2L experience, wireless versus wired D2L connection experience, and how it affects D2L performance as well. More research could be performed to discover if fast retransmit and fast recovery mechanisms are taking

place after the TCP slow start. A more detailed study could be conducted in understanding packet level details versus the Bro summary.

Behavioral analysis could be performed in understanding the post-D2L referer URLs, and the typical sites visited by users right after accessing a LMS. If the course names of the corresponding course IDs are provided, it could reveal the most popular courses at the U of C. A content popularity analysis could be more reasonable in that scenario.

Understanding the file structure and types of file accessed become possible through `mitmproxy`. Extensive analysis could be performed to understand the requested media, distribution of media, downloaded media, and content popularity.

Machine Learning or Deep Learning techniques like K-Means clustering could be used to find the clustering IPs and identify usage patterns. Topic modeling techniques like Latent Dirichlet allocation (LDA) would allow us to analyze the course popularity and model user behavior seen while operating D2L.

# Appendix A

# Analysis Scripts

This appendix contains several of the data analysis scripts used for generating the results reported in this thesis. Several of these scripts are bash shell scripts and several are Python scripts.

## A.1 HTTP Log Analysis Scripts

Listing A.1: Script to extract data from HTTP Logs

```
import subprocess
import os
from sharedMethods import *
import glob
# WINTER 2016 (4 months)

class Statistic:
        rootDir = "/data4/"

        def __init__(self):
         os.chdir(self.rootDir)

        def httpInfo(self):
         """This function extracts all the records containing "d2l.
            ucalgary" in everyday's http logs.
            Field seperator is "\t", record seperator is "\n".
                """
```

```
            currentDir = subprocess.check_output("ls", shell = True)
             currentDir = currentDir.rsplit()
             for folder in currentDir:
              if os.path.isdir(folder) and folder[:2] == "20" and
                 folder[:10] >= "2016-01-01" and folder[:10] <= "
                 2016-04-30":
                 if not os.path.exists("/home/sourish/m/httpRequests/
                    " + folder):
                  os.makedirs("/home/sourish/m/httpRequests/" +
                     folder)
             for files in glob.glob(folder + "/http.*.log.gz"):
                 httpRequestsPerHour = subprocess.check_output("zcat␣
                    ./" + files + "␣|␣gawk␣'/d2l\.ucalgary\.ca/␣{
                    print␣$0}'", shell = True)
      writeToFile(httpRequestsPerHour, "/home/sourish/m/httpRequests/
         " + files[0:33] + ".txt", "wb+")
     subprocess.call("gzip␣-q␣/home/sourish/m/httpRequests/" + files
        [0:33] + ".txt", shell = True)
stat = Statistic()
stat.httpInfo()
```

Listing A.2: Script to analyze extracted data from HTTP Logs

```
import os
import subprocess
import glob
from sharedMethods import *
import datetime
```

```
class AnalyzeData:
        rootDir = "/home/sourish/m/"

        def __init__(self):
                os.chdir(self.rootDir)
                if not os.path.exists("./httpInfo_1year"):
                        os.makedirs("./httpInfo_1year")
                if not os.path.exists("./ipInfoWinter2016"):
                        os.makedirs("./ipInfoWinter2016")
                if not os.path.exists("./fileInfo"):
                        os.makedirs("./fileInfo")
                self.httpRequestTotalPerDay()
                self.ipInfo()
                self.httpMethod()
                self.httpStatusCode()
                self.checkUserAgent()
                self.ipBookPerDay()

        def __getDateList(self):
                """This function returns a list of strings, of all
                   the dates.
                """
                start = datetime.date(2016, 01, 01)
                end = datetime.date(2016, 04, 30)
                daysRange = (end - start).days
                return [str(start + datetime.timedelta(days = i))
                   for i in range(daysRange + 1)]
```

```
def httpRequestTotalPerDay(self):
        """This function returns a dictionary containing the
            total number of requests per hour per day.
        E.g. {'2016-12-01': {'00...-01...': '1000\n',
           '01...-02...': '23423\n'} ...}
        Also the dictionary is written to ISM/httpInfo/
           httpRequestTotalPerDay.json file.
        """
        httpRequestTotalPerDay = {}
        dates = self.__getDateList()
        for date in dates:
                httpRequestTotalPerDay[date] = {}
                for files in glob.glob("./httpRequests1year/
                   " + date + "/http.*.txt.gz"):
                        httpRequestTotalPerDay[date][files.
                           split("/")[-1].split(".")[1]] =
                           subprocess.check_output("zcat " +
                            files + " | gawk 'BEGIN {i=0; RS
                           =\"\\\\\\\\\n\"; FS=\"\\\\\\\\\t\"}
                           $5~/199.30.181.42/ {i++} END {
                           print i}'", shell = True)
        writeToFile(httpRequestTotalPerDay, "./
           httpInfo_1year/httpRequestTotalPerDay.json", "wb+
           ")



    def ipInfo(self):
```

```
        ipBook = {}
        dates = self.__getDateList()
        for date in dates:
                ipBookPerDay = subprocess.check_output("zcat
                   ␣./httpRequests/" + date + "/http.*.gz" +
                    "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\\n\";␣FS
                   =\"\\\\\\\\\t\"}␣$5~/199.30.181.42/␣{print
                   ␣$3}'", shell = True).rsplit('\n')
                for i in ipBookPerDay:
                        if i not in ipBook:
                                ipBook[i] = 1
                        else:
                                ipBook[i] += 1
        writeToFile(ipBook, "./ipInfoWinter2016/IPWinter2016
           .json", "wb+")

def ipBookPerDay(self):
        """This function uses the data in httpRequests,
           output a dictionary to the ipInfo/ipBookPerDay
           folder.
        """
        if not os.path.exists("./ipInfo/ipBookPerDay/"):
                os.makedirs("./ipInfo/ipBookPerDay/")
        dates = self.__getDateList()
        for date in dates:
                if os.path.exists("./httpRequests/" + date):
                        ipBookPerDay = {}
                        ipDataPerDay = subprocess.
```

```
                                check_output("zcat␣./httpRequests
                                /" + date + "/http.*.gz" + "␣|␣
                                gawk␣'BEGIN␣{RS=\"\\\\\\\\\n\";␣FS
                                =\"\\\\\\\\\t\"}␣$5
                                ~/199.30.181.42/␣{print␣$3}'",
                                shell = True).rsplit('\n')
                            for i in ipDataPerDay:
                                    if i not in ipBookPerDay:
                                            ipBookPerDay[i] = 1
                                    else:
                                            ipBookPerDay[i] += 1
                    writeToFile(ipBookPerDay, "./ipInfo/
                       ipBookPerDay/" + date + ".json", "wb+")


def httpMethod(self):
 httpMethod = {}
    dates = self.__getDateList()
    for date in dates:
     httpMethod[date] = {}
        if os.path.exists("./httpRequests_1week/" + date):
         for files in glob.glob("./httpRequests_1week/" +
            date + "/http.*.txt.gz"):
             time = files.split("/")[-1].split(".")[1]
                httpMethod[date][time] = {}
                result = subprocess.check_output("zcat␣" +
                   files + "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\n
                   \";␣FS=\"\\\\\\\\\t\"}␣$5~/199.30.181.42/␣
```

```
                    {print␣$8}'", shell = True).rsplit("\n")
                for record in result:
                 if record != "":
                     if record not in httpMethod[date][time
                        ]:
                         httpMethod[date][time][record] = 1
                        else:
                         httpMethod[date][time][record] += 1
        writeToFile(httpMethod, "./httpInfo_1week/httpMethod
           .json", "wb+")

def checkUserAgent(self):
        userAgent = {}
        dates = self.__getDateList()
        for date in dates:
                methodsPerDay = subprocess.check_output("
                   zcat␣./httpRequests/" + date + "/http.*.
                   gz" + "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\n\";␣
                   FS=\"\\\\\\\\\t\"}␣$5~/199.30.181.42/␣{
                   print␣$12}'", shell = True).rsplit('\n')
                for i in methodsPerDay:
                        if i != "":
                                if i not in userAgent.keys()
                                   :
                                        userAgent[i] = 1
                                else:
                                        userAgent[i] += 1
        writeToFile(userAgent, "./httpInfo/
```

```
            userAgent_checking.json", "wb+")


def httpStatusCode(self):
 httpStatusCode = {}
    dates = self.__getDateList()
    for date in dates:
     httpStatusCode[date] = {}
        if os.path.exists("./httpRequests/" + date):
         for files in glob.glob("./httpRequests/" + date + "
            /http.*.txt.gz"):
             time = files.split("/")[-1].split(".")[1]
                httpStatusCode[date][time] = {}
                result = subprocess.check_output("zcat␣" +
                   files + "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\n
                   \";␣FS=\"\\\\\\\\\t\"}␣($9~/d2l.ucalgary.
                   ca/)␣||␣($10~/d2l.ucalgary.ca/)␣||␣($11~/
                   d2l.ucalgary.ca/)␣{print␣$15}'", shell =
                   True).rsplit("\n")
                for record in result:
                 if record != "":
                     if record not in httpStatusCode[date][
                        time]:
                         httpStatusCode[date][time][record]
                            = 1
                        else:
                         httpStatusCode[date][time][record]
                            += 1
    writeToFile(httpStatusCode, "./httpInfo/
```

```
            httpStatusCode_All.json", "wb")
        print "httpStatusCode_All" + ":␣Done"

analyze = AnalyzeData()
```

Listing A.3: Script to compare HTTP vs HTTPS Traffic for a single day

```
#!/bin/python

###  HTTP vs HTTPs (Daily Analysis on Log Scales)
import json
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import urllib2
import os
from sharedMethods import *
import datetime
import itertools



if not os.path.exists("./week/March42016"):
    os.makedirs("./week/March42016")

httpRequests = readFromDict("./ssl2016Info/March_4_2016.json")
dates = sorted(httpRequests.keys())
print dates
for date in sorted(httpRequests.keys()):
  day = sorted(httpRequests[date].items(), key = lambda x: x[0])
```

```
        print day


httpRequests1 = readFromDict("./httpInfo/March_4_2016.json")
for date1 in sorted(httpRequests1.keys()):
                day1 = sorted(httpRequests1[date1].items(), key =
                   lambda x: x[0])
fig = plt.figure(figsize = (15, 8))
y1 = np.array([int(i[1].rstrip()) for i in day1])
print y1
y = np.array([int(i[1].rstrip()) for i in day])
print y
x = np.arange(len(y))
plt.bar(x, y, width = 0.25, color = 'r', align = 'center', label='
   HTTPS')
plt.bar(x+0.25, y1, width = 0.25, color = 'b', align = 'center',
   label='HTTP')
plt.xlim([-1, x.size])
plt.ylim([1,100000])
xticksString = [i[0].split("-")[0] + "-\n" + i[0].split("-")[1] for
   i in day]
plt.xticks(x, xticksString, rotation='vertical')
plt.xlabel('Time␣of␣Day',size=20)
plt.ylabel('Number␣of␣Requests',size=20)
plt.yscale('log')
plt.title(dates,size=20)
plt.subplots_adjust(bottom = 0.15)
plt.legend()
plt.show()
```

```
fig.savefig('./week100/March42016/' + date[0] + '.eps', format = '
   eps', dpi = fig.dpi)
```

## A.2 SSL Log Analysis Scripts

Listing A.4: Script to extract data from SSL Logs

```
import subprocess
import os
from sharedMethods import *
import glob

class Statistic:
 rootDir = "/data4/"

 def __init__(self):
  os.chdir(self.rootDir)

 def httpsInfo(self):
    """This function extracts all the records containing "d2l.
       ucalgary" in everyday's http logs.
        Field seperator is "\t", record seperator is "\n".
        """
  currentDir = subprocess.check_output("ls", shell = True)
  currentDir = currentDir.rsplit()
  for folder in currentDir:
   if os.path.isdir(folder) and folder[:2] == "20" and folder[:10]
      >= "2016-01-01" and folder[:10] <= "2016-04-30":
    if not os.path.exists("/home/sourish/https/sslRequests/" +
```

```
        folder):
      os.makedirs("/home/sourish/https/sslRequests/" + folder)
     for files in glob.glob(folder + "/ssl.*.log.gz"):
      httpRequestsPerHour = subprocess.check_output("zcat␣./" + files
          + "␣|␣gawk␣'/d2l\.ucalgary\.ca/␣{print␣$0}'", shell = True)
      writeToFile(httpRequestsPerHour, "/home/sourish/https/
         sslRequests/" + files[0:33] + ".txt", "wb+")
      subprocess.call("gzip␣-q␣/home/sourish/https/sslRequests/" +
         files[0:33] + ".txt", shell = True)

stat = Statistic()
stat.httpsInfo()
```

Listing A.5: Script to analyze data from SSL Logs

```
import os
import subprocess
import glob
from sharedMethods import *
import datetime
### This script analyzes the extracted data to compute specific
   requirements Eg. List of IPs, HTTPS Requests, HTTPS methods, etc.

class AnalyzeData:
 rootDir = "/home/sourish/https/"

 def __init__(self):
  os.chdir(self.rootDir)
  if not os.path.exists("./sslInfo1year"):
```

```
  os.makedirs("./sslInfo1year")
 if not os.path.exists("./ipInfoWinter2016"):
  os.makedirs("./ipInfoWinter2016")
 if not os.path.exists("./fileInfo1year"):
  os.makedirs("./fileInfo1year")
            self.httpsRequestTotalPerDay()
            self.ipInfo()
            self.httpsMethod()
            self.httpsStatusCode()
            self.ipBookPerDay()


def __getDateList(self):
 """This function returns a list of strings, of all the dates."""
 start = datetime.date(2016, 01, 01)
 end = datetime.date(2016, 04, 30)
 daysRange = (end - start).days
 return [str(start + datetime.timedelta(days = i)) for i in range(
    daysRange + 1)]


def httpsRequestTotalPerDay(self):
 """This function returns a dictionary containing the total number
    of requests per hour per day.
 E.g. {'2016-12-01': {'00...-01...': '1000\n', '01...-02...':
    '23423\n'} ...}"""
 httpsRequestTotalPerDay = {}
 dates = self.__getDateList()
```

```
 for date in dates:
  httpsRequestTotalPerDay[date] = {}
  for files in glob.glob("./sslRequests1year/" + date + "/ssl.*.txt
     .gz"):
   httpsRequestTotalPerDay[date][files.split("/")[-1].split(".")
      [1]] = subprocess.check_output("zcat␣" + files + "␣|␣gawk␣'
      BEGIN␣{i=0;␣RS=\"\\\\\\\\\n\";␣FS=\"\\\\\\\\\t\"}␣$5
      ~/199.30.181.42/␣{i++}␣END␣{print␣i}'", shell = True)
 writeToFile(httpsRequestTotalPerDay, "./sslInfo1year/
    httpsRequestTotalPerDay.json", "wb+")


def ipInfo(self):
 ipBook = {}
 dates = self.__getDateList()
 for date in dates:
  ipBookPerDay = subprocess.check_output("zcat␣./sslRequests/" +
     date + "/ssl.*.gz" + "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\n\";␣FS
     =\"\\\\\\\\\t\"}␣$5~/199.30.181.42/␣{print␣$3}'", shell = True)
     .rsplit('\n')
  for i in ipBookPerDay:
   if i not in ipBook:
    ipBook[i] = 1
   else:
    ipBook[i] += 1
 writeToFile(ipBook, "./ipInfoWinter2016/ipBook.json", "wb+")
def ipBookPerDay(self):
 """This function uses the data in httpsRequests, output a
```

```
    dictionary to the ipInfo/ipBookPerDay folder."""
 if not os.path.exists("./ipInfo1year/ipBookPerDay/"):
  os.makedirs("./ipInfo1year/ipBookPerDay/")
 dates = self.__getDateList()
 for date in dates:
  if os.path.exists("./sslRequests1year/" + date):
   ipBookPerDay = {}
   ipDataPerDay = subprocess.check_output("zcat␣./sslRequests1year/
      " + date + "/ssl.*.gz" + "␣|␣gawk␣'BEGIN␣{RS=\"\\\\\\\\\n\";␣
      FS=\"\\\\\\\\\t\"}␣$5~/199.30.181.42/␣{print␣$3}'", shell =
      True).rsplit('\n')
   for i in ipDataPerDay:
    if i not in ipBookPerDay:
     ipBookPerDay[i] = 1
    else:
     ipBookPerDay[i] += 1
  writeToFile(ipBookPerDay, "./ipInfo1year/ipBookPerDay/" + date +
     ".json", "wb+")



def httpsMethod(self):
 httpsMethod = {}
 dates = self.__getDateList()
 for date in dates:
  httpsMethod[date] = {}
  if os.path.exists("./sslRequests1year/" + date):
   for files in glob.glob("./sslRequests1year/" + date + "/ssl.*.
      txt.gz"):
```

```
    time = files.split("/")[-1].split(".")[1]
    httpsMethod[date][time] = {}
    result = subprocess.check_output("zcat␣" + files + "␣|␣gawk␣'
       BEGIN␣{RS=\"\\\\\\\\\\n\";␣FS=\"\\\\\\\\\\t\"}␣$5
       ~/199.30.181.42/␣{print␣$8}'", shell = True).rsplit("\n")
    for record in result:
     if record != "":
      if record not in httpsMethod[date][time]:
       httpsMethod[date][time][record] = 1
      else:
       httpsMethod[date][time][record] += 1
 writeToFile(httpsMethod, "./sslInfo1year/httpsMethod.json", "wb+")


def httpsStatusCode(self):
 httpsStatusCode = {}
 dates = self.__getDateList()
 for date in dates:
  httpsStatusCode[date] = {}
  if os.path.exists("./sslRequests1year/" + date):
   for files in glob.glob("./sslRequests1year/" + date + "/ssl.*.
      txt.gz"):
    time = files.split("/")[-1].split(".")[1]
    httpsStatusCode[date][time] = {}
    result = subprocess.check_output("zcat␣" + files + "␣|␣gawk␣'
       BEGIN␣{RS=\"\\\\\\\\\\n\";␣FS=\"\\\\\\\\\\t\"}␣$5
       ~/199.30.181.42/␣{print␣$15}'", shell = True).rsplit("\n")
    for record in result:
```

```
      if record != "":
       if record not in httpsStatusCode[date][time]:
        httpsStatusCode[date][time][record] = 1
       else:
        httpsStatusCode[date][time][record] += 1
  writeToFile(httpsStatusCode, "./sslInfo1year/httpsStatusCode.json"
     , "wb+")


analyze = AnalyzeData()
```

## A.3 Connection Log Analysis Scripts

Listing A.6: Script to extract inbound and outbound data using D2L IP as filter

```
#!/bin/bash
#shell used to count all the inbound and outbound packet size of D2L
    records for Winter 2016. Since we are only concerned with D2L
   traffic  the D2L IP 199.30.181.42 is used as a filter.

log_path_in="/data4/*"
result_path_in="$HOME/results/d2lconnlog/in-size"
date_form_in=2016-0[1-4]{1}-[0-9]{2}

#date_form_in="2016-01-02"
for dir in $log_path_in; do
        if [[ -d $dir && $dir =~ $date_form_in ]]; then
                new_path_in="$dir/*"
```

```
                month_in=${dir###*/}
                month_in=${month_in%-*}
                mkdir -p "$result_path_in/$month_in"
                cd "$result_path_in/$month_in"
                date_name_in=${dir###*/}
                mkdir $date_name_in
                cd $date_name_in
                for file in $new_path_in; do
                        if [[ $file =~ "conn." ]]; then
                                date
                                name_of_file_in=${file#*.}   #first
                                   '.' and left
                                name_of_file_in=${name_of_file_in
                                   %.*}  #last '.' and right
                                name_of_file_in=${name_of_file_in
                                   %.*}  #last '.' and right
                                echo -ne '' > $name_of_file_in
                                less $file | awk -F"\t" '
                                   /199.30.181.42/␣{print␣$10,$11␣}'
                                    >> $name_of_file_in
                                echo -n "done!␣-␣$name_of_file_in␣-␣
                                   ${dir###*/}␣␣␣"
                                date
                        fi
                done
        fi
done
```

Listing A.7: Script to extract D2L session durations

```
#!/bin/bash
#shell script used to extract D2L session durations.

#Winter 2016 Term
log_path_in="/data4/*"
result_path_in="$HOME/results/d2lduration2016/in-duriation-2016"
date_form_in=2016-0[1-4]{1}-[0-9]{2}
for dir in $log_path_in; do
        if [[ -d $dir && $dir =~ $date_form_in ]]; then
                new_path_in="$dir/*"
                month_in=${dir##*/}
                month_in=${month_in%-*}
                mkdir -p "$result_path_in/$month_in"
                cd "$result_path_in/$month_in"
                date_name_in=${dir##*/}
                mkdir $date_name_in
                cd $date_name_in
                for file in $new_path_in; do
                        if [[ $file =~ "conn." ]]; then
                                date
                                name_of_file_in=${file#*.}
                                name_of_file_in=${name_of_file_in
                                   %.*}
                                name_of_file_in=${name_of_file_in
                                   %.*}
                                echo -ne '' > $name_of_file_in
                                less $file | awk -F"\t" '
```

```
                                    /199.30.181.42/␣{print␣$9}' >>
                                    $name_of_file_in
                                echo -n "done!␣-␣$name_of_file_in␣-␣
                                    ${dir##*/}␣␣␣"
                                date
                        fi
                done
        fi
done
```

## A.4 URL Analysis Scripts

Listing A.8: Script to extract Pre-D2L and Post-D2L Referer URLs

```
import os
import subprocess
import glob
from sharedMethods import *
import datetime

### Extracts Pre-D2L Referer and Post-D2L Referer URLS from HTTP
    logs
class AnalyzeData:
        rootDir = "/home/sourish/m/"

        def __init__(self):
         os.chdir(self.rootDir)
         if not os.paath.exists("./httpInfo"):
             os.makedirs("./httpInfo")
```

```
        self.postrefererBookTotal()
        self.refererBookTotal()

    def __getDateList(self):
        """This function returns a list of strings, of all the
           dates.
           """
        start = datetime.date(2016, 01, 01)
           end = datetime.date(2016, 04, 30)
           daysRange = (end - start).days
           return [str(start + datetime.timedelta(days = i)) for i
              in range(daysRange + 1)]

###PRE REFERERS###
    def refererBookTotal(self):
        refererBookTotal = {}
        dates = self.__getDateList()
           for date in dates:
            for files in glob.glob("./httpRequests/" + date + "/
               http.*.txt.gz"):
                 result = subprocess.check_output("zcat " + files +
                     " | gawk 'BEGIN {RS=\"\\\\\\\\\\n\"; FS
                    =\"\\\\\\\\\t\"} ($12 !~ /Wget/) && ($9~/d2l.
                    ucalgary.ca/) {if ($11 != \"-\") {print $11
                    \"\\t\" $14}}'", shell = True).rsplit("\n")
                   for i in result:
                    i = i.split("\t")
                       if (i[0] != "-") and (i[0] != "") and (i[1]
```

```
                        != "-") and (i[1] != ""):
                    if (i[0] not in refererBookTotal):

                        refererBookTotal[i[0]] = [1, int(i[1])]
                        else:
                            refererBookTotal[i[0]][0] += 1
                            refererBookTotal[i[0]][1] += int(i
                                [1])
        writeToFile(refererBookTotal, "./httpInfo/
            refererBookTotal.json", "wb")
        print "refererBookTotal1" + ": Done"

###POST REFERERS###

    def postrefererBookTotal(self):
     refererBookTotal = {}
        dates = self.__getDateList()
        for date in dates:
         for files in glob.glob("./httpRequests/" + date + "/
             http.*.txt.gz"):
              result = subprocess.check_output("zcat " + files +
                 " | gawk 'BEGIN {RS=\"\\\\\\\\\\n\"; FS=\"\\\\\\\\
                 t\"} ($12 !~ /Wget/) && (($10~/d2l.ucalgary.ca/)
                 || ($11~/d2l.ucalgary.ca/)) {if ($11 != \"-\") {
                 print $11 \"\\t\" $14}}'", shell = True).rsplit
                 ("\n") #For actual referers $11
                 for i in result:
                  i = i.split("\t")
```

```
                        if (i[0] != "-") and (i[0] != "") and (i[1]
                           != "-") and (i[1] != ""):
                            if (i[0] not in refererBookTotal):
                                refererBookTotal[i[0]] = [1, int(i[1])]
                            else:
                                refererBookTotal[i[0]][0] += 1
                                   refererBookTotal[i[0]][1] += int(i
                                      [1])
            writeToFile(refererBookTotal, "./httpInfo/
               post_URIs_referer_1week_4to11march.json", "wb")
            print "refererBookTotal1" + ":␣Done"

analyze = AnalyzeData()
```

Listing A.9: Script to analyze Pre-D2L and Post-D2L Referer URL Popularity along with Best Fit Line

```
#/bin/python

# Scripts to draw POST-D2L and PRE-D2L REFERERS along with BEST FIT
   LINE

import numpy as np
import matplotlib.pyplot as plt
import os
import matplotlib.pyplot as plt
import numpy as np
import math
from numpy import *
```

```
import scipy
from scipy.interpolate import *
from scipy.stats import linregress
from scipy import stats
from matplotlib.ticker import FuncFormatter
from scipy.optimize import curve_fit
from scipy import polyfit
font = {'family': 'serif',
    'color':  'darkblue',
        'weight': 'normal',
        'size': 15,
    }



path1='/Users/roysourish/Desktop/fileInfo/'
#filename1=path1+'postreferer_4months_Winter_2016.txt' #course,count
   .. split('\t') For POST_REFERER
filename1=path1+'pre_refererTime.txt' #FOR PRE_REFERER
#To accepts any of \r, \n, \r\n as a newline we could use 'U' (
   universal newline) file mode:
file = open(filename1, 'r')
lines = file.readlines()

result = {}
x=[]
y=[]
for line in lines:
```

```
    #course,count = line.lstrip().rstrip('\n').split('\t') #4months
       post referer
    course,count = line.lstrip().rstrip('\n').split('\t\t') #4months
        pre referer
    if course not in result:
        result[course] = int(count)
    else:
        result[course] += int(count)
file.close()

frequency = sorted(result.items(), key = lambda i: i[1], reverse=
   True)
x=[]
y=[]
i=0
for element in frequency:
    x.append(element[0])
    y.append(element[1])
z=[]
fig=plt.figure()
ax = fig.add_subplot(111)
z=np.arange(len(x)) + 1
print z
print y
rank = [np.log10(i) for i in z]
freq = [np.log10(i) for i in y]
m, b, r_value, p_value, std_err = stats.linregress(rank, freq)
print "slope:␣", m
```

```
print "r-squared:␣", r_value**2
print "intercept:", b
plt.plot(rank, freq, 'o', color = 'r')
abline_values = [m * i + b for i in rank]
plt.plot(rank, abline_values)
plt.title('Frequency-Rank␣Profile␣for␣Pre-D2L␣Referer␣URLs␣(Winter␣
   2016)', fontdict=font)
#plt.title('Frequency-Rank Profile for Post-D2L Referer URLs (Winter
    2016)', fontdict=font)
plt.xlabel('log10(Rank)', fontdict=font)
plt.ylabel('log10(Frequency)', fontdict=font)
plt.ylim(0,6)
plt.xlim(0,4)
plt.show()
```

# Appendix B

# Security Related Analysis

In this appendix we discuss two security related issues that are peripherally related to the thesis. The first section discusses cookies that are used by the D2L system. The second section discusses end-user devices that were not using HTTPS for their D2L interactions.

## B.1 Cookie Analysis

Cookies

- CAS: 2 cookies
- d2l : 2 cookies
- ucalgary.ca : 19 cookies
- d2l.ucalgary.ca: 2 cookies
- ucalgary.blogs.ca.dnsdb.org : 1 cookie
- cpsc.ucalgary.ca : 3 cookies
- www.fs.ucalgary.ca : 1 cookie
- fed.ucalgary.ca : 2 cookies
- my.ucalgary.ca : 2 cookies
- password.ucalgary.ca : 1 cookie
- www.ucalgary.ca : 1 cookie

Table B.1: Domains hosted by IPs

| IP | Number of Domains hosted |
|---|---|
| 136.159.208.23 | 30 |
| 136.159.208.52 | 1 |
| 136.159.208.56 | 1 |
| 136.159.208.132 | 1 |

## B.2 Unencrypted Devices

We observed a few unencrypted devices accessing D2L during the Winter 2016 term. Most of these devices are seen multiple times in our HTTP logs. The visible devices are summarized in Table B.2. All the Nexus traffic is from the same mobile device and browsers as is evident from the device model number `Nexus 6 Build/MMB29S` and the user agent information in the trace which shows: `Version/4.0 Chrome/44.0.2403.117 Mobile Safari/537.36`.

A few reasons for the visibility of these device could be:

1. The user of the Nexus device with an Android OS had experimented with the device and had turned off the device encryption, which makes it visible in our logs. The device encryption is automatically set for users in general.

2. It is possible to find out more details about the device owner, since the only two courses accessed over the four month period from this device were `106867` and `122347`.

3. We speculate that the device might use a man in the middle proxy server to access HTTPS data. Such proxies are presently available.

This information should be forwarded to the U of C and D2L, so that the correct reason for the visibility of these devices can be identified, as this could be a security vulnerability.

Table B.2: Visible Devices accessing D2L

| Device / OS | Count |
|---|---|
| X11; Linux x86_64; rv:10.0 | 68 |
| X11; Fedora; Linux x86_64 | 39 |
| Android 6.0.1; Nexus 6 Build/MMB29S; wv | 37 |
| Windows NT 6.3; WOW64 | 6 |
| Windows NT 6.1; WOW64; Trident/7.0; rv:11.0 | 6 |

# Appendix C

# NAT Analysis

This appendix discusses the details of the network address translation (NAT) devices used by the Department of Computer Science (CPSC) at the University of Calgary (U of C). Theses devices were in place at the time of our study.

## C.1 NAT Analysis

There are 5 varieties of NAT in total for the Computer Science (CS) department, and the range of these NAT-generated IPs are illustrated in Table C.1. We observed thirty such NAT generated IPs for the Computer Science department. We performed `nslookup` on each of these IPs in a Linux environment to derive the NAT names. Table C.1 shows that there are six IPs allocated for every CPSC NAT category, and five separate NAT categories can be seen, namely `server`, `tech`, `studentUNIX`, `facultyMS`, and `studentMS`. However, there is one NAT category `buildnet`, which has only a single IP allocated to it. For students, the IPs are mostly allocated from the studentUNIX or the studentMS category.

Table C.1: NAT types for the Computer Science Dept. at the U of C

| NAT type | IP Range | NAT Name |
|---|---|---|
| server | 136.159.16.1 - 136.159.16.6 | `ms-server-NAT0.cs.ucalgary.ca` - `ms-server-NAT5.cs.ucalgary.ca` |
| tech | 136.159.16.10 - 136.159.16.15 | `ms-tech-NAT0.cs.ucalgary.ca` - `ms-tech-NAT5.cs.ucalgary.ca` |
| studentUNIX | 136.159.16.20 - 136.159.16.25 | `ms-studentUNIX-NAT0.cs.ucalgary.ca` - `ms-studentUNIX-NAT5.cs.ucalgary.ca` |
| facultyMS | 136.159.16.40 - 136.159.16.45 | `ms-facultyMS-NAT0.cs.ucalgary.ca` - `ms-facultyMS-NAT5.cs.ucalgary.ca` |
| studentMS | 136.159.16.50 - 136.159.16.55 | `ms-studentMS-NAT0.cs.ucalgary.ca` - `ms-studentMS-NAT5.cs.ucalgary.ca` |
| buildnet | 136.159.16.240 | `ms-buildnet-NAT0.cs.ucalgary.ca` |

Table C.2: NAT generated HTTPS IP rank frequency

| IP Rank | NAT IP | IP Frequency |
|---|---|---|
| 15 | 136.159.16.7 | 619,818 |
| 33 | 136.159.16.16 | 192,414 |
| 49 | 136.159.16.20 | 31,847 |
| 60 | 136.159.16.14 | 14,060 |
| 107 | 136.159.16.10 | 4,877 |
| 2,938 | 136.159.16.19 | 571 |
| 7,627 | 136.159.16.4 | 50 |
| 9,665 | 136.159.16.240 | 14 |

Table C.3: NAT generated HTTP IP rank frequency

| IP Rank | NAT IP | IP Frequency |
|---|---|---|
| 16 | 136.159.16.7 | 27,804 |
| 33 | 136.159.16.16 | 8,339 |
| 55 | 136.159.16.20 | 808 |
| 80 | 136.159.16.14 | 308 |
| 135 | 136.159.16.10 | 182 |
| 478 | 136.159.16.19 | 66 |
| 3,358 | 136.159.16.4 | 12 |
| 7,062 | 136.159.16.240 | 2 |

As we had mentioned earlier the highest IP rank holders are mostly generated by DHCP servers or NAT. We consider them to be DHCP since they are selected from a pool of available IPs. For HTTPS the top 14 IPs are all DHCP generated, with a total of 25,107,109. The list of NAT generated HTTPS IPs along with their ranks are shown in Table C.2.

For HTTP the top 14 IPs are also DHCP generated, with a total frequency of 529,573. A similar list of NAT generated HTTP IPs along with their ranks are shown in Table C.3.

# Bibliography


[1] J. Abbate. *From ARPANET to Internet: A History of ARPA-sponsored Computer Networks*. PhD thesis, Philadelphia, PA, USA, 1995.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control RFC 5681, September 2009.

[3] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of Educational Media Server Workloads. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '01, pages 21–30, Port Jefferson, New York, USA, January 2001. ACM.

[4] M. Arlitt and C. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '96, pages 126–137, Philadelphia, Pennsylvania, USA, May 1996. ACM.

[5] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[6] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. A Comparative Analysis of Web and Peer-to-peer Traffic. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 287–296, Beijing, China, April 2008. ACM.

[7] M. Belshe, M. Thomson, and R. Peon. Hypertext Transfer Protocol Version 2 (HTTP/2), RFC 7540. *IETF Internet Engineering Task Force*, May 2015.

[8] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing User Behavior in Online Social Networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 49–62, Chicago, Illinois, USA, November 2009. ACM.

[9] T. Berners-Lee. Information Management: A proposal, Technical Report, CERN. March 1989.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol (HTTP/1.0), RFC 1945 , May 1996.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134, New York, NY, USA, March 1999. IEEE.

[12] Brightspace By D2L. Overview of the Brightspace Cloud Content Delivery Network. `https://documentation.brightspace.com/EN/architecture/cdn/admin/cdn_overview.htm`.

[13] Brightspace By D2L. The Brightspace Cloud Content Delivery Network. `https://community.brightspace.com/resources`.

[14] Bro. The Bro Network Security Monitor. `https://www.bro.org/`.

[15] R. Cáceres, P.B. Danzig, S. Jamin, and D.J. Mitzel. Characteristics of Wide-area TCP/IP Conversations. In *Proceedings of the Conference on Communications Architecture & Protocols*, SIGCOMM '91, pages 101–112, Zurich, Switzerland, August, 1991. ACM.

[16] Cisco Community. Layer 3. `https://supportforums.cisco.com/document/68426/communication-network-layer-layer-3`.

[17] Content Delivery Networks. `https://www.incapsula.com/cdn-guide/what-is-cdn-how-it-works.html`.

[18] M. Crovella and B. Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., 2006.

[19] M. Crovella and M. Taqqu. Estimating the Heavy Tail Index from Scaling Properties. *Methodology And Computing In Applied Probability*, 1(1):55–79, July 1999.

[20] D2L. Brightspace Binder. `https://www.d2l.com/products/binder/`.

[21] D2L. Competency Based Education. `https://www.d2l.com/newsroom/releases/desire2learn-supports-educators-striving-to-enhance-student-learning-outcomes/`.

[22] D2L. D2L Selects Amazon Web Services. `https://www.d2l.com/newsroom/releases/d2l-selects-amazon-web-services-cloud-improve-learning-worldwide/`.

[23] D2L. Integrated Learning Platform. `https://www.d2l.com/newsroom/releases/desire2learn-offers-the-first-fully-integrated-learning-platform-\to-higher-education-k-12-and-enterprises-worldwide/`.

[24] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol, RFC 5246. *IETF The Internet Engineering Task Force*, August 2008.

[25] N. Duffield. Sampling for Passive Internet Measurement: A Review. *Statistical Science*, 19(3):472–498, August 2004.

[26] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol—HTTP/1.1, RFC 2068. *IETF Internet Engineering Task Force*, January 1997.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol–HTTP/1.1 RFC 2616. *Network Working Group*, 1999.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol–HTTP/1.1., The Internet Society. Technical report, 1999.

[29] V. Fuller and T. Li. Classless Inter-Domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan, RFC 4632. *The Internet Society*, 2006.

[30] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube Traffic Characterization: A View from the Edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 15–28, San Diego, California, USA, October 2007. ACM.

[31] Google. Search Console Help. `https://support.google.com/webmasters/answer/6073543?hl=en`.

[32] LBNL's Network Research Group. Tcpdump. `http://www.tcpdump.org/`.

[33] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[34] T. Haenselmann. Lecture on Computer Networks, Saarland University, Germany. `saarland.de/teaching/dn08/02_physical.pdf`.

[35] F. Hernández-Campos, K. Jeffay, and F.D. Smith. Tracking the Evolution of Web Traffic: 1995-2003. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 16–25, Orlando, FL, USA, USA, October, 2003. IEEE.

[36] HTTP Statuses httpstatuses.com. 3XX Redirection. `https://httpstatuses.com/304`.

[37] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance, RFC 1323. *IETF The Internet Engineering Task Force*, May 1992.

[38] B. Krishnamurthy, P. Gill, and M. Arlitt. A Few Chirps About Twitter. In *Proceedings of the First Workshop on Online Social Networks*, WOSN '08, pages 19–24, Seattle, WA, USA, August 2008. ACM.

[39] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001.

[40] J. Kurose and K. Ross. *Computer Networking: A Top-down Approach*. Addison-Wesley Reading, 2010.

[41] A. Luotonen and K. Altis. World-Wide Web Proxies. In *Selected Papers of the First Conference on World-Wide Web*, pages 147–154, Amsterdam, The Netherlands, November 1994. Elsevier Science Publishers B. V.

[42] A. Mahanti, N. Carlsson, A. Mahanti, M. Arlitt, and C. Williamson. A tale of the tails: Power-laws in internet measurements. *IEEE Network*, 27(1):59–64, February 2013.

[43] mitmproxy. Man in the Middle Proxy. `https://mitmproxy.org/`.

[44] P. Mockapetris. Domain Names: Implementation and Specification, RFC 883. *IETF Internet Engineering Task Force*, November 1983.

[45] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 50–60, Banff, Alberta, Canada, June 2005. ACM.

[46] B. Newton, K. Jeffay, and J. Aikat. The Continued Evolution of Web Traffic. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, pages 80–89, San Francisco, CA, USA, August 2013. IEEE.

[47] V. Paxson. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks*, 31(23):2435–2463, December 1999.

[48] V. Paxson. Growth Trends in Wide-area TCP Connections. *IEEE Network*, 8(4):8–17, July-August 1994.

[49] V. Paxson and S. Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995.

[50] D2L Developer Platform. Investigating Role Permissions. http://docs.valence.desire2learn.com/samples/permissionCheck.html.

[51] J. Postel et al. , Internet Protocol, RFC 791. *IETF The Internet Engineering Task Force*, September 1981.

[52] J. Postel et al. , Transmission Control Protocol, RFC 793, September 1981.

[53] E. Rescorla. HTTP over TLS, RFC 2818. *IETF The Internet Engineering Task Force*, May 2000.

[54] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 457–466, Hyderabad, India, March 2011. ACM.

[55] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding Online Social Network Usage from a Network Perspective. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 35–48, Chicago, Illinois, USA, November 2009. ACM.

[56] Shibboleth. About shibboleth. https://shibboleth.net/about/.

[57] P. Smith. *Professional Website Performance: Optimizing the Front-End and Back-End*. John Wiley & Sons, 2012.

[58] Kun Tan and Jingmin Song. Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks. In *in 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006*, Nara, Japan, February 2006.

[59] Statistics How To. Heavy-tailed distribution. http://www.statisticshowto.com/heavy-tailed-distribution/.

[60] tutorialspoint. HTTP - Requests. https://www.tutorialspoint.com/http/http_requests.htm.

[61] tutorialspoint. HTTP - Response. https://www.tutorialspoint.com/http/http_responses.htm.

[62] W3C. HTTP Request Fields. https://www.w3.org/Protocols/HTTP/HTRQ_Headers.html.

[63] Wikimedia. TCP header. https://commons.wikimedia.org/wiki/File:TCP_header.png.

[64] Wikimedia. UDP header. https://commons.wikimedia.org/wiki/File:UDP_header.png.

[65] Wikipedia. About D2L CEO. https://en.wikipedia.org/wiki/John_Baker_(entrepreneur).

[66] Wikipedia. Classless Inter-Domain Routing. https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

[67] Wikipedia. Competency based learning. https://en.wikipedia.org/wiki/Competency-based_learning.

[68] Wikipedia. Content Delivery Network. https://en.wikipedia.org/wiki/Content_delivery_network.

[69] Wikipedia. cURL. https://en.wikipedia.org/wiki/CURL.

[70] Wikipedia. Ephemeral Port. https://en.wikipedia.org/wiki/Ephemeral_port.

[71] Wikipedia. Heavy-tailed distribution. https://en.wikipedia.org/wiki/Heavy-tailed_distribution.

[72] Wikipedia. History of D2L. https://en.wikipedia.org/wiki/D2L#cite_note-4.

[73] Wikipedia. Integrated Learning Systems. https://en.wikipedia.org/wiki/Integrated_learning_systems.

[74] Wikipedia. Internet Protocol Suite. https://en.wikipedia.org/wiki/Internet_protocol_suite.

[75] Wikipedia. IPv4. https://en.wikipedia.org/wiki/IPv4.

[76] Wikipedia. Network Address Translation. https://en.wikipedia.org/wiki/Network_address_translation.

[77] Wikipedia. Transport layer security. https://en.wikipedia.org/wiki/Transport_Layer_Security.

[78] Wikipedia. User Datagram Protocol. https://en.wikipedia.org/wiki/User_Datagram_Protocol.

[79] Wikipedia. Web Server. https://en.wikipedia.org/wiki/Web_server.

[80] Wikipedia. Port (computer networking), 2017.

[81] C. Williamson. Internet Traffic Measurement. *IEEE Internet Computing*, 5(6):70–74, November/December 2001.

[82] Wireshark. https://www.wireshark.org/.

[83] Wireshark. Ethereal is now Wireshark. https://www.wireshark.org/news/20060607.html.