A publication of ISCA*:
International Society for Computers
and Their Applications

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

*** "International Journal of Computers and Their Applications is abstracted and indexed in INSPEC and Scopus."**

Volume 25, No. 3, Sept. 2018 **ISSN 1076-5204**

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

ISCA Headquarters···•···P. O. Box 1124, Winona, MN 55987 USA···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca@isca-hq.org.

Guest Editorial: Selected Papers from CATA 2018

CATA (Computers and their Applications) is one of the flagship conferences for the International Society of Computers and their Applications. The intent of this conference has been to blend theory and practice as a means of stimulating researchers from both research dimensions. The papers for this special issue are extensions of their conference version and illustrate the spectrum of the 38 papers presented at the CATA 2018 conference.

Paper 1: *Labeling Method Using EEG to Predict Drowsy Driving with Facial Expression Recognition Technology.* Authors: Daichi Naito, Ryo Hatano and Hiroyuki Nishiyama. In this paper, the authors develop a non-contact method for predicting a driver's drowsiness based on a set of classifiers that make use of a combination of EEG readings and facial expression capture for the same time period. They use a 3D camera and employ several machine learning methods as well as automated labeling for prediction. The Gaussian kernel SVM method gave the best performance and results show that their approach provides a viable and efficient approach for predicting a driver's drowsiness.

Paper 2: *LMS Performance Issues: A Case Study of D2L.* Authors: Sourish Roy, Carey Williamson, and Rachel McLean. The paper focuses on a particular network, the Learning Management System at the University of Calgary, and the authors perform a study of the traffic flow, identify the causes for traffic congestion, offer several solutions to the congestion problem, and present a web cache simulation study to show that this simple solution alleviates traffic problems.

Paper 3: *Performance Evaluations of Two Fast Join Query Processing Methods: Join Core and Join Indices.* Authors: Reham M. Almutairi, Mohammed Hamdi, Feng Yu, and Wen-Chi Hou. An evaluation of the performance of the relational database model is provided in this paper. In particular, the authors compared the performance of the Jive-join algorithm, the Join Core algorithm and the MySQL implementation. It is shown that the join core performs faster that the join indices, although both methods perform well for TCP-H benchmark datasets.

Paper 4: *Novel Low Latency Load Shared Multicore Multicasting Schemes – An Extension to Core Migration.* Authors: Bidyut Gupta, Ashraf Alyanbaawi, Koushik Sinha, and Nick Rahimi. The authors present four schemes for load shared multicasting using static networks. Two of the methods consider load sharing as the performance metric, while the other two also focus on low latency multicasting. Further, the authors discuss the feasibility of these strategies.

Gordon Lee (Program Chair, CATA 2018) and Les Miller (General Chair, CATA 2018)

Labeling Method Using EEG to Predict Drowsy Driving with Facial Expression Recognition Technology

Daichi Naito*, Ryo Hatano*, and Hiroyuki Nishiyama*
Tokyo University of Science, Yamazaki 2641, Noda-shi, CHIBA, 278-8510, JAPAN

Abstract

Accidental death caused by driver negligence is a serious problem, and the most common cause of such accidents is careless driving, in particular, driving while drowsy. Drowsy driving can be caused by overwork and may lead to serious accidents. Studies have proposed using an electroencephalogram (EEG), pulse, and facial expressions to predict drowsy driving. Predicting drowsy driving using an EEG or pulse, however, may require the driver to wear cumbersome devices, which is likely to disturb their concentration while driving. We therefore aim to predict drowsy driving from a driver's facial expressions in a non-contact manner. In particular, we record facial expressions with a 3D camera, and predict drowsy driving using machine learning. This, however, requires visually checking the driver’s state, to label the data, which is time consuming and makes it difficult to prepare sufficient labeled data for machine learning. In this study, we propose an automated method for labeling, using a machine learning technique based on EEG data. Evaluation of the proposed method is conducted, based on accuracy and the F-value, for unknown data prediction.

Key Words: Drowsy, driving, machine learning, labeling, EEG, facial expressions.

1 Introduction

Deaths caused by accidents resulting from driver negligence have become a serious problem in recent years. Figure 1 shows the yearly trend in causes of fatal accidents due to driver negligence in Japan [7]. The Metropolitan Police Department Transportation Bureau reports that the most common cause of accidental death from driver negligence is careless driving. It is therefore seen as being greatly beneficial if the rate of careless driving could be reduced.

“Careless driving” refers to situations in which the driver is unable to concentrate on driving – such as driving when drowsy. As drowsy driving can cause significant traffic accidents, we focus on predicting its occurrence. Much of the literature on drowsy driving prediction deals with the driver’s pulse, electroencephalogram (EEG), and facial expressions, among other indicators [3]. To predict drowsy driving using an EEG or pulse, it is necessary to attach a device to the driver’s body, and this may disturb their concentration while driving, making the use of such attachable devices to predict drowsy driving impractical or undesirable.

We therefore aim to predict drowsy driving with a non-contact approach. To that end, we record drivers’ facial expressions using a 3D camera, and predict drowsy driving with machine learning. However, developing an effective machine learning model requires a considerable amount of time and effort, as it is necessary for humans to watch many videos of drivers to categorize (label) the training data. This makes it difficult to prepare enough training data to obtain a generalized prediction model, an issue which is probably also similar for other studies that employ an approach similar to ours. To tackle this problem, we propose a new labeling approach, using EEG data to shorten the time required for labeling, as it is known that an EEG is helpful for determining the level of sleepiness [1, 6].

We organize the rest of this paper as follows: Section 2 summarizes several studies with respect to sleeping and drowsy driving; Section 3 presents an overview of our method and the principles of machine learning; Section 4 shows the experimental environment and the analysis of our results; and Section 5 concludes this paper with further remarks.

2 Related Works

Nishiyama et al. [5] proposed a method to predict drowsy driving using drivers’ facial expression data, obtained with a 3D camera. They applied inductive logic programming to the data, and obtained a set of rules that could be interpreted as corresponding formulas of first-order logic. The obtained rules involved expressions that allowed for prediction of drowsy driving 10 s before its occurrence. Nevertheless, they faced the aforementioned problem of collecting enough training data, owing to the time required to label videos. For this reason, there was a risk of overfitting such a model, and increasing the size of the training dataset was viewed as imperative.

* Department of Industrial Administration, Faculty of Science and Technology. Email: 7417615@ed.tus.ac.jp.

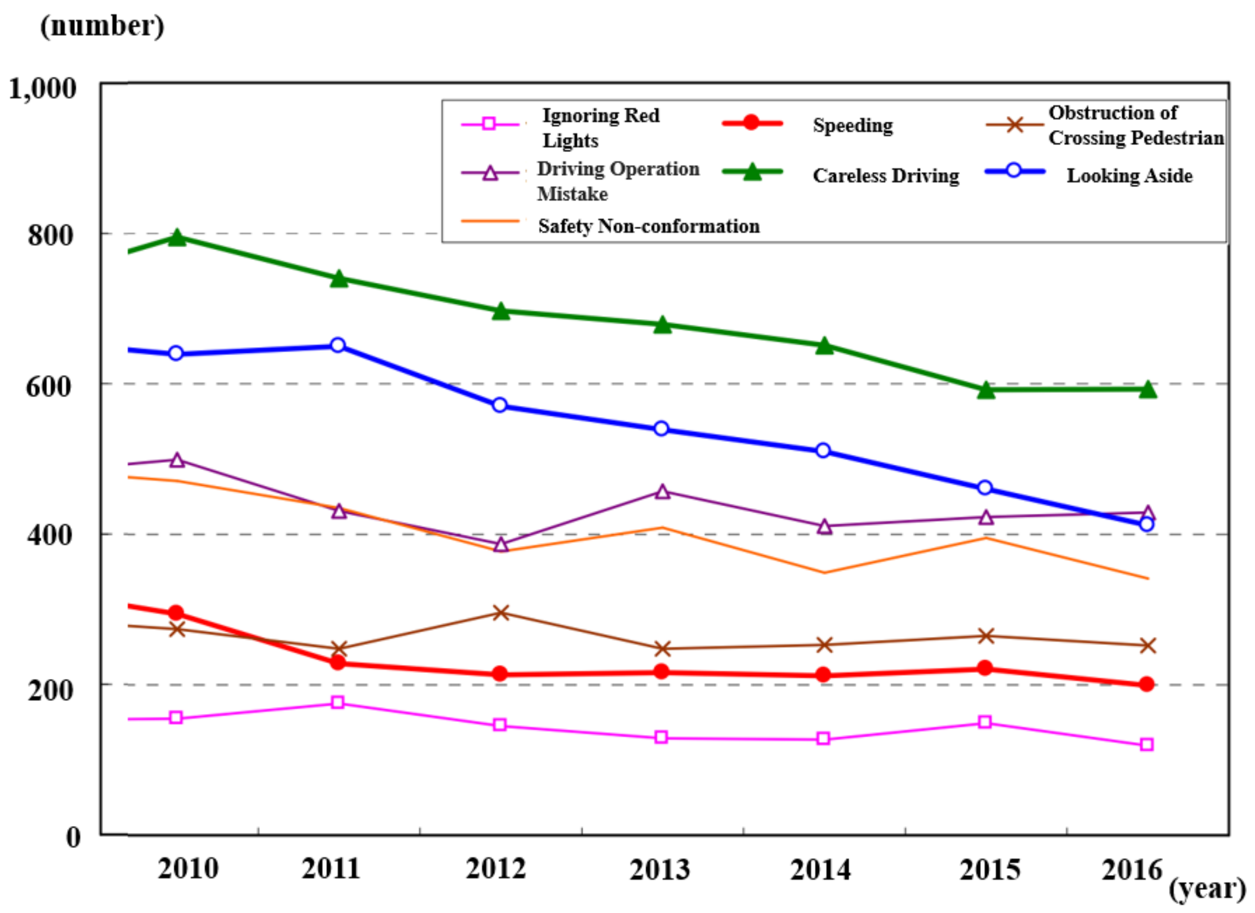


Figure 1: Trend in the number of fatal accidents by law violation

Tarek et al. [2] classified the sleep stage of a person with a support vector machine (SVM) using EEG, electrooculogram (EOG), and electromyogram (EMG) data. This study involved classification of whether a subject was sleeping throughout the night, and hence this approach could not be directly applied for classification of drowsy driving. Moreover, the device used appeared to be too large to attach to subjects, for our purpose.

To predict drowsy driving precisely, using facial expressions occurring while driving, a large training set is required. In this study, we propose a labeling method using EEGs to construct such a training set efficiently.

3 A Method for Automatic Data Labeling

3.1 Overview of our Method

Our method is roughly divided into two phases, as shown in Figure 2, and in this method, the experiment is performed twice. Firstly, we record the EEG during driving (Experiment 1), and then secondly, we record both the EEG and facial expressions (Experiment 2). In both Experiments 1 and 2, we video the driver to verify their driving behavior after the experiments. The EEGs are recorded by an electroencephalograph, and the driver's facial expressions are recorded with a 3D camera. To make a classification in Phase 1, we use the recorded video to label the state of the driver manually, for the EEG dataset acquired in Experiment 1. In Phase 2, we apply the EEG dataset acquired in Experiment 2 to the classifier obtained from Phase 1; the classification results are then used to label the facial expression dataset acquired in Experiment 2. Once we obtain the classifier from Phase 1, we can automatically obtain a labeled dataset using the classifier in Phase 2.

3.2 Creating a Dataset

In this study, we define labels for two states - awake and drowsy. In general, EEGs are divided into five types at each range of frequencies - α, β, γ, δ, and θ waves. The amount of EEG activity is also thought to change depending on the actions of the brain [8]. In this study, we categorize each

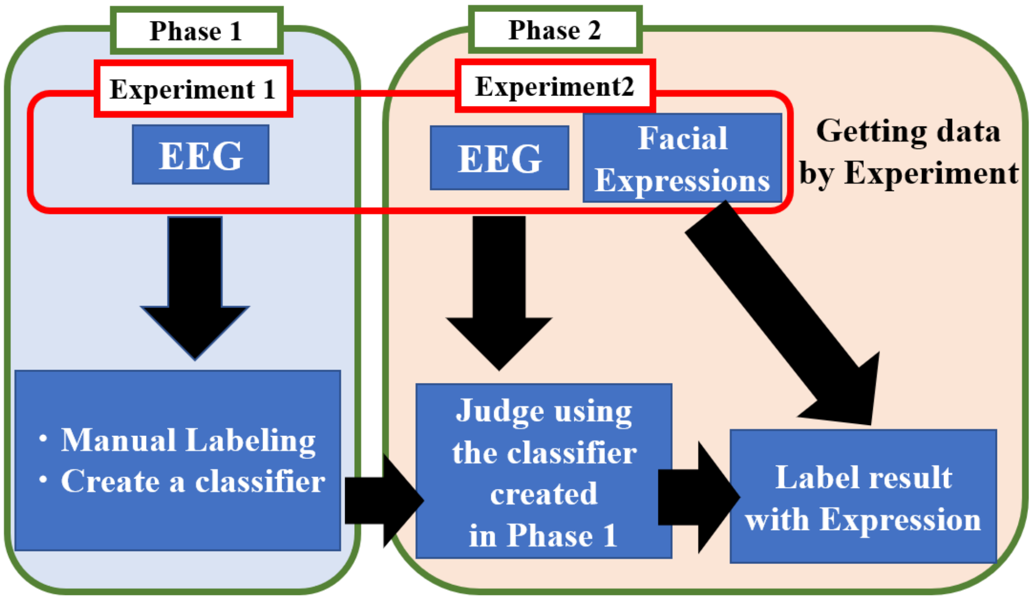

Phase 1
Experiment 1
EEG
・Manual Labeling
・Create a classifier
Phase 2
Experiment2
EEG
Facial Expressions
Getting data by Experiment
Judge using the classifier created in Phase 1
Label result with Expression

Figure 2: Proposed Method Flow

EEG into one of eight types (see also Table 2): α1, α2, β1, β2, γ1, γ2, δ, and θ waves. The definition of these types is based on the specification of the electroencephalograph used in this study [4]. Here, we define the set E = {α1, α2, β1, β2, γ1, γ2, δ, and θ} of EEG types for convenience. The electroencephalograph used in this study can acquire the power spectral density for the range of each of the above eight types of EEG, every second.

As we deal with time series data, it is necessary to extract the feature quantity as one window every few seconds, and it is also important to be able to determine features in real time. For this reason, we extract the features using a sliding window (the size of a window is 5 s) that shifts every second. The sliding window technique is often used for time series data, and shifting the window every second means that prediction can be performed with 5 s of data every second, which allows us to construct a rich dataset for the training data. Each window of 5 s was labeled as either "drowsy", or "awake", according to the following conditions:

- Drowsy: a driver is drowsy for more than 3 s.
- Awake: a driver remains awake for more than 5 s.

We excluded instances of drowsy driving lasting 1–2 s, since the status of a person in this case seemed ambiguous.

From each of the acquired EEG types, we extract features of the sliding window defined above. In Table 1, we present all features used in this study, where e ∈ E, and Per(e) is the average percentage of the EEG for 5 s, as defined in Formula (1).

$$\mathbf{Per(e)} = \frac{1}{5}\sum_{t=1}^{5}\left(\frac{e_t}{\sum_{d\in E}(d_t)}\right) \qquad (1)$$

Table 1: Features

Feature Name	Corresponding Formula	Description
f1～f8	Ave(e)	Average Value of e ∈ E for 5 s
f9～f16	Var(e)	Variance of e ∈ E for 5 s
f17～f24	Max(e)	Maximum Value of e ∈ E for 5 s
f25～f32	Min(e)	Minimum Value of e ∈ E for 5 s
f33～f40	Per(e)	Average Value of Percentage of e ∈ E for 5 s

4 Experiment

4.1 Environment

The subjects taking part in the experiment were four students in their twenties. They drove for about 5 hours each, with short breaks. For the electroencephalograph, we used NeuroSky's Mind Wave Mobile, which can send EEG data to a computer, via Bluetooth. The EEG frequencies obtained by Mind Wave Mobile were divided into eight types as shown in Table 2. Figure 3 provides an illustration of a person wearing the Mind Wave Mobile and the data obtained from it. By attaching the electrodes to the forehead and the earlobe, we obtained a power spectral density for each type of EEG every second. The acquired data was saved as a CSV file, in which rows are the time series data, and columns represent different EEG types.

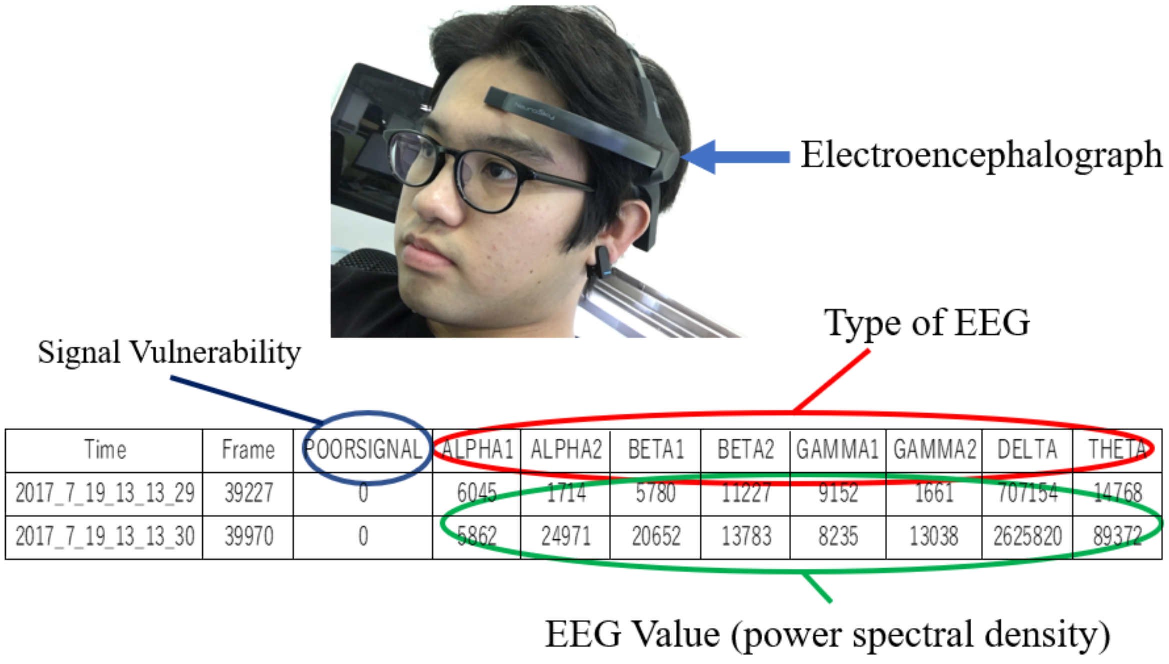
To obtain the driver's facial expression, we used Intel's RealSense SR300 as a 3D camera.

Figure 4 shows an example of a facial state captured by the camera, and the 3D coordinates of the acquired points. By using infrared ray data, the camera outputs the 3D coordinates of 78 points on a face at almost 30 fps. Besides these data, we recorded movies with RealSense to judge visually whether subjects were drowsy, for future studies, and we will label these coordinates with the phase 2 proposed method. We emphasize that, in this paper, we labelled these coordinates using EEG data only.

Table 2: EEG types

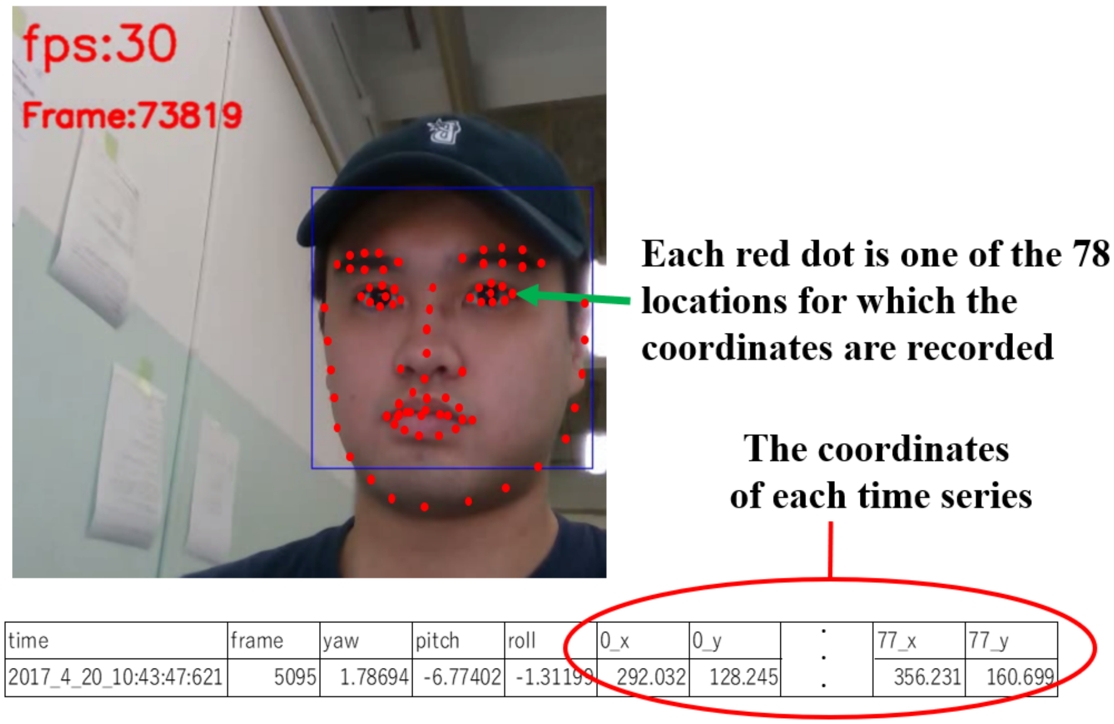
EEG Type	Frequency
γ2	41~50 Hz
γ1	31~40 Hz
β2	18~30 Hz
β1	13~17 Hz
α1	10~12 Hz
α2	8~9 Hz
θ	4~7 Hz
δ	1~3 Hz

We used a driving simulator, so as to observe the subjects' driving without putting them at risk when they became drowsy. Specifically, we reproduced a driving environment by projecting the PlayStation 2 Shutoko Battle (also known as "Tokyo Xtreme Racer" in the US, and "Tokyo Highway Challenge" in Europe) on a screen wit a projector. To simulate the driving environment more precisely, we employed Logicool's GT FORCE for the pedals, and a steering wheel, thereby allowing operation of the simulated cars with steering wheels and pedals, instead of gamepad controllers. Our equipment was arranged as shown in Figure 5, and the experiments were conducted in a dark room to encourage sleepiness in the driver.



Time	Frame	POORSIGNAL	ALPHA1	ALPHA2	BETA1	BETA2	GAMMA1	GAMMA2	DELTA	THETA
2017_7_19_13_13_29	39227	0	6045	1714	5780	11227	9152	1661	707154	14768
2017_7_19_13_13_30	39970	0	5862	24971	20652	13783	8235	13038	2625820	89372

Figure 3: Example of EEG data obtained from Mind Wave Mobile


fps:30
Frame:73819
Each red dot is one of the 78 locations for which the coordinates are recorded
The coordinates of each time series

time	frame	yaw	pitch	roll	0_x	0_y	.	77_x	77_y
2017_4_20_10:43:47:621	5095	1.78694	-6.77402	-1.31199	292.032	128.245	.	356.231	160.699

Figure 4: Example of the 78 points recorded in a facial expression

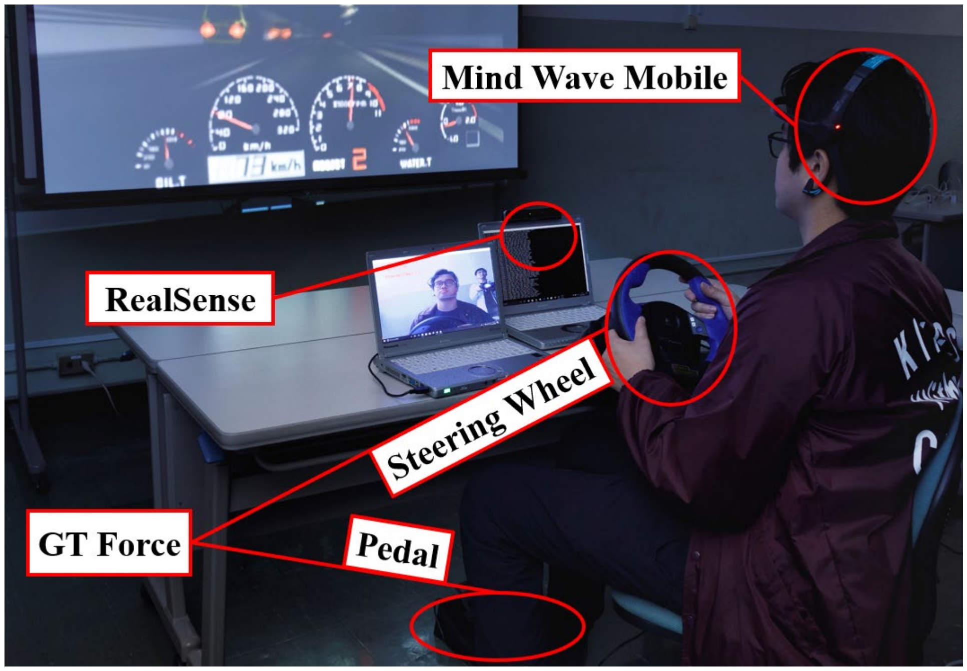


Figure 5: Experimental environment

4.2 Preliminary Experiment

To select an appropriate supervised learning model for classification of the EEG data, we compared the following classifiers: k-nearest neighbors (KNN), SVM, linear discriminant analysis, and a random forest. For SVM, we tested both a linear kernel, and a Gaussian kernel, with certain parameters. We verified the performance of each trained classifier by 10-fold cross validation, using data from a driver whose drowsy driving was observed for an hour. The breakdown of this dataset was 1,010 instances of drowsy state, and 1,649 instances of awake state. Figure 6 is a scatter diagram of Ave(α1) versus Ave(θ), calculated from the raw data, i.e., the recorded CSV file. As shown in Figure 6, one can observe a large difference between the portion where the positive data and the negative data overlap, and the maximum value of the negative data. To capture possible boundaries in these data more effectively, we computed the common logarithm of the raw data. Figure 7 is a scatter diagram of Ave(α1) versus Ave(θ), generated with common logarithm

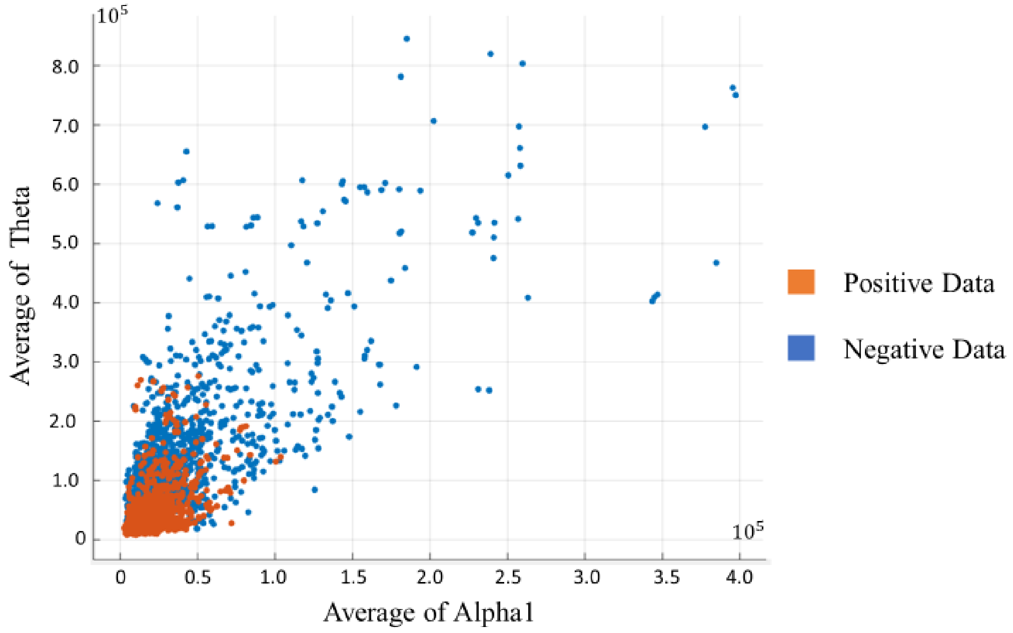


Figure 6: Scatter diagram of average values of raw data

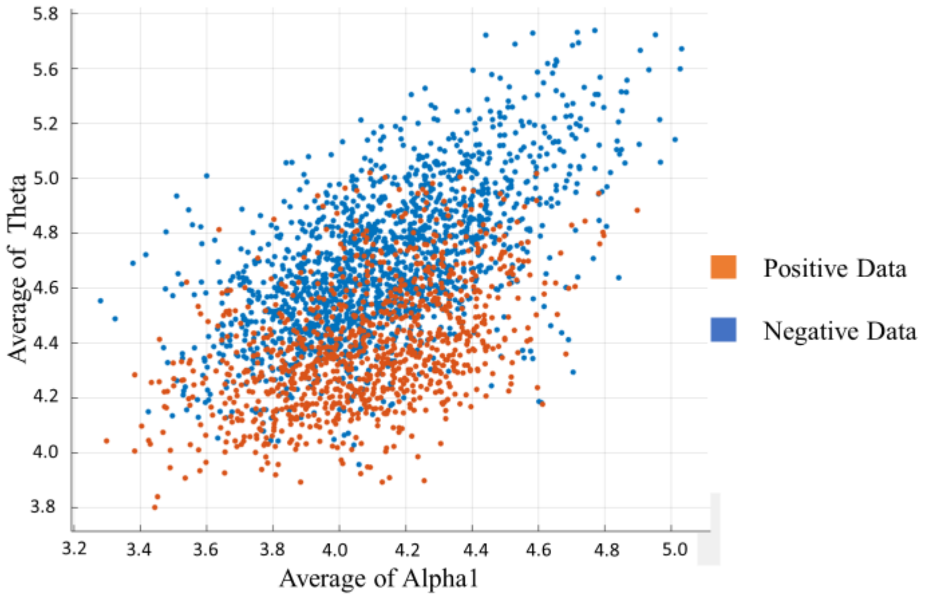


Figure 7: Scatter diagram of average values of the common logarithm of the data

values, and it can be seen that this technique allowed the boundary between the positive and negative data to be distinguished more easily.

As usual, we evaluated each learning model by its accuracy and F-value, which were defined as follows.

$$\mathbf{Accuracy} := \frac{\mathbf{TP+TN}}{\mathbf{TP+TN+FN+FP}} \tag{2}$$

$$\mathbf{Precision} := \frac{\mathbf{TP}}{\mathbf{TP+FP}} \tag{3}$$

$$\mathbf{Recall} := \frac{\mathbf{TP}}{\mathbf{TP+FN}} \tag{4}$$

$$\mathbf{F-value} := \frac{\mathbf{2Recall \cdot Precision}}{\mathbf{Recall+Precision}} \tag{5}$$

In these equations, TP, FP, TN, and FN are defined as follows:

- TP: Total count for which a drowsy state was judged as drowsy
- FP: Total count for which an awake state was wrongly judged as drowsy
- TN: Total count for which an awake state was judged as awake
- FN: Total count for which a drowsy state was wrongly judged as awake.

The accuracy and F-value of each learning model are shown in Table 3, and from these results, we can observe that KNN, the random forest, and the Gaussian kernel SVM were appropriate for classifying our dataset. Results obtained using the common logarithm were also better than those from the raw data. We therefore used features based on the common logarithm in what follows, except for $\mathrm{per}(e)$, and the reason that we did not use this method for $\mathrm{per}(e)$ is because if we computed its common logarithm, it became very close to the value for each type of EEG data (cf. Figure 7). Hence, we calculated $\mathrm{per}(e)$ directly from the raw data.

4.3 Evaluation of our Proposed Method

We evaluated the proposed method based on prediction results for unknown data with respect to the created classifiers. We created each classifier with a training set consisting of an hour of data for the driver that was used in the preliminary experiment, and then evaluated the classifiers based on prediction results for two hours of test data from the same driver. We used KNN, a Random Forest, and a Gaussian Kernel SVM, based on the results of our preliminary experiment.

Figure 8 shows the accuracy and F-value for each learning model, illustrating that SVM was best in terms of both accuracy and the F-value. Using SVM, we can classify unknown data from the same subject with high accuracy, and a high F-value. KNN achieved the best results in the preliminary experiment, whereas it performed worst in this experiment. In summary, it seems best to use the Gaussian Kernel SVM as the classifier for our proposed method, with a certain parameter.

4.4 Verification of the Influence of the Size of the Training Dataset

We reviewed whether the size of our dataset affected the prediction results. We divided our dataset, which contained three hours of data, into three equal parts, and used $k \in \{1,2\}$ hours as training data, and 3 - k hours as test data for each. Table 4 shows the accuracy and F-values for each setting. Based on these results, we can obtain reasonable accuracy and F-values with a single hour of training data, while two hours of training data led to a better outcome, in terms of F-value.

5 Conclusion

In this study we wanted to predict drowsy driving, while keeping the driving environment realistically comfortable. For this reason, we employed a method to predict drowsy driving using facial expressions only. From a driver's facial expression data acquired by a 3D camera, we are able to predict whether the driver is drowsy or not with machine learning. However, to classify the data for such purposes, we have to visually check the state of the driver – as drowsy or awake - and

Table 3: Results of the preliminary experiment

		Learning Model				
		KNN	Linear Discriminant Analysis	Random Forest	Gaussian Kernel SVM	Linear Kernel SVM
Raw Data	Accuracy	89.0%	80.2%	89.4%	85.5%	82.1%
	F-Value	91.6%	66.2%	90.3%	88.3%	84.3%
Common Logarithm Data	Accuracy	94.4%	87.3%	91.5%	93.1%	89.1%
	F-Value	95.3%	85.7%	91.3%	92.6%	85.8%

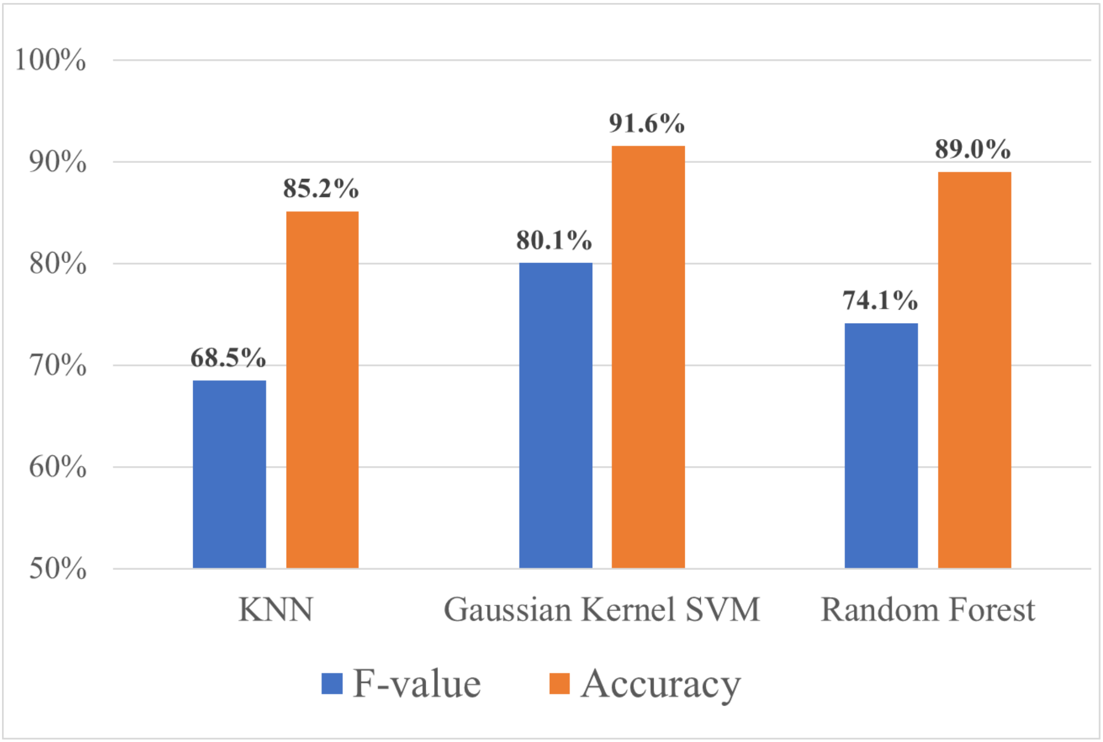


Figure 8: Evaluation results

Table 4: Results for different amounts of training data

Amount of Training Data	Accuracy	F-Value
One Hour	87.4%	73.7%
Two Hours	90.7%	82.1%

manually label the dataset, which takes considerable time and effort. For this reason, preparing enough training data has been a problem, and to resolve this, we proposed a new labeling method, using EEG data. We used the power spectral density for each type of EEG acquired from the electroencephalograph, and then from each of the acquired EEG types, we extracted features for each 5 s window, created with a sliding window.

In a preliminary experiment, we examined learning models suitable for our method, and found that SVM, KNN, and random forest performed best. In addition, to help clarify our results, we extracted features using the common logarithm of the raw data. Next, we evaluated our method on new data from the same subject, and in this evaluation, the Gaussian kernel SVM achieved the highest accuracy, at 91.6%, and an F-value of 80.1%, indicating that SVM technique was the most suitable for our method.

Labeling unknown data from the same person seems possible, although we have not verified its benefit against other test subjects. In the future, we hope to add drowsy driving data from other drivers, and confirm that our classifier system works as well with them. Moreover, for this study, we labeled facial expressions using EEG data, however in future work, we would review correlation between EEGs and facial expressions with the intention of confirming that use of facial expression alone would allow prediction of drowsy driving. So, while we need to test our results with more subjects and different age groups, as in this experiment we only used test subjects in their twenties, our current work indicates that, from the correlation, it might be possible to estimate brain state – and therefore predict drowsy driving - using detailed analysis of facial expression alone.

References

[1] American Academy of Sleep Medicine, "The AASM Manual for the Scoring of Sleep and Associated Events: Rules", *Terminology and Technical Specifications. Westchester: AASM*, 2007.

[2] Tarek Lajnef, Sahbi Chaibi, Perrine Ruby, Pierre-Emmanuel Aguera, Jean-Baptiste Eichenlaub, Mounir Samet, Abdennaceur Kachouri, and Karim Jerbi, "Learning Machines and Sleeping Brains: Automatic Sleep Stage Classification using Decision-Tree Multi-Class Support Vector Machines", *Journal of Neuroscience Methods*, 250:94-105, 2015.

[3] Boon-Giin Lee, Boon-Leng Lee, and Wan-Young Chung, "Wristband-Type Driver Vigilance Monitoring System using Smartwatch", *IEEE Sensors Journal*, 15(10):5624-5633, 2015.

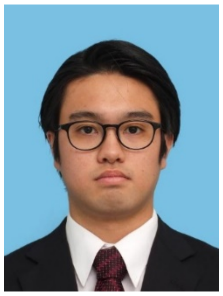
[4] "Neurosky Support Site; What are the Different EEG Band Frequencies?" http://support.neurosky.com/kb/science/eeg-band-frequencies, August 2009, (visited: 29th June 2018).

[5] Hiroyuki Nishiyama, Yusuke Saito, and Hayato Ohwada, "Machine Learning to Detect Drowsy Driving by Inductive Logic Programming using a 3D Camera", *Proc. of the 2016 International Symposium on Semiconductor Manufacturing Intelligence*, ID 42, 6 pp., 2016.

[6] Allan Rechtschaffen and Anthony Kales, *A Manual of Standardized Terminology, Techniques, and Scoring Systems for Sleep Stages of Human Subjects*, 1968.

[7] Tokyo Metropolitan Police Department Transit Authority, "About Traffic Death Accident in 2016", https://www.npa.go.jp/toukei/koutuu48/H28_jiko.pdf, February 2017, (visited: 21st November 2017).

[8] Madoka Yamazaki and Masato Sugiura, "The EEG of Adults and Elderly People", (in Japanese), *Clinical Neurophysiology*, 42(6):387-392, 2014.



Daichi Naito is a master's student at Tokyo University of Science. He received a B.S. from Tokyo University of Science in 2017. He majors in Industrial Administration and is interested in data science and machine learning. His e-mail address is 7417615@ed.tus.ac.jp.

Ryo Hatano is an Assistant Professor at Tokyo University of Science.

He obtained his Ph.D. from Japan Advanced Institute of Science and Technology in 2017.

His research interests span both mathematical logic and machine learning for artificial intelligence. Most of his work has been on developing reasoning system and linear algebraic semantics in dynamic epistemic logic. E-mail address: r-hatano@rs.tus.ac.jp.

Hiroyuki Nishiyama is a Professor at Tokyo University of Science. He obtained his Ph.D. from Tokyo University of Science in 2000. He is interested in the study of designing human-machine cooperative system, user interface, multi-agent system and network security based on Artificial Intelligence. He is a member of IEEE and ACM. His e-mail address is hiroyuki@rs.noda.tus.ac.jp.

LMS Performance Issues: A Case Study of D2L

Sourish Roy*, Carey Williamson*, and Rachel McLean*
University of Calgary, Calgary, AB, CANADA T2N 1N4

Abstract

In this article, we present a network traffic measurement study of the Learning Management System (LMS) used by the University of Calgary, which is called Desire2Learn (D2L). The motivation for our study comes from anecdotal reports of sluggish D2L performance, particularly for file uploads. Using a combination of active and passive network measurement techniques, we are able to identify the root causes of the poor D2L performance. The main issues identified are: (1) excessive HTTP redirections in our university's D2L setup; (2) non-negligible round-trip times (RTTs) to the server hosting the D2L content; and (3) default TCP window size settings that limit the maximum end-to-end throughput. We discuss these issues, and identify potential solutions, including improved protocol configurations, and the use of Web proxy caching. Our discrete-event simulation results indicate that a simple Web proxy cache at our university could reduce D2L request traffic by 50-70%, which would substantially improve user-perceived D2L performance.

Key Words: Learning management system (LMS), network traffic measurement, user-perceived performance, web-based systems, TCP, throughput, response time.

1 Introduction

A Learning Management System (LMS) is an important part of the IT infrastructure at many universities to help support their educational mandate. LMS technology augments classroom learning with support for online learning, by providing a single repository for all course content, and providing a unified environment for teaching-learning interactions between faculty and students. It provides instructors with tools to create, manage, and deliver course content, while providing students with access to content at their own pace, to support onsite, remote, or asynchronous learning. An LMS also allows instructors to check student participation and engagement in the course curriculum, and assess student performance on assignments, quizzes, and exams.

The concept of e-learning has been around for many years, but it is only relatively recently that the deployment of Web-based LMS software has become ubiquitous at educational institutions. Commercial LMS solutions today include Blackboard, Canvas, D2L, and others, as well as open-source solutions such as Moodle. Prior to Internet deployment, most LMS solutions were closed, proprietary, or custom systems within individual institutions.

One popular LMS is Brightspace by D2L (Desire2Learn), which we refer to as "D2L" in this paper [7]. D2L is a Canadian company headquartered in Kitchener, Ontario, Canada. They have more than 800 employees around the world, including Canada, Australia, Brazil, Singapore, UK, and the USA. D2L was started by John Baker in 1999 as a system to manage courses and student learning. A particular feature in D2L is the tracking and modeling of student learning activities, and making this information available to instructors. D2L also facilitates the uploading, grading, and return of assignments to students with appropriate feedback.

In 2014, the University of Calgary selected D2L as its new LMS as a replacement for Blackboard. Since then, every instructor and student has been given access to D2L, and more and more courses are now available within D2L. There are tens of thousands of on-campus users who use D2L each day to create/view course content, record/watch lectures, and enter/view grades. These LMS activities generate a lot of traffic on our campus network, using thousands of TCP connections, from many IP addresses, and multiple heterogeneous devices.

Knowledge of the D2L traffic patterns can help us understand its impact on the learning environment at the University of Calgary. Thus, the primary motivation for our work is a workload characterization study of D2L usage on our campus [17]. A secondary motivation for our work is user-perceived D2L performance. Specifically, anecdotal reports from faculty and students at the University of Calgary indicate that D2L is "slow". This problem has existed since 2014, but has not yet been resolved.

One underlying reason for the sluggishness of D2L is that the content is hosted remotely in Ontario, approximately 3200 km from Calgary, and thus has a non-negligible network round-trip time (RTT) for access. Indeed, our network measurements confirm that this network latency is an important contributing factor. However, we also find other technical issues with the D2L configuration that hamper its user-perceived performance. For example, file uploads for content producers (i.e., faculty/staff) are much slower than file downloads for content consumers (i.e., students).

An analysis of D2L traffic can provide insights into the

* Department of Computer Science, Email: {sourish.roy, cwill, rarmclea}@ucalgary.ca.

reasons for its slow performance. In computer networking, traffic measurement and analysis are crucial to the design, operation, and maintenance of local and wide-area networks. This area of research [11] is used extensively in both academia and industry. By collecting network traffic measurement data, we can assess the usage of a network, develop improved communication protocols, and improve the design of future networks. For example, one common technique is to use content caching in a network, to reduce traffic volumes and lower the network latency for popular content. As part of our work, we explore the applicability of content caching on campus to reduce requests for remote D2L content.

The rest of this paper is organized as follows. Section 2 provides some background on network traffic measurement, and prior research literature. Section 3 describes the research methods used for our work. Section 4 presents the results from our study, focusing on HTTP redirection, network latency, and TCP throughput. Section 5 presents more recent measurement results to further investigate D2L performance, including caching. Section 6 concludes the paper.

2 Background and Related Work

Network traffic measurement is a well-established technique to analyze network usage and understand network application performance. Over the years, a lot of prior research has focused on the characterization of Internet traffic, starting from the early 1990's [9], and continuing to the present [11] (e.g., Web traffic [3, 6, 13, 15], peer-to-peer file sharing [4], YouTube [10, 12], online social networks [5], Netflix [1, 14]).

Network traffic measurement studies such as these are useful not only for workload characterization, but also for network troubleshooting, protocol debugging, and performance evaluation. Below, we discuss several influential prior works on Web-based and education-oriented systems, which provide background context for our own work.

In 1996, Arlitt, et al. [3] characterized Web server workloads by analyzing access logs recorded at Web servers. These logs record requests for Web site URLs, including time of request, client IP address, content accessed, and document size. They used 6 different data sets in their study: three from universities, two from research organizations, and one from a commercial Internet provider. They identified common workload characteristics, such as Zipf-like object popularities and heavy-tailed file size distributions. In addition to the broad understanding of Web server workloads, this paper also discussed the importance of Web object caching. Based on their findings, they proposed improved Web caching systems with frequency-based cache management policies.

In 1999, Breslau, et al. [6] conducted a detailed study of Web caching performance. In particular, they identified the presence of power-law structures in Web object referencing, which is often referred to as a Zipf or a Zipf-like distribution. They used mathematical models to relate the Zipf characteristics to the effectiveness of Web caching. Their study provides guidelines on the achievable performance for Web caches in the presence of different workload characteristics.

In 2001, Almeida, et al. [2] analyzed server log data for educational media servers at two major US universities (University of Wisconsin, and University of California, Berkeley). Their paper focused on the eTeach system and BIBS (Berkeley International Broadcasting System), which delivered high quality media content. Their study provides a benchmark against which future media server workloads can be compared.

Newton, et al. [15] conducted a long-term Web traffic measurement study to see how this traffic has changed over time. They used the TCP/IP packet headers (1999-2012) in packet traces collected on the Internet link for the University of North Carolina. They performed an in-depth analysis of HTTP request sizes and responses, identifying growth in the size and complexity of Web pages, and increased use of cookies.

Two common themes throughout these prior works are the importance of knowing the workload characteristics for a given network application, and the importance of caching to improve the performance of any network-based system. Our paper builds upon the methods and insights from these prior works, and focuses on the network performance of D2L, as an example of an LMS. We are particularly interested in a workload characterization of D2L, identifying its performance bottlenecks, and exploring the use of local content caching to improve D2L performance.

3 Methodology

Network traffic measurement refers to a set of well-established techniques to collect and analyze empirical data from an operational network [11, 20]. In general, these techniques can be classified based on the type of network monitor used (e.g., hardware or software), where the network monitor is placed (e.g., edge or core), and by the data collection approach (e.g., passive or active). In our work, we use a combination of passive and active measurements on our edge network.

In passive network measurements, data is gathered by listening to ambient network traffic, without generating any additional traffic that might interfere with this flow, or affect the measurements themselves. In our study, we use specialized networking hardware to collect information about all campus-level traffic on our edge network. Our network monitor records information about the inbound/outbound network traffic passing through the university's edge routers. This collection takes place through a mirrored stream of all packet-level Internet traffic entering/leaving the University of Calgary network.

Our 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. 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.

Bro is an open-source framework for network analysis and security [8, 16]. In our work, the Bro logging system monitors all packet-level network activities, and produces connection-level logs summarizing all the traffic. 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/outbound data), duration, and termination state. The HTTP log helps us identify the source/destination IPs, HTTP methods, hosts, URIs, referer URLs, and user agents. Finally, the SSL logs show us HTTPS connections, with fields like timestamps, TLS/SSL encryption methods, plus source/destination ports.

Bro collects and generates logs on an hourly basis, which we aggregate together to provide a semester-long view of D2L traffic. We collect and analyze data from the HTTP, SSL, and connection logs to produce the results reported in this paper.

In addition to the Endace/Bro data collection described above, we also use Wireshark [21] to collect packet-level details on several D2L test sessions from our own desktop computers. Wireshark captures packets in real time, and displays them in a human-readable format. Using Wireshark, we can explore the details of D2L interactions for our own test sessions.

Unlike passive approaches, active measurements generate extra packets on the network as part of their data collection process. These can be used to measure the time taken to reach a target destination, the capacity available for a network path, or the response time for an application. Since this category of measurement generates additional traffic, we have performed active measurements selectively, using basic active measurement tools like ping and traceroute that have minimal impact on the network. Where appropriate, we have also generated some small test sessions in D2L, in order to assess the impacts of different file sizes, browsers, and configuration settings. These sessions constitute a very small proportion of the overall D2L traffic observed.

4 Measurement Results

4.1 D2L Traffic Overview

Figure 1 provides a high-level overview of the D2L traffic observed on our campus network during the Winter 2016 semester (January-April 2016). Note that D2L traffic occurs over both HTTP (for initiation/termination of D2L sessions) and HTTPS (for actual D2L interactions), and that there is a strong correlation between the two types of traffic.

Our measurements illustrate the traffic volume, data volume, response time, and throughput for all D2L users during the Winter 2016 semester. When lectures began on January 11, the D2L traffic increased. The D2L traffic pattern varies throughout the semester, with a dip during Reading Week break in February, and a sharp decline after final exams in April.

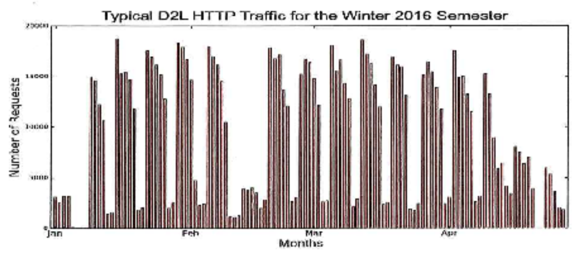
The D2L traffic shows strong daily and weekly patterns, with weekday traffic far exceeding that on weekends. On a typical weekday, we observe about 16,000 HTTP requests to D2L from the University of Calgary network, and about 500,000 HTTPS requests to D2L. The 30-fold difference between HTTP and HTTPS indicates that most D2L interactions occur via HTTPS. The HTTP traffic is primarily for session initiation/termination.

4.2 HTTP Redirection Issue

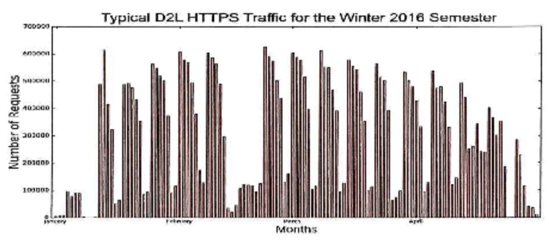
The first D2L performance issue that we have identified is related to how D2L is configured to operate within the University of Calgary IT infrastructure. In particular, session initiation involves user authentication. This step actually involves several HTTP redirections to the Central Authentication Service (CAS) at the University of Calgary. These interactions are complex, and add noticeable latency to the D2L experience.

Figure 2 shows a schematic illustration of a D2L test session that we conducted. This session lasts about 10 minutes, and involves several steps. First, the initial attempt to contact D2L via HTTP is redirected to use HTTPS instead. Second, the request is redirected from D2L to CAS at the University of Calgary for user authentication. Third, the Web browser uses multiple TCP connections in parallel to load the CAS login page (i.e., CSS file, logo, background, Javascript). Fourth, once the user logs in successfully, another HTTPS redirection occurs to re-connect with D2L. The Web browser then launches multiple TCP connections to retrieve the different components of the D2L landing page, including colour template, university logo, menu buttons, and course home page. The user is now ready to begin their D2L session. In this test session, the user browses several course pages, viewing slides from the course content, and uploading and downloading a few files. Finally, the user logs out.

During logout, another series of HTTP redirections occur.

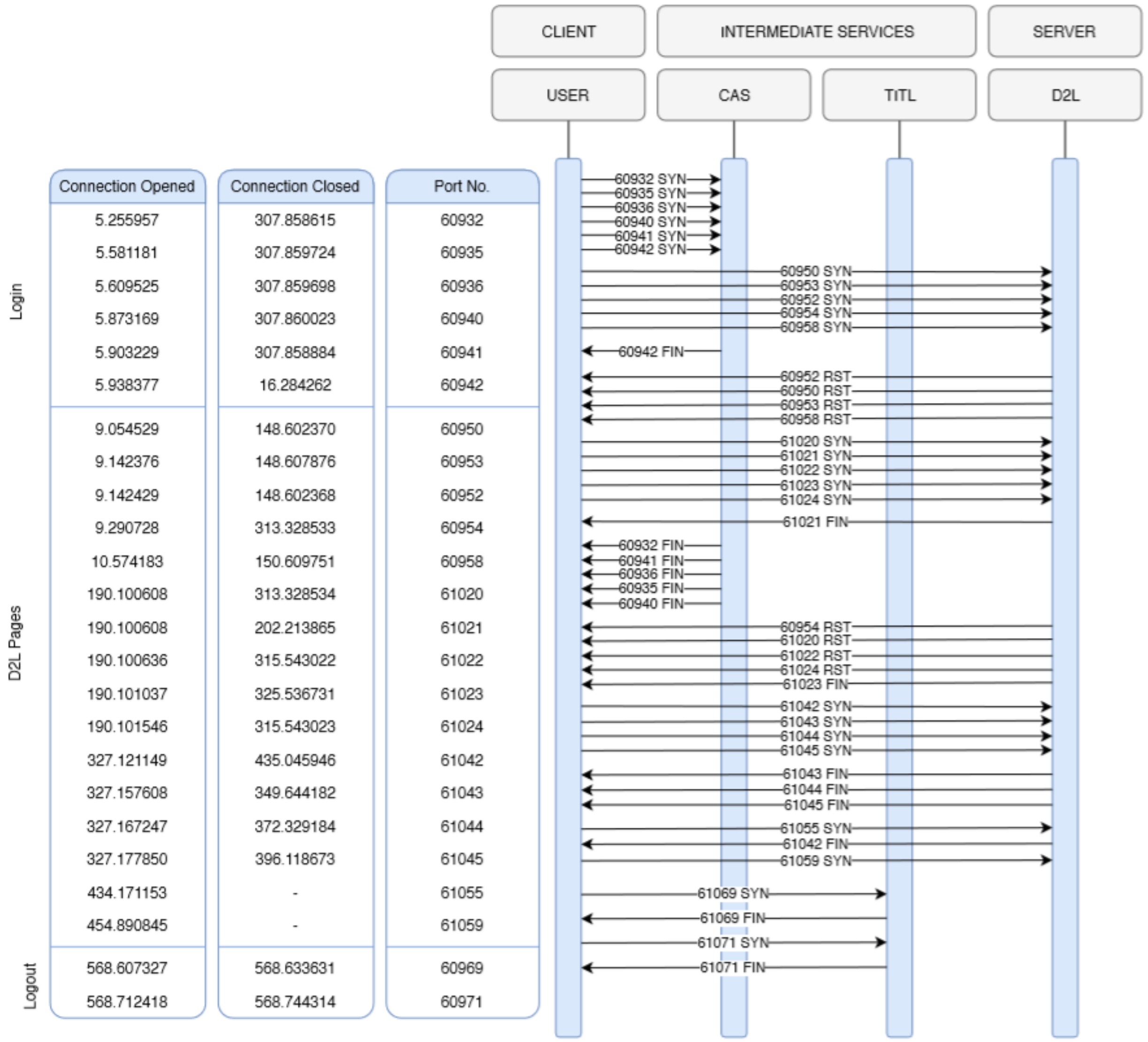


(a) HTTP requests per day



(b) HTTPS requests per day

Figure 1: D2L Traffic Profile for Winter 2016



	Connection Opened	Connection Closed	Port No.
Login	5.255957	307.858615	60932
	5.581181	307.859724	60935
	5.609525	307.859698	60936
	5.873169	307.860023	60940
	5.903229	307.858884	60941
	5.938377	16.284262	60942
D2L Pages	9.054529	148.602370	60950
	9.142376	148.607876	60953
	9.142429	148.602368	60952
	9.290728	313.328533	60954
	10.574183	150.609751	60958
	190.100608	313.328534	61020
	190.100608	202.213865	61021
	190.100636	315.543022	61022
	190.101037	325.536731	61023
	190.101546	315.543023	61024
	327.121149	435.045946	61042
	327.157608	349.644182	61043
	327.167247	372.329184	61044
	327.177850	396.118673	61045
	434.171153	-	61055
	454.890845	-	61059
Logout	568.607327	568.633631	60969
	568.712418	568.744314	60971

Figure 2: Example of D2L Browsing Session (IIS 10.0)

The first of these is from D2L to an e-learning server hosted by the Taylor Institute for Teaching and Learning (TITL) at the University of Calgary. Next, the Web browser uses parallel TCP connections to load the different components of the session logout page. Finally, there is a superfluous HTTP redirection from the TITL server to itself, to change the URL from "logout" to "logout/".

4.3 Network Latency Issue

The second D2L performance problem relates to how far away the D2L server is from the University of Calgary. In particular, the network round-trip-time (RTT) is about 40 ms, which is non-negligible. Figure 3 shows how an on-campus user accesses D2L. In this figure, the campus network is enclosed within a triangle, while the D2L hosted service in Ontario is indicated by the oval on the right. We are interested in characterizing the Internet path between the two.

Figure 4 shows the traceroute results, which indicate that the

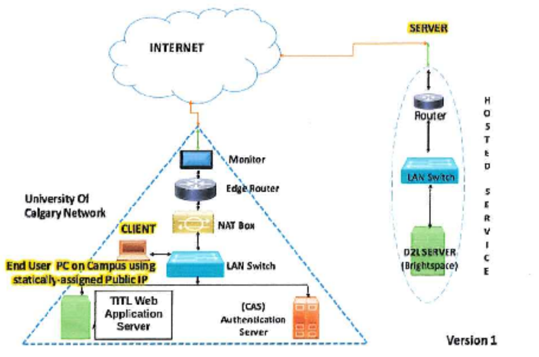

INTERNET
SERVER
Router
LAN Switch
D2LSERVER (Brightspace)
HOSTED SERVICE
Monitor
Edge Router
NAT Box
University Of Calgary Network
CLIENT
End User PC on Campus using statically-assigned Public IP
LAN Switch
TITL Web Application Server
(CAS) Authentication Server
Version 1

Figure 3: Network Path for D2L Users on Campus

D2L hosted service (i.e., desire2learn.ip4.torontointernet exchange.net) is located at a data center in Toronto. The network path has 17 hops with a total RTT of 37 ms.

A recurring theme in our study is the adverse impact of network latency on user-perceived performance in D2L. The performance of D2L is affected by these high RTT values. Users spend time waiting for responses from a distant data center in Toronto. This hinders the responsiveness of the D2L Web site, particularly when multiple HTTP/HTTPS redirections occur. Furthermore, the network bandwidth is not well utilized during TCP slow start, and D2L performance suffers.

4.4 TCP Throughput Issue

The third problem with D2L at our university is the TCP throughput, which is an important factor that affects network application performance. Using our empirical measurement data, we calculated the Average Data Rate (ADR) for D2L data transfers, which is the size of a transferred file divided by the elapsed time duration. This metric indicates the average throughput for D2L connections, in bits per second (bps).

Figure 5 shows a Log-Log Complementary Distribution (LLCD) plot of the ADR from some of our empirical data. 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. Note that these 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 handshaking, slow start effects, and any timeouts used for persistent connections.

There are two intriguing observations in Figure 5. First, the throughput values are very low (i.e., much lower than expected on CANARIE's fast national network). Second, the average throughput differs for uploads and downloads, almost by a factor of two.

```
$ traceroute d2l.ucalgary.ca
traceroute to d2l.ucalgary.ca (199.30.181.42), 30 hops max, 60 byte packets
1 deptNFSgate (172.17.10.1) 0.233 ms 0.217 ms 0.302 ms
2 * * *
3 10.58.48.1 (10.58.48.1) 0.367 ms 0.370 ms 0.363 ms
4 * * *
5 10.16.18.1 (10.16.18.1) 0.433 ms 0.404 ms 0.401 ms
6 10.16.18.4 (10.16.18.4) 0.302 ms 0.246 ms 0.237 ms
7 10.16.17.1 (10.16.17.1) 0.438 ms 0.403 ms 0.432 ms
8 10.59.226.26 (10.59.226.26) 0.334 ms 0.324 ms 0.333 ms
9 h74.gpvpn.ucalgary.ca (136.159.199.74) 3.296 ms 3.333 ms 3.471 ms
10 h66-244-233-17.bigpipeinc.com (66.244.233.17) 0.744 ms 0.633 ms 0.624 ms
11 h208-118-103-166.bigpipeinc.com (208.118.103.166) 0.880 ms 0.869 ms 0.836 ms
12 clgr2rtr2.canarie.ca (199.212.24.66) 0.721 ms 0.755 ms 0.726 ms
13 wnpg1rtr2.canarie.ca (205.189.33.199) 36.400 ms 36.180 ms 36.307 ms
14 canariecds.ip4.torontointernetxchange.net (206.108.34.170) 36.543 ms 36.514 ms 36.368 ms
15 desire2learn.ip4.torontointernetxchange.net (206.108.34.184) 36.668 ms 36.511 ms 36.484 ms
16 * * *
17 ucalgary.desire2learn.com (199.30.181.42) 36.727 ms 36.810 ms 36.770 ms
```

Figure 4: Traceroute Results for d2l.ucalgary.ca

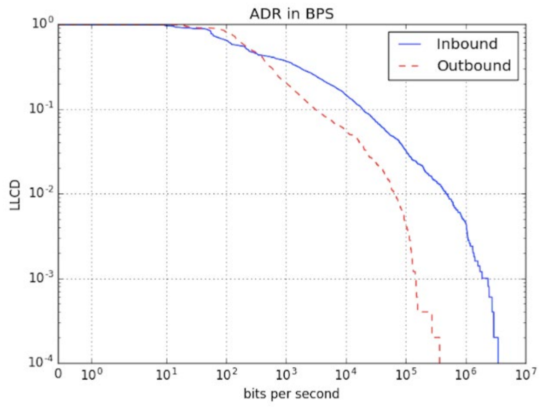


Figure 5: LLCD Plot of D2L Throughput

To further investigate this issue, we conducted some D2L test sessions involving uploads and downloads for a single 3.2 MB data file. Table 1 summarizes the results of our experiment, which confirms the asymmetric performance for uploads and downloads. The highest throughput achieved for downloads was 14 Mbps, while that for uploads was 7 Mbps. These results were consistent across all test scenarios considered.

We used Wireshark and some active measurements to learn more about the TCP version and settings used by D2L data transfers. Wireshark provides information such as TCP options, maximum segment size (MSS), slow start, window size, sequence number analysis, and others. OS fingerprinting allows

Table 1: TCP Throughput for D2L Transfers (3.2 MB)

Scenario	Device	OS	Download	Upload
On campus, wired	Desktop	Windows 8	14 Mbps	7 Mbps
On campus, wireless	Laptop	Mac OS X	14 Mbps	7 Mbps
Off campus, wireless	Laptop	Mac OS X	14 Mbps	7 Mbps

us to infer the operating system (Windows 2008 R2) and TCP version (Compound TCP [18, 19]) used by the D2L server, which is running Microsoft's Internet Information Server (IIS version 7.5)

Figure 6 and Figure 7 illustrate the dynamics observed on a large file upload. Figure 6 shows a TCP sequence number plot of how the data is transferred over time. This graph shows several irregularities in the data transfer process. Figure 7 illustrates the TCP receiver window size advertised by the D2L server during a file upload. The first observation is that the maximum advertised window size is 64 KB, which is the default socket buffer size for Compound TCP. This is a very small window size to use on networks with a large delay-bandwidth product, such as our scenario. The second observation is that the advertised window size fluctuates a lot, indicating that the server is slow in processing the arriving data packets. There are a half-dozen occurrences of small windows where the data transfer is inhibited. There is even a window stall event between 11 and 12 seconds, where the receiver window is almost zero (395 bytes, too small for the uploader to send another MSS).

These results demonstrate that the D2L data transfer performance is window-limited. Even if data transfers were perfect, with 64 KB of data exchanged every 40 ms, the maximum throughput would be 14 Mbps, which is what we observed in our download experiments. A larger window size of approximately 1 MB would be required to better exploit the network path between Calgary and Toronto.

Understanding why upload performance (7 Mbps) is worse than downloads (14 Mbps) requires even further investigation. To obtain insight into this problem, we conducted our own active measurement experiment on a D2L test session using special software called mitmproxy, which acts as a man-in-the-middle (mitm) proxy. This software intercepts the traffic between a client and a server, and can report all the HTTP/HTTPS traffic requests made by the user.

With mitmproxy in place, we can view the details of our D2L test sessions, including HTTP requests/responses, file names/sizes, and response times. These experiments showed that downloads use the GET method, while uploads use the POST method. However, the file uploads involve many POST requests, each with a small transfer size. Furthermore, D2L internally updates a file directory structure on uploads, as indicated by UpdateTreeBrowser in one of its URLs. In addition, there is an activity feed popup right after the new content is uploaded into D2L, as a notification for the user. These activities all increase the delay for D2L file uploads.

5 Additional Results

In May 2018, we repeated our D2L traffic measurements to see what, if anything, had changed from our earlier measurements in Winter 2016. Our additional results focus on D2L server performance, recent upgrades, and potential Web proxy cache performance.

5.1 D2L Server

The first observation from our May 2018 measurements is that the server-side infrastructure for D2L has changed since Winter 2016. In particular, the HTTP response headers now show that the D2L Web server is running IIS 10.0, rather than IIS 7.5. The server hardware and operating system have also been upgraded.

This upgrade has improved the performance for D2L file uploads, as shown in Figure 8. In this example of a 2.1 MB file upload, the TCP sequence number plot shows that the data transfer is completed in about 1.7 seconds, for an average throughput of 14 Mbps. Furthermore, Figure 9 shows that the server has no problem keeping up with the data, since the receiver advertised window for TCP is almost always 64 KB. To be specific, the initial advertised window is about 12 KB, but

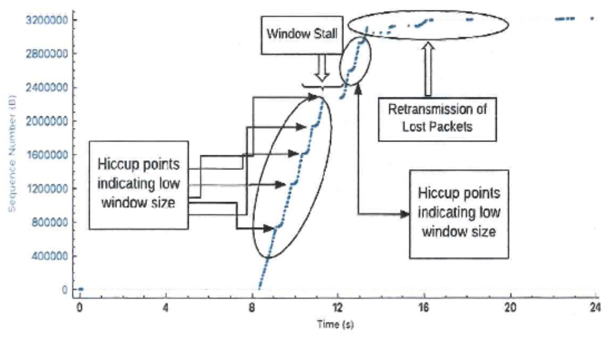


Figure 6: File Upload with D2L's IIS 10.0 Server

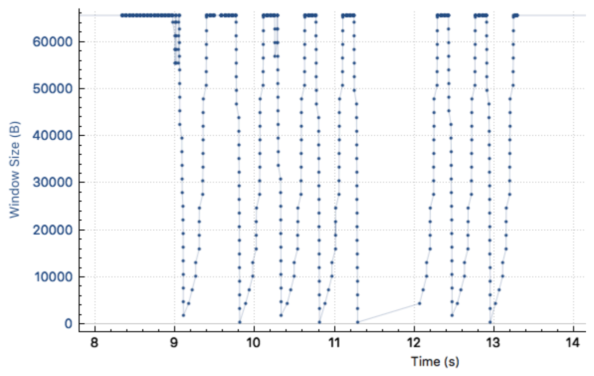


Figure 7: TCP Receive Window for D2L File Upload

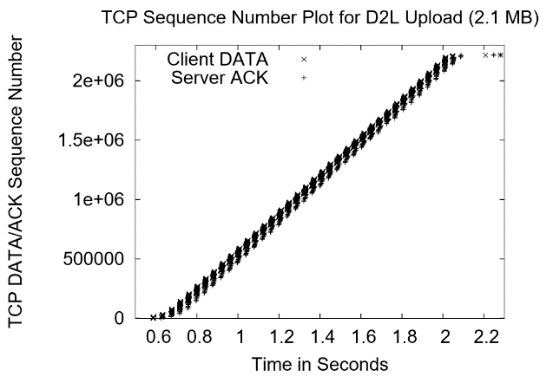


Figure 8: TCP Seqnum Plot for D2L File Upload

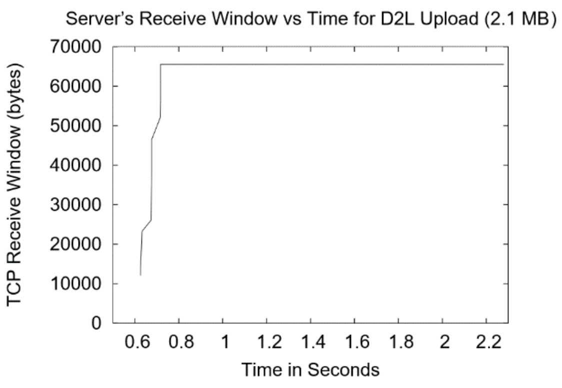


Figure 9: TCP Window during File Upload (IIS 10.00)

this window is dynamically resized by TCP until it reaches the maximum of 64 KB, and stays at that value for the rest of the transfer. The good news here is that uploads are now just as fast as downloads; the bad news is that uploads and downloads are both still window-limited. The TCP sequence number plot in Figure 8 still shows bursts of 64 KB at a time, with an ensuing wait of about 40 ms for the window's worth of acknowledgements to return. As a result, the average throughput never exceeds 14 Mbps, despite the multi-Gbps network path between the client and the server. TCP window scaling would be required to better utilize this network path.

5.2 D2L User Interface

On May 4, 2018, our university launched a new version of D2L with an improved user interface. The intent of the new release was to improve the user experience, with a new design layout, easier navigation, and mobile-friendly content. We collected detailed measurements both before and after the new release, in order to understand the differences. Importantly, this "upgrade" was only to the user interface, and did not address any of the fundamental network performance issues identified earlier.

While the new design is aesthetically pleasing, there are two new problems with the D2L deployment at our university. One problem is that the D2L home page is much larger and more complicated than it was before. Another problem is that D2L seems to have made almost all content uncacheable. We comment on these two issues next.

Table 2 provides a comparison of D2L Web pages both before and after the user interface upgrade. These are examples of pages in a D2L browsing session for a computer science student registered in the CPSC 413 course on Complexity Theory. The most striking observation is that the D2L home page, which all students visit by default upon login, has approximately doubled in size (KB) and complexity (number of objects). This results in slower Web browsing performance than before the upgrade. The other example pages are comparable to or smaller than they were before.

The second issue relates to content caching. Based on the analysis of HTTP response headers in Wireshark, most D2L content is now marked as uncacheable (i.e., "private, max-age=0"), unlike the previous version of D2L which allowed caching of static content (i.e., "public, max-age=3600"). Such content was typically cacheable for one hour (3600 seconds).

The underlying reason for making content uncacheable is not clear. We speculate that it is to enable full tracking of student learning activities, with all Web requests coming to the origin server. Another possibility is that it reflects the new General Data Protection Regulation (GDPR) policy enacted by the European Union in April 2016, which became effective on May 25, 2018. In simple terms, this privacy regulation states that no user data should be stored and used without the consent and knowledge of the user. However, one drawback of this regulation, if naively applied, is that it could compromise network performance.

The downside of uncacheable content is that the content must be retrieved repeatedly from the origin server far away. The most glaring example in our experiments was the user profile image (21 KB), which was downloaded 61 times during a 10-minute browsing session. If the content is hosted far away, then these repeated transfers will affect the user-perceived Web

Table 2: Comparison of D2L Web Page Complexity

Type of Web Page	**Before Update**		**After Update**	
	Objects	Size	Objects	Size
D2L Home Page	21	638 KB	45	1,239 KB
Course Home Page	27	466 KB	11	567 KB
Course Content Page	21	219 KB	17	173 KB
View Content Page	19	44 KB	16	52 KB

browsing performance.

As with our earlier work, we have shared these results with the University of Calgary Information Technologies (UCIT) team at our university. They are in touch with D2L to find better technical solutions for our D2L performance problems. One obvious solution would be to host all D2L content on campus, or provide a CDN node on campus that stores this private information locally, rather than in the cloud. Another solution might be to ensure that private information is always stored and accessed in encrypted form.

5.3 Web Caching

Our final result relates to the potential benefits of Web proxy caching to improve D2L performance at our university. We use trace-driven simulation for this analysis, with an arbitrarily chosen D2L workload trace from February 14, 2018. There were approximately 402,000 Web object transfers on that day.

Our discrete-event simulation models a simple Web proxy cache on campus, which stores and remembers previously seen static Web objects. Since the cache has a finite size, a cache replacement policy is used to manage the cache, and remove previous items when more space is needed to store a new item. We use this caching simulator to estimate the potential savings in D2L requests with a Web proxy cache.

We consider several different replacement policies to manage the cache. RAND (Random) removes a randomly-chosen object when more space is needed. FIFO (First In First Out) removes the oldest object in the cache. LRU (Least Recently Used) removes objects that have not been used recently. SIZE removes large objects, while keeping small objects. LFU (Least Frequently Used) removes objects that have not been used often. We consider two variants of the latter: In-Cache LFU (CLFU) only tracks the popularity of objects that are in the cache, resetting the popularity of an object to zero when it leaves the cache, while Global LFU (GLFU) always remembers the cumulative popularity of an object, whether it is still in the cache or not.

Figure 10 shows the results from our Web proxy cache simulations. On this graph, the horizontal axis shows the cache size in Megabytes (MB) on a logarithmic scale, while the vertical axis shows the Document Hit Ratio (DHR) on a linear scale. Higher values of DHR are better, since they represent more objects being retrieved from the local proxy cache on campus, and fewer requests going all the way to the D2L server.

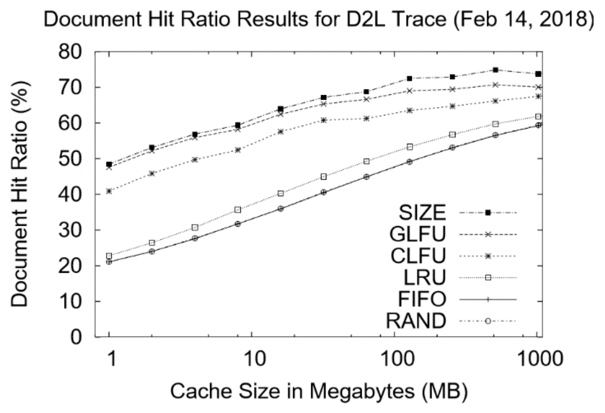


Figure 10: Simulation results for web proxy cache

In general, the results in Figure 10 show very good potential for Web proxy caching. Even a modest size cache (10 MB) can provide a hit ratio of over 50%, reducing by half the number of requests to the origin server. With a cache size of 1 GB, the DHR would be approximately 70%. Figure 10 also shows that there are notable performance differences among the replacement policies used to manage the cache. The best policy is SIZE, which focuses on caching a large number of small objects, such as the icons, sprites, and logos used on many of the D2L pages. The frequency-based policies also perform well, with GLFU always outperforming CLFU (as expected). The LRU policy is next best. FIFO and RAND are poor cache management policies, as expected.

The main takeaway message from these experiments is that for D2L, a very simple and modest-sized Web proxy cache on campus would be highly effective in reducing D2L requests to the remote server. This cache would improve the user-perceived latency for accessing D2L content, and reduce repeated transfers of the same content across the Internet. Furthermore, in addition to the on-demand caching described above, there should be very good potential for doing pre-fetching (i.e., prediction) of D2L content requests, based on the browsing patterns of students.

Two caveats apply to these Web proxy caching results. First, these results assume that all D2L objects are static Web content, and eligible for caching. This assumption may not hold for dynamic content, or for uncacheable requests that D2L uses to track student learning behavior. Second, our caching study ignores the issue of end-to-end encryption, which D2L uses to protect user privacy. Any practical caching solution for D2L, such as a Web proxy cache or a CDN node, would have to consider these two practical issues.

6 Conclusions

In this paper, we presented an empirical measurement study of the Desire2Learn (D2L) Learning Management System (LMS) adopted for use at the University of Calgary. The motivation for our study was to gain a better understanding of the system configuration, and its performance limitations.

While studying an LMS such as D2L is complex, there are three main technical issues that emerge from our study as root causes for the poor performance of D2L. The first issue is the excessive use of HTTP redirection at the University of Calgary to manage login/logout for D2L sessions. The second issue is the network RTT latency for D2L users in Calgary to access course content that is remotely hosted in Ontario. Finally, the TCP configuration on the D2L server has a maximum window size of 64 KB, which limits data transfer throughput.

The main conclusion from our study is that D2L is slow, and unnecessarily so. Fortunately, the observed performance problems are all fixable, as follows. First, we observed over one million HTTP redirects during the Winter 2016 term, which could be eliminated to minimize network round-trips and reduce server load. Second, there is a 40 ms RTT latency for University of Calgary users to access D2L content. Using a content delivery network (CDN), or placing a CDN node locally on campus, could greatly accelerate content delivery. Our simulation-based study suggests that a simple Web proxy cache, properly managed, could reduce D2L content requests by 50-70%. Finally, the TCP window size used by D2L is small, and does not scale dynamically based on the observed characteristics of the network path. An expanded TCP window size would solve this problem, improving throughput for both uploads and downloads. We hope that the insights from our study will improve future deployments of the D2L LMS, both at our university and elsewhere.

Acknowledgements

Financial support for this research was provided by NSERC. The authors thank D'Arcy Norman at TITL for his detailed knowledge of D2L, and the CATA 2018 reviewers for their feedback and suggestions on an earlier version of this paper.

References

[1] V. Adhikari, Y. Guo, F. Hao, V. Hilt, Z-L. Zhang, M. Varvello, and M. Steiner, "Measurement Study of Netflix, Hulu, and a Tale of Three CDNs", *IEEE/ACM Transactions on Networking*, 23(6):1984-1997, December 2015.

[2] J. Almeida, J. Krueger, D. Eager, and M. Vernon, "Analysis of Educational Media Server Workloads", *Proceedings of ACM NOSSDAV*, Port Jefferson, NY, pp. 21-30, January 2001.

[3] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE/ACM Trans. on Networking*, 5(5):631-645, Oct. 1997.

[4] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt, "A Comparative Analysis of Web and Peer-to-Peer Traffic", *Proceedings of WWW*, Beijing, China, pp. 287-296, April 2008.

[5] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing User Behavior in Online Social Networks", *Proceedings of ACM IMC*, Chicago, IL, pp. 49-62, November 2009.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", *Proceedings of IEEE INFOCOM*, New York, NY, pp. 126-134, March 1999.

[7] Brightspace By D2L, The Brightspace Cloud Content Delivery Network. https://community.brightspace.com/resources, Aug 2017.

[8] Bro, The Bro Network Security Monitor,https://www.bro.org, Jan 2018.

[9] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel, "Characteristics of Wide-area TCP/IP Conversations", *Proceedings of ACM SIGCOMM*, Zurich, Switzerland, pp. 101-112, August 1991.

[10] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User-Generated Content Video System", *Proceedings of ACM IMC*, San Diego, CA, pp. 1-14, November 2007.

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

[12] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic: A View from the Edge", *Proceedings of ACM IMC*, San Diego, CA, pp. 15-28, November 2007.

[13] F. Hernandez-Campos, K. Jeffay, and F. Smith, "Tracking the Evolution of Web Traffic: 1995-2003", *Proceedings of IEEE MASCOTS*, Orlando, FL, pp. 16-25, October 2003.

[14] M. Laterman, M. Arlitt, and C. Williamson, "A Campus-Level View of Netflix and Twitch: Characterization and Performance Implications", *Proceedings of SCS SPECTS*, Seattle, WA, pp. 15-28, July 2017.

[15] B. Newton, K. Jeffay, and J. Aikat, "The Continued Evolution of Web Traffic", *Proceedings of IEEE MASCOTS*, San Francisco, CA, pp. 80-89, August 2013.

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

[17] S. Roy, *Characterizing D2L Usage at the U of C*, MSc Thesis, University of Calgary, August 2017.

[18] K. Tan and J. Song, "Compound TCP: A Scalable and TCP-friendly Congestion Control for Highspeed Networks", *Proceedings of 4th International Workshop on Protocols for Fast Long-Distance Networks*, Nara, Japan, 8 pages, February 2006.

[19] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long-Distance Networks", *Proceedings of IEEE INFOCOM*, Barcelona, Spain, pp. 1217-1228, April 2009.

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

[21] Wireshark. www.wireshark.org, Jan 2018.

Sourish Roy completed his B.Tech. in Computer Science and Engineering from St. Thomas College of Engineering and Technology in Kolkata, India, and his M.Sc. in Computer Science from the University of Calgary in Calgary, Alberta, Canada. His interests include computer networks, Software Defined Networking, and big data analytics. He is currently employed as a Big Data Engineer at Index Exchange in Toronto, Ontario, Canada.

Carey Williamson is a Professor in the Department of Computer Science at the University of Calgary. He holds a BSc (Honours) in Computer Science from the University of Saskatchewan, and a PhD in Computer Science from Stanford University. His research interests include computer networks, internet protocols, network traffic measurement, simulation, and Web performance.

Rachel McLean is an undergraduate student at the University of Calgary. She is in her final year of study towards a BSc in Computer Science with a concentration in game design. In Spring 2018, she received a Canadian NSERC Undergraduate Student Research Award, allowing her to participate in network research under Dr. Carey Williamson.

An Evaluation of the Performance of Join Core and Join Indices Query Processing Methods

Reham M. Almutairi*, Mohammed Hamdi*
Southern Illinois University, Carbondale, IL, USA

Feng Yu†
Youngstown State University, Youngstown, OH, USA

Wen-Chi Hou*
Southern Illinois University, Carbondale, IL, USA

Abstract

Over the last few decades, much research has been devoted to designing and evaluating efficient join algorithms. The focus of this paper is a comparison of the two fastest join methods: Join indices and Join Core. Join indices generate index tables that contain tuples identifiers for matching tuples. Joins can be performed by scanning each input relation only once. On the other hand, Join Core is a data structure that stores join relationships to facilitate join query processing. With Join Core, join queries can be answered without having to perform costly join operations. Both methods have been implemented and we have performed extensive experiments on TPC-H benchmark datasets and queries. Results show that while both methods are much faster than conventional systems, such as MySQL, the Join Core approach is the fastest query processing method for the datasets studied.

Key Words: Query processing, join queries, equi-join, join indices, decision support systems.

1 Introduction

A database is a gathering of data, depicting the activities of at least one related association. Questions are posted by normal customers and applications constantly. Reacting to the asked for data from an enormous measure of information is a fundamental piece of database management systems [19].

The join operations consolidate data from no less than two relations [16], and are seemingly the most important operations in query processing. Since a client is holding up for the database to respond with an answer, the time for delivering results is exceptionally critical. Consequently, much research has been dedicated to designing efficient join algorithms over the most recent couple of decades [1-2, 6-7, 11, 13-14, 21-22, 17].

*Department of Computer Science. Email: reham.mutlaq@hotmail.com, {mhamdi, hou}@cs.siu.edu.
†Department of Computer Science and Information Systems. Email: fyu@ysu.edu.

The objective of this paper is to compare the performance of two of the most efficient join algorithms: Join Core and Join indices. Join indices [3, 4, 15, 18, 24] use indices to perform join operations. It has been reported that Join indices had better performance than traditional join methods. On the other hand, Join Core [8] had also reported superior performance to traditional systems, e.g., MySQL, as it can answer a wide variety of queries without performing joins, including equi-, outer-, and anti-joins. In this paper, we shall conduct extensive experiments to compare the performance of these two join methods.

The rest of the paper is organized as follows. In Section 2, we give a review of the traditional join algorithms. Section 3 explains in detail the multi-way Jive-join using Join Indices. Join Core is described in Section 4. Experimental results are reported in Section 5. Section 6 is the conclusions and the future work.

2 Traditional Join Algorithms

Traditional join algorithms generally can be categorized into three types: hash-based joins, sort-merge joins, and nested-loop joins. Here, we briefly review these methods:

2.1 Nested-Loop Joins

The nested-loop join is the simplest among the three traditional joins. Two relations that participate in the join operation are named: the outer relation (or the source relation) and the inner relation (or the target relation). Nested-loop joins algorithms are one-and-half passes because for each variation, it scans each tuple in the outer relation once and for the inner relation, it is repeatedly scanned. Nested-loop joins does not require that relations fit into memory, any size of a relation can be used in these joins [6]. Nested-loop join is most appropriate for smaller relations.

2.2 Sort-Merge Joins

The fundamental idea of the sort-merge join is sorting both relations on the join attribute using one of many sorting methods (e.g., n-way merge), and then searching for qualifying tuples from both relations by basically combining two relations [5].

2.3 Hash-Based Joins

The main idea behind the hash-based (partition) joins is to use hashing to partition the relations and match tuples in respective partitions.

Tuples in the outer relation, called the build relation, is partitioned into smaller files so that each partition can fit into memory. Tuples in the inner relation, called the probe relation, is partitioned using the same hash function. Then, each partition of the build relation is read into memory to build a hash table for tuples in the corresponding partition of the probe relation to find matches. Hash-based joins are generally very efficient in traditional methods [20].

3 Join Core

This section we briefly describe the Join Core technique [8].

3.1 Basis

Join Core is a data structure created to facilitate complex join queries processing. It is a gathering of equi -join relationships of tuples. The join relationships are stored in a set of tuples based on the equi-joins they satisfy. It intends to answer join queries without the need to perform expensive joins. It has been shown that it can answer a wide variety of join queries that include arbitrary (legitimate) combinations of equi-, semi-, outer, and anti-joins.

3.2 Structure

A join graph is commonly used to describe the equi-join relationships between pairs of relations.

Join Graph of a Database. Let D be a database with n relations R_1 , R_2 , …, R_n , and G (V, E) be the join graph of D, where V is a set of nodes that represents the set of relations in D, i.e., $V = \{R_1, R_2, R_3, ..., R_n\}$, and $E = \{(R_i, R_j) \mid R_i, R_j \in V, ij\}$, is a set of edges, in which each represents an equi-join relationship that has been defined between R_i and R_j, $i \neq j$.

Figure 1 shows the join graph of a database. Relations are connected by the (equi-)join edges. Each edge, e.g., <R1, R2>, specifies the equality requirements for a pair of tuples from the two relations, R1 and R2, must satisfy to generate a result tuple in the equi-join. Each join or edge is assigned a number as its ID. For example, the join between R1 and R2 or <R1, R2> is given an ID 4.

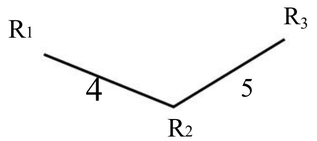


Figure 1: Join graph

Figure 2 shows an example of a database with the join graph shown in Figure 1. The edge between tuples, e.g., *S* and *W*, indicates that *S* and *W* have the same join attribute values for the join 4.

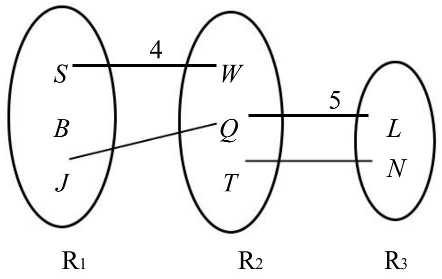


Figure 2: Matching of join attributes

[8] introduces the concept of trivial (equi-)joins with join predicates: R_i.key = R_i.key, $1 \leq i \leq 3$. Consequently, each tuple in relation R_i would automatically satisfy the trivial join i. Notice that all join edges in the join graph are non-trivial joins (or regular equi-joins), e.g. 4, 5. Join 1, 2, and 3 are reserved for trivial joins, which are not displayed in Figure 1.

Maximally Extended Match Tuple. Given a database $D = \{R_1, …, R_n\}$ and its join graph G, an extended match tuple (t_k, …, t_l), where $1 \leq k, ..., l \leq n$, $t_k \in R_k$, …, $t_l \in R_l$, and $R_k, …,R_l$ are all distinct relations, represents a set of tuples $\{t_k, …, t_l\}$ that generates a result tuple in $\{t_k\} \bowtie … \bowtie \{t_l\}$. A maximally extended match tuple (t_k, …, t_l), is an extended match tuple if no tuple t_m in R_m ($\notin \{R_k, …, R_l\}$) matches any of the tuples t_k, …, t_l in join attribute values.

Example. The join relationships of tuples are captured in the maximally extended match format [8], namely, (*S, W*), (*B*), (*J, Q, L*), (*T, N*). These join relationships are stored in different tables based on joins, both trivial and non -trivial ones, they satisfy. For example, (*S, W*) satisfies join 4 and trivial joins 1 and 2. Therefore, it is stored in a table called $J_{1,2,4}$. B is a tuple in R1, but it does not satisfy any non-trivial join, and thus is stored in J_1. These tables form the Join Core, as shown in Figure 3.

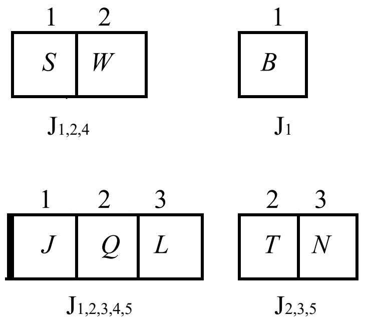


Figure 3: Join core structure

3.3 Join Core Construction

The Join Core can be constructed easily by first consecutively performing an outer -join for each join edge in the join graph, and then assigning the resulting tuples to Join Core tables based on the joins, both trivial and non-trivial ones, they have satisfied.

3.4 Answering Join Queries

Answering a join query is to look for tables whose names contain the join edge specified in the query without having to perform joins. For example, to answer a join query: R2 R3, the algorithm looks for join core tables whose names contain the numbers 2, 3 and 5. Only tables J 1,2,3,4,5 and J2,3,5 meet the search criteria, and they are returned as answers, after some simple processing, such as removing unwanted attributes. Consequently, join queries can be answered rapidly.

It has been shown that Join Core can be directly applied to queries with arbitrary legitimate combinations of equi-, semi-, outer- and anti-joins.

4 Jive-Join

This section presents the Jive-join technology. It explains the terminology and the basic structure of Jive-join algorithm.

4.1 Basis

In this algorithm, only one sequential passes through each input relation, one sequential passes through the join index, and two sequential passes through a temporary file that its size is half of the join index size are needed. To reduce the use of main memory, the algorithm stores the intermediate and final results vertically on disks. The Jive-join algorithm helps to join relations that are much larger than the main memory [11].

4.2 Structure

In Jive-join method, the join index has a tuple-id column for every input relation. It will be a set of tuple -ids (t_1, t_2, t_3, …, t_i) from relations R_1, R_2, R_3,…., R_i. The algorithm divides relations R_2,…R_r into ranges. Then, R_i tuple-ids will be divided into K $_i$ ranges, for r=2, …, i. Every tuple (t_2, …, t_r) from tuple -ids can belong to one of $Y=K_2K_3…K_r$ partitions. After that, the algorithm processes each partition separately [9].

We use an example of a database with two relations, Student and Activity, are shown in Figure 4.

The join index table, namely, J, contains tuple-ids for each relation for only matched tuples. Firstly, the algorithm divides the join index table into two partitions as shown in Figure 5. Every partition has an associated output file buffer and associated temporary file buffer. When these buffers are full, each of them will flush its contents to a corresponding output file and temporary file on the disk.

In the second step, the algorithm scans J and R_1 sequentially, and discovers the partition that this tuple belongs to, based on the R_2 tuple-ids in J. Then, it executes two operations: write the attribute for R_1 to the output file buffer for the partition, and write R_2 tuple-ids to a corresponding temporary file buffer for the partition, as shown in Figure 5.

Student	Activity
Smith	Swimming
Bloggs	Golf
John	Squash
Davis	Football
Mark	Tennis

R_1: Relation Student

Activity	Code
Swimming	084
Squash	182
Tennis	219
Golf	100

R_2: Relation Activity

Student tuple-id	Activity tuple-id
1	1
2	4
3	2
5	3

Join Index (J)

Figure 4: Database

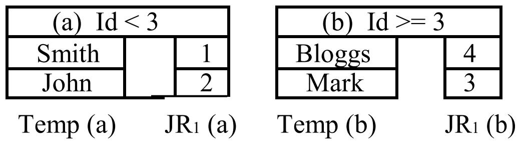


Figure 5: Partitioning J

In step 3, for each partition, the algorithm reads in and sorts the temporary file with duplicates eliminated, by keeping the original version of the temporary file. Next, it scans the relation R_2 by reading only tuples that are mentioned in the temporary file, and retrieves tuples in order. After that, tuples are written to the output file JR_2 in the order of the original temporary files, as shown in Figure 6.

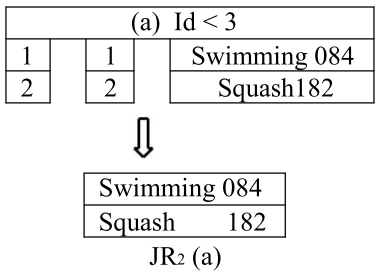


Figure 6(a): Output file (JR_2): partition (a)

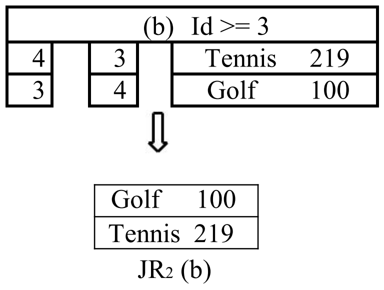


Figure 6(b): Output file (JR_2): partition (b)

A vertically partitioned data structure, which is called a transposed file, is used to store the final join result [11]. The partitions of each relation are linked together into a single file JR_i separately from other partitions of another relation such that, the first row in each of the files corresponds to the first join result tuple, the second rows to the second join result tuple, and so on. Noted that, attributes that are shared by more than one relation are placed arbitrarily in one of the vertical partitions, as shown in Figure 7.

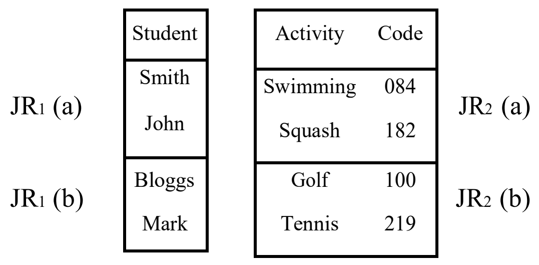


Figure 7: Final join result

4.3 Multi-Way Jive-Join

The structure of multi-way Jive-join is not much different from the case of the two joins. In multi-way Jive-join, the join index has a tuple -id column for every input relation. It will be a set of tuple-ids (t_1, t_2, t_3, …, t_i) from relations R_1, R_2, R_3,…., R_i. The algorithm divides relations R_2,…R_r into ranges. Then, R_i tuple-ids will be divided into K_i ranges, for r=2, …, i. Every tuple (t_2, …, t_r) from tuple-ides can belong to one of $Y=K_2K_3…K_r$ partitions. After that, the algorithm processes each partition separately [10].

We use an example of a database with three relations, Student, Course, Instructor and Room, are shown in Figure 8.

Student	Course
Smith1	101
Smith2	109
Davis1	102
Davis2	105
Davis3	106
Brown	102
Black	103
Frick	107

Student (R_1)

Course	Instructor
101	Green
102	Yellow
103	Green
104	White
105	Evans
106	Albert
107	Beige
108	Red
109	Grey

Course (R_2)

Instructor	Time
Green	3:00
Yellow	10:00
White	11:00
Evans	8:00
Albert	2:00
Red	1:00
Grey	12:00

Instructor (R_3)

Room	Course
C300	105
C301	102
C302	108
C303	109
C304	104
C305	106
C309	103

Room (R_4)

Figure 8: Database

In the second step, the algorithm scans J, in Table 1, and R_1 sequentially, and discovers the partition that this tuple belongs to, based on the R_2, R_3, and R_4 tuple-ids in J. Then, it executes two operations: write the attribute for R1 to the output file buffer for the partition, and write R_2, R_3, R_4 tuple-ids to a corresponding temporary file buffer for the partition [11], as shown in Figure 9.

Table 1: Join index(J)

Student	Course	Time	Room
2	9	8	4
3	4	3	5
4	2	2	2
5	5	4	1
6	6	5	6
7	2	2	2
8	3	1	8

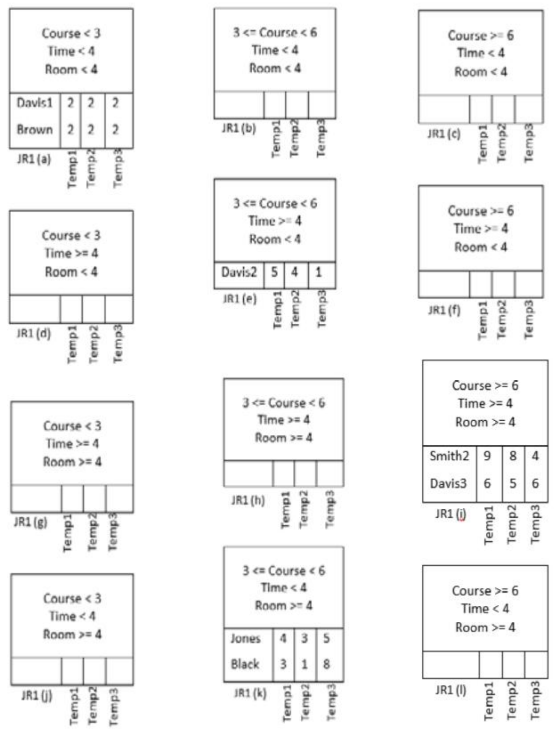


Figure 9: Partitioning J

After finishing step 2, the part of the output, namely JR_1, have been generated. The partitions of JR_1 will be linked together into a single file, which is JR_1, as shown in Figure 10. Also, as shown in Figure 11, we have been generating several temporary files that are used in step 3 to generate the other parts of the output: JR_2, JR_3, JR_4.

For each relation R_2, R_3, and R_4, tuple-ids in temporary files for all partitions for this relation are copied into one large array, namely H_1, H_2, H_3 corresponding to relations R_2, R_3, R_4 respectively. Then each array is sorted in ascending order with rejecting duplicates. Next, it scans each relation and the corresponding array, and retrieves tuples in order. After that, the tuples are written to the output files JR_2, JR_3, JR_4 in the order of the original temporary files, as shown in Figure 12.

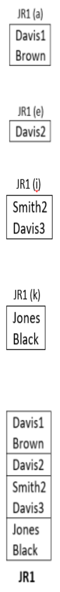


Figure 10: Partitions of JR1

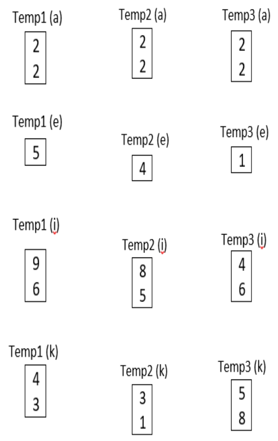


Figure 11: Temporary files

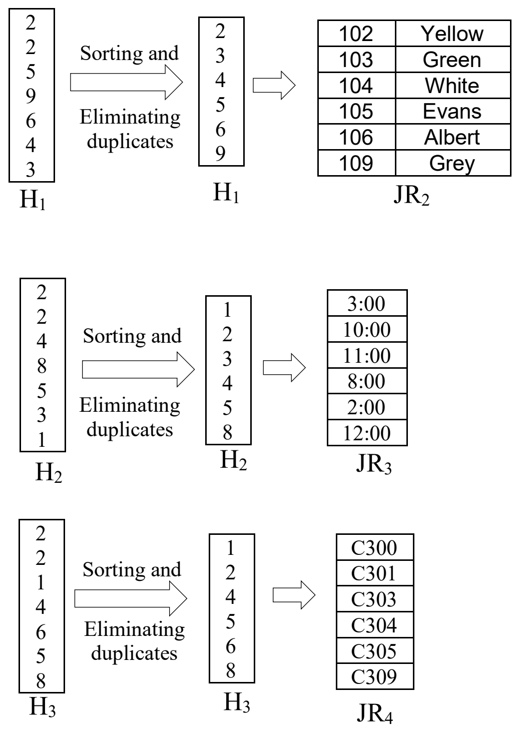
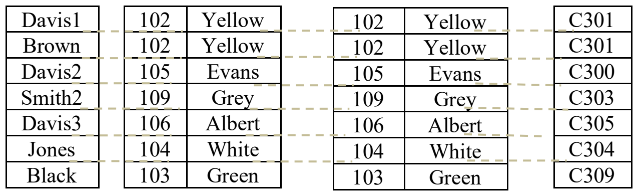


Figure 12: Output files JR2, JR3, and JR4

Vertically partitioned data structured, which is called a transposed file, is used to store the final join result [12]. The partitions of each relation are linked together into a single file JR_i separately from other partitions of another relation. In the way that, the first row in each of the files corresponds to the first join result tuple, the second rows correspond to the second join result tuple, and so on. Figure 13 shows the final output of this join. Noted that, attributes that are shared by more than one relation are placed arbitrarily in one of the vertical partitions. For example, the column for Course number is common in R_1 and R_2, which are Student relation and Course relation respectively, will be placed arbitrarily in only one vertical file: JR_1 or JR_2.



Davis1	102	Yellow	102	Yellow	C301
Brown	102	Yellow	102	Yellow	C301
Davis2	105	Evans	105	Evans	C300
Smith2	109	Grey	109	Grey	C303
Davis3	106	Albert	106	Albert	C305
Jones	104	White	104	White	C304
Black	103	Green	103	Green	C309

Figure 13: Final Result

5 Experimental Results

In this section, we report the experimental results on the multi- way Jive-join and Join Core by measuring their time and space consumptions. We have implemented multi-way Jive-join and Join Core using the JAVA programming language. Many factors, such as the number of CPUs, disks (and types of disks, magnetic or SSD), etc., can affect the performance of query processing. In this preliminary study, we will use only the simplest set up to see how the proposed method alone can improve query processing, leaving other performance improving factors to future work. All experiments are performed on a laptop computer with 1.80 GHz CPU, 4 GB RAM, and 930 GB hard drive.

5.1 Datasets

We generate TPC-H benchmark datasets with three sizes 1GB, 4GB, and 10GB, respectively. The datasets have eight separate tables. Figure 14 shows the TPC-H Schema. The arrows specify the direction of many-to-one relationships between tables.

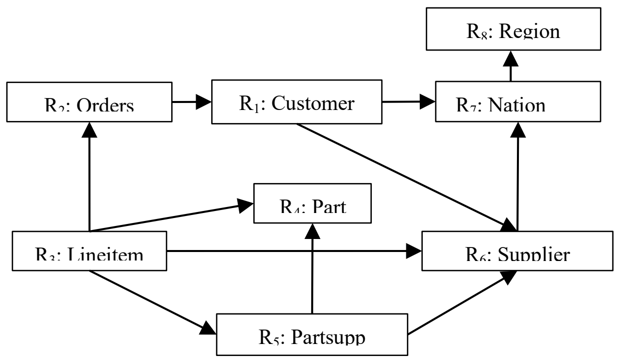


Figure 14: TPC-H Schema

5.2 Space Consumption

In the TPC-H benchmark dataset, Lineitem table is the largest one while Region is the smallest one. For the 1GB dataset, Line item has 6,001,215 tuples, Orders has 1,500,000 tuples, and Customer has 150,000 tuples. For the 4GB dataset, Lineitem table has 23,996,604 tuples, Orders has 6,000,000 tuples, and Customer has 600,000 tuples. For 10GB dataset, Lineitem has 42,297,504 tuples, Orders has 15,000,000 tuples,and Customer has 1500,000 tuples. However, in all three datasets, Nation has 25 tuples and Region has only five tuples.

Table 2 shows index tables' size for each query for our datasets. We can notice that even though the size of the dataset is large, the size of index tables are small because the index table includes tuples identifiers instead of tuples' values.

On the other hand, the full Join Core sizes, without applying any space reduction methods, are 4, 13.8, and 39.7 GB for the 1, 4, and 10 GB TPC-H datasets, respectively in Table 3. The larger sizes of the Join Cores are mainly due to the replications of tuples of smaller relations. In addition, there are several cycles, each of which introduces an additional alias relation in the graph. After removing Region, Nation, and Supplier, noted as "Reduced 1", is shown in Table 3. "Reduced 2", in Table 3, is the resulting graph after further removing the Customer relation from "Reduced 1" [8]. Consequently, Multi-way Jive-join method has better memory utilization than Join Core.

Table 2: Index tables' size

Query	1GB Dataset	4GB Dataset	10GB Dataset
Q_{12} / Q_4 : {R_2, R_3}	89.7 MB	118 MB	374 MB
Q_{14} / Q_{19}: {R_3, R_4}	87.3 MB	121 MB	367 MB
Q16: {R_4, R_5}	11 MB	53 MB	102 MB
Q3/ Q18: {R_1, R_2, R_3}	91.6 MB	394 MB	715 MB
Q_{10}: {R_1, R_2, R_3, R_7}	97 MB	406 MB	802 MB
Q_2: {R_4, R_5, R_6, R_7, R_8}	24.8 MB	92 MB	135 MB
Q_5: {R_1, R_2, R_3, R_6, R_7, R_8}	104.8 MB	472 MB	844 MB

Table 3: Join core size [8]

Join Core Size	Datasets		
	1GB	4GB	10GB
Full	4 GB	13.8 GB	39.7 GB
Reduced 1	2.3 GB	7.1 GB	20.1 GB
Reduced 2	1.7 GB	5.4 GB	15.8 GB

5.3 Time Consumption

We have applied test queries that come with TPC-H benchmark datasets. We have removed any grouping, aggregation, and ordering functions from the test queries so that we can focus only on the join operation. We included "distinct" for the queries to generate distinct result tuples.

A multi-way jive-join algorithm reads the tables from the disk into main memory to perform the join operation, then writes the result tuples to the disk. The time from the beginning to the time when the first result tuple is written to the disk is called the response time, while the time from the beginning until all result tuples are written to the disk is the elapsed time. For each query, we have measured the elapsed time for all the three datasets 1, 4, and 10 GB.

Table 4 shows the query processing time for Multi-way Jive-join and full Join Core tables. The first column identifies the ID of the TPC-H query followed by the relations that participate in this query. Relations are numbered as shown in the TPC-H dataset graph (Figure 8). The second column is the elapsed time in Join Core for 1, 4, and 10 GB datasets. The next two columns are Multi-way Jive-join elapsed time and MySQL elapsed time, respectively, for 1, 4, and 10 GB datasets. The final column is the result tuples for the queries. All the times are measured in seconds. We ignored queries that took more than 4 hours, which equals to 14,400 seconds, as showed by –'s in the table.

As we see in Table 4, the queries processed with MySQL took longer time than both Multi-way Jive-join and Join Core methods. For example, to process query 16 for 1 GB dataset using MySQL, it took 107.32 seconds. On the other hand, processing the same query took 47.030 seconds with Multi-way Jive-join, and took 0.812 seconds with Join Core, which both are much less than the processing time with MySQL.

Table 4: Time consumption

Query	Join Core Elapsed 1/4/10GB	Multi-way Jive Join Elapsed 1/4/10GB	MySQL Elapsed 1/4/10GB	Result Tuples
12: $\bowtie$ {R_2, R_3}	5.456	48.219	403.21	38,928
	22.409	161.532	817.02	155,585
	56.023	382.157	2,714	388,058
14: $\bowtie$ {R_3, R_4}	0.502	36.453	465.10	1,717
	1.865	203.187	1,422	6,718
	3.865	477.594	2,203	16,943
19: $\bowtie$ {R_3, R_4}	0.012	68.093	308.75	200
	0.041	277.704	971.14	864
	0.103	604.438	1,705	2,096
4: $\bowtie$ {R_2, R_3}	0.397	34.297	336.40	3,040
	1.518	124.547	911.38	11,889
	3.625	302.969	2,488	29,447
16: $\bowtie$ {R_4, R_5}	0.812	47.030	107.32	3,795
	3.005	269.782	446.21	15,208
	9.686	721.328	901.74	38,195
3: $\bowtie$ {R_1, R_2, R_3}	1.579	51.688	7,256	11,620
	8.016	155.563	10,342	45,395
	17.455	941.687	-	114,003
18: $\bowtie$ {R_1, R_2, R_3}	0.010	36.234	57.516	6
	0.012	154.156	163.65	11
	0.013	412.047	427.14	22
10: $\bowtie$ {R_1, R_2, R_3, R_7}	1.706	36.594	2,816	3,773
	5.667	248.625	5,516	14,800
	14.560	325.484	8,341	36,975
2: $\bowtie$ {R_4, R_5, R_6, R_7, R_8}	1.890	26.313	691.38	3,162
	7.005	401.875	2,810	12,723
	18.609	811.406	6,447	31,871
5: $\bowtie$ {R_1, R_2, R_3, R_6, R_7, R_8}	1.760	38.031	9,471	15,196
	6.809	218.469	-	60,798
	16.355	398.359	-	152,102

Additionally, from Table 4, we also notice that the query processing time with Multi-way Jive-join is larger than the time with Join Core algorithm. This is because Multi- way Jive-join needs to generate temporary files and output files JR_i. On the other hand, for the Join Core algorithm, there is no need to generate any intermediate results or perform expensive join operations. Therefore, queries are answered instantly. For example, for process query 2 for 1 GB dataset, it took 26.313 seconds in Multi- way Jive-join, but in the Join Core, it took only 1.890 seconds.

As observed from Table 4, the query processing time is generally increased as the size of datasets are increased for all the methods. For example, for query 19, it took 68.093, 277.704, and 604.438 seconds for the Multi-way Jive-join for 1, 4, and 10 GB datasets, respectively. This is because the larger datasets have a larger number of partitions, thus, generating larger output files and temporary files. Thus, larger datasets require more time to read and write.

It is pointed out that the sizes of the query results determine the query processing time for the Join Core method. For example, consider query 4 and query 18. Query 4 has one join and generates a larger number of result tuples. On the other hand, query 18 has two joins, including the join of the query 4, but generates a smaller number of result tuples. Consequently, the processing time for query 4 is larger than the processing time for query 18. As shown in Table 4, to process query 4, it took 0.397, 1.518, and 3.635 seconds for 1, 4, and 10 GB datasets, respectively, but it took 0.010, 0.012, and 0.013 seconds for 1, 4, 10 GB datasets respectively to process Query 18.

On the other hand, for Multi-way Jive -join, the complexity, the number of joins, of the query largely determines the query processing time, as illustrated by the elapsed time of queries 4 and 18. Even though the query 4 generates a larger number of results, it took less time than query 18 since it has only one join. Query 18 generates a smaller number of results, but it took longer to complete because it has two joins. As shown in the table, it took 34.297, 124.547, and 302.969 seconds to process the query 4 for 1, 4, and 10 GB datasets, respectively, and it took 36.234, 154.156, and 412.047 seconds to process query 18 for 1, 4, and 10 GB datasets, respectively. Hence, the complexity of the query, not the result size, determines the query processing time in Multi-way Jive-join.

Our experimental results show that Multi-way Jive-join has less space consumption than Join Core. Mostly, the main memory is used by more than one task. So, by consuming less memory for the join operation in Multi-way Jive-join, it helps other tasks to use the main memory at the same time. Also, our result shows that processing query with Multi-way Jive-join algorithm is faster than with MySQL. However, Multi-way Jive-join takes a longer time to process queries than Join Core strategy. This is because Multi -way Jive-join requires generating output files and temporary files before generating the final results. On the other hand, Join Core algorithm has instant response.

In addition, our result shows that processing query with Multi-way Jive-join algorithm is faster than with MySQL. However, Multi-way Jive-join takes a longer time to process queries than the Join Core. This is because Multi-way Jive-join requires generating output files and temporary files before generating the final results. On the other hand, Join Core algorithm does not perform joins or generate intermediate results; it also generates instant responses.

6 Conclusions

In this paper, we have implemented a version of Join Indices, which is the Multi-way Jive-join, to contrast its performance with the most recent join algorithm: Join Core. Join indices produce index tables that contain tuples identifiers for coordinated tuples. It scans each input relation just once, the join index once, and scans temporary files twice. on the other hand, Join Core is a data structure created to facilitate complex join queries promptly. With the Join Core, join queries can be dealt with rapidly without performing costly join operations. Also, no intermediate results should be made or recovered during the process. Therefore, join queries can be handled quickly. We have produced TPC-H benchmark datasets with three sizes 1GB, 4GB, and 10GB, as an experiment. Our test result demonstrates that Multi-way Jive-join method has better memory usage over Join Core. Moreover, processing queries with Multi-way Jive -join is quicker than with MySQL. In any case, our results demonstrate that Multi-way Jive-join method has longer processing time than Join Core technique.

References

[1] Mihaela A Bornea, Vasilis Vassalos, Yannis Kotidis, and Antonios Deligiannakis, "Double Index Nested-Loop Reactive Join for Result Rate Optimization", *IEEE 25th International Conference on Data Engineering, ICDE'09*, IEEE, pp. 481-492, 2009.

[2] Shumo Chu, Magdalena Balazinska, and Dan Suciu, "From Theory to Practice: Efficient Join Query Evaluation in a Parallel Database System", *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 63-78, 2015.

[3] Joseph Vinish D'silva, Bettina Kemme, Richard Grondin, and Evgueni Fadeitchev, "Powering Archive Store Query Processing via Join Indices", *Proc. 20th International Conference on Extending Data Base Technology (EDBT)*, pp. 644-655, March 21-24, 2017.

[4] Bipin C Desai, "Performance of a Composite Attribute and Join Index," *IEEE Transactions on Software Engineering*, 15(2):142-152, 1989.

[5] Jost Enderle, Matthias Hampel, and Thomas Seidl, "Joining Interval Data in Relational Databases," *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 683-694, 2004.

[6] Goetz Graefe, "New Algorithms for Join and Grouping Operations," *Computer Science-Research and DevelopmentI*, 27(1):3-27, 2012.

[7] Gunasekaran Hemalatha and ThanushkodiKeppana Gowder, "New Bucket Join Algorithm for Faster Join

Query Results.," *International Arab Journal of Information Technology (IAJIT)*, 12(6A):701-707, 2015.

[8] Mohammed Hamdi, Feng Yu, Sarah Alswedani, and Wen-Chi Hou, "Storing Join Relationships for Fast Join Query Processing," *International Conference on Database and Expert Systems Applications*, Springer, Cham, pp. 167-177, 2017.

[9] Ralph Kimball and Kevin Strehlo, "Why Decision Support Fails and How to Fix It," *Acm Sigmod Record*, 24(3):92-97, 1995.

[10] Ramon Lawrence, "Early Hash Join: A Configurable Algorithm for the Efficient and Early Production of Join Results," *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB Endowment, pp. 841-852, 2005.

[11] Zhe Li and Kenneth A. Ross, 'Fast Joins using Join Indices," *The VLDB Journal—The International Journal on Very Large Data Bases*, 8(1):1-24, 1999.

[12] Stefan Manegold, Peter A. Boncz, and Martin L. Kersten, "What Happens During a Join? Dissecting CPU and Memory Optimization Effects," *Proceedings of the 26th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., pp. 339-350, 2000.

[13] Stefan Manegold, Peter Boncz, Niels Nes, and Martin Kersten, "Cache-Conscious Radix-Decluster Projections," *Proceedings of the Thirtieth International Conference on Very Large Data BasesI*, VLDB Endowment, 30:684-695, 2004.

[14] Mohamed F. Mokbel, Ming Lu, and Walid G. Aref, "Hash-Merge Join: A Non-Blocking Join Algorithm for Producing Fast and Early Join Results," *Proceedings 20th International Conference on Data Engineering*, IEEE*,* pp. 251-262, 2004.

[15] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra, "Worst-Case Optimal Join Algorithms," *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ACM, pp. 37-48, 2012.

[16] Hung Q. Ngo, Christopher Ré, and Atri Rudra, "Skew Strikes Back: New Developments in the Theory of Join Algorithms," *ACM SIGMOD Record,* 42(4):5-16, 2014.

[17] Patrick O'Neil and Goetz Graefe, "Multi-Table Joins through Bitmapped Join Indices," *ACM SIGMOD Record,* 24(3):8-11, 1995.

[18] Raghu Ramakrishnan and Johannes Gehrke, *Database Management Systems*, (3rd ed). McGraw Hill, 2003.

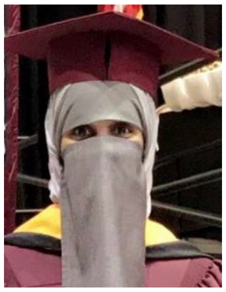
[19] Michael L. Rupley Jr., *Introduction to Query Processing and Optimization*, technical_reports/TR-20080105-1.pdf, Indiana University, 2008, International Journal of Computer Applications, 2011.

[20] Ronald Sam Lightstone, Guy Lohman, Ippokratis Pandis, Vijayshankar Raman, Richard Sidle, G. Attaluri, Naresh Chainani, Barber, and David Sharpe, "Memory-Efficient Hash Joins," *Proceedings of the VLDB Endowment,* 8(4):353-364, 2014.

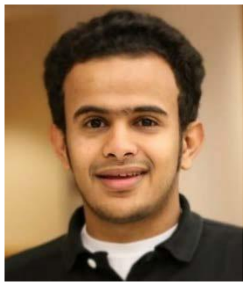
[21] Dong Keun Shin and Arnold Charles Meltzer, "A New Join Algorithm," *ACM SIGMOD Record,* 23(4): 13-20, 1994.

[22] TPC-H http://www.tpc.org:80/information/bench marks .asp, 2015.

[23] Patrick Valduriez, "Join Indices," *ACM Transactions on Database Systems (TODS),* 12(2): 218-246, 1987.



Reham Almutairi received her B.S in 2011 from University of Dammam in Saudi Arabia. In 2014, Reham got a scholarship from her government to study in the US. In 2017, she received her M.S. from Southern Illinois University in Carbondale. Studying in a different country with students from different cultures taught her that people have to be confident and continue to overcome their setbacks to adapt to a new culture! Her main research interests are databases, data mining, and big data. She can be contacted at reham.mutlaq@hotmail.com.



Mohammed Hamdi is an Assistant Professor in the Department of Computer Science at Najran University- KSA. He received his master's & PhD degrees in Computer Science from Southern Illinois University-Carbondale in 2013 and 2018. His main research interests are databases, query optimization, data mining, and big data.

Feng "George" Yu is an Associate Professor of Computer Science and Information Systems at Youngstown State University. He conducts a research lab called Data Lab focusing on Data-Oriented Sciences. His primary research interests include database management systems, big data, and cloud computing. He developed a new storage framework to speed up the data processing on a next-generation database called column-store database. He leads multiple cloud computing projects at YSU including the STEM Cloud. As a campus champion of NSF XSEDE, a powerful collection of supercomputing centers, Dr. Yu has organized at YSU multiple nationwide educational workshops on high-performance computing and big data.

Wen-Chi Hou received the MS and PhD degrees in Computer Science and Engineering from Case Western Reserve University, 1989, respectively. He is presently a Professor of Computer Science at Southern Illinois University at Carbondale. His interests include statistical databases, query optimization, concurrency control, XML databases

Novel Low Latency Load Shared Multicore Multicasting Schemes – An Extension to Core Migration

Bidyut Gupta*, Ashraf Alyanbaawi*
Southern Illinois University, Carbondale, USA

Nick Rahimi*
Southeast Missouri State University, Cape Girardeau, USA

Koushik Sinha*
Southern Illinois University, Carbondale, USA

Ziping Liu*
Southeast Missouri State University, Cape Girardeau, USA

Abstract

Shared tree multicast approach is preferred over source-based tree multicast because in the latter construction of minimum cost tree per source is needed unlike a single shared tree in the former approach. However, in shared tree approach a single core is needed to handle the entire traffic load resulting in degraded multicast performance. In addition, it also suffers from ‘single point failure’. In this paper, we have presented four novel and efficient schemes for load shared multicore multicasting for static networks. While the first two schemes emphasize load sharing, the last two schemes aim at achieving low latency multicasting along with load sharing for delay sensitive multicast applications. We have presented a unique approach for core migration, which uses two very important parameters, namely, *depth* of a core tree and *pseudo diameter* of a core. One noteworthy point from the viewpoint of fault tolerance is that the degree of fault- tolerance can be enhanced from covering single point-failure to any number of core failures.

Key Words: Core selection, pseudo diameter, multicore multicasting, load sharing, core migration.

1 Introduction

Multicast communication in the Internet [3] uses either the idea of source-based trees [1, 21, 26] or the idea of core-based trees (CBT) [2]. One major problem of the source-based-tree multicasting approach is that each router in the tree has to keep the pair information (source, group) and it is a one tree per source. In reality, the Internet is a complex heterogeneous environment and it potentially has to support many thousands of simultaneously active multicast groups, the majority of which are usually sparsely distributed. Therefore, this technique clearly does not scale well. It has been observed that shared tree-based approaches such as CBT [2, 17] and protocol independent multicasting – sparse mode (PIM-SM) [8] offer an improvement in scalability by a factor of the number of active sources.

In the core-based tree/shared tree approach [2] the tree branches are made up of other routers, the so-called non-core routers, which form a shortest path between a member-host’s directly attached router and the core. A core need not be topologically centered, because multicasts vary in nature and the geographical locations of the receivers (member-hosts) can be anywhere in the Internet; therefore, the structure of a core-based tree can vary [2] as well. CBT is attractive compared to source-based tree because of its key architectural features like scaling, tree creation, and unicast routing separation. The major concerns of shared tree approach are: core selection and core as a single point failure. Core selection [2, 10] is the problem of appropriate placement of a core or cores in a network for the purpose of improving the performance of the tree(s) constructed around these core(s).

In static networks, any core selection scheme depends on the knowledge of the entire network topology. It involves all routers in the network. There exist several important works [12, 21, 24-25] which take-into-account network topology while selecting a core. Maximum Path Count (MPC) core selection method [12] needs to know complete topology to calculate the shortest paths for all pairs. The nodes are then sorted in descending order of their path counts. The first nodes are selected to be the candidate cores. In Delay Variant Multicast Algorithm (DVMA) it is assumed that the complete topology is available at each node [21]. It works on the principle of k-shortest paths to the group of destination nodes concerned. If these paths do not satisfy a delay constraint, then it may find a longer path, which is a shortfall of DVMA. Optimal Cost Based Tree (OCBT) [23-25] approach calculates the cost of the tree rooted at each router in the network and selects the one which gives the lowest maximum delay over all other roots with lowest cost. It needs knowledge of the whole topology.

* Department of Computer Science.

1.1 Related Works

In the case of a single core-tree based multicast, the core has to handle all traffic load. It degrades the performance. To overcome the problem, shared tree multicast using multiple cores is the only solution. It distributes the total traffic load on the cores resulting in improved load balancing and thereby causing improved multicast performance. There exist in the literature some important contributions in this area of multicore-based multicasting [11, 23-25, 27]. The goal of these works is to achieve load balancing. In [11], Jia et al. have presented a Multiple Shared-Trees (MST) based approach to multicast routing. In their approach, the tree roots are formed into an anycast group so that the multicast packets can be anycast to the nearest node at one of the shared trees. However, load balancing is at the level of each source choosing the best core closest to it rather than attempting to utilize all the candidate cores simultaneously. This may lead to congestion in a core if multiple sources choose that core based on their shortest delay metric.

In [23], a unique tree consisting of multiple cores is maintained with one of the cores being the root. The objective of the work is to develop a loop free multi-core based tree by assigning level numbers to the cores and the nodes to join the tree to help maintain the tree structure. The cores need to coordinate with one another for their operations.

Zappala et.al. [27] have considered two different approaches for multicore load shared multicasting. The first one is senders-to-many scheme; it partitions the receivers of a group among the trees rooted at different cores so that each receiver is exactly on one core tree at a time. Therefore, one core tree spans some of the group members only. Even though it offers less routing state, yet it has the complex task to take care of newly arriving group members, i.e., partition them appropriately to the different core trees. In their second scheme, each core tree spans over the entire receiver group. To transmit data, different senders in a multicast group can use different trees with respect to the proximity of the source to the tree; it helps in balancing the traffic load and improve performance. The trees are maintained independently unlike the work in [23]. The core distribution method follows the hash-based scheme of [5]. Note that a different kind of load sharing (not load balancing) approach exists for splitting the load over Equal Cost Multipath (ECMP). Multicast traffic from different sources or from different sources and groups are load split across equal-cost paths provided such equal cost paths exist and are recognized as well [18].

The idea of core-based multicasting has been extended to allow migration of cores [4]. When the performance at an alternate core is 'reasonably' better than that at the current core, migration takes place to the alternate core. It helps in controlling multicast latency and load sharing.

It may be noted that there exist several important contributions [9, 19, 22], which have attempted to solve the problem of core selection with varied levels of complexity for delay and delay-variation constrained multicasting in static networks. These works aim mainly at reducing delay in communication to satisfy the delay specifications of applications; these works do not consider load sharing among multiple cores, which is a problem of immense importance in multicasting particularly when considering simultaneous presence of many different multicast sessions.

1.2 Our Contribution

In this paper, we have considered shared tree multicast with multiple cores. The motivation of the work is to improve multicast performance via load sharing. We prefer the term 'sharing' to 'balancing' because the objective of the work is to engage all cores whenever possible and more so because it is neither possible to know a priori the duration of different multicast sessions running at a given time, nor it is possible to know a priori the number of possible senders per multicast group. In this paper, we have adopted the multiple core-selection scheme reported in [7] to design the proposed various load-sharing algorithms. In this context, note that finding an optimal placement for multiple cores is more complex than for a single core. This problem can be viewed as given the maximum distance d from a node to a core, determines the smallest number of cores that satisfy this criterion. This minimum d-dominating set problem is NP-hard [6]. To avoid such complexity in the core selection process, in [7] a practical approach for multiple core selection has been considered with complexity $O(n^2)$.

A brief sketch of the proposed work is as follows. The core selection process in [20] is different from the ones in [5] and [27] and uses the idea of *pseudo diameter* [13-15]. Cores are determined statically using the routing information of all routers in the network. It is independent of any underlying unicast protocol active in the domain. The metric used in the determination is the one used in the unicast routing tables of the routers. The best core, i.e., the core with the minimum pseudo diameter is the primary core. We have proposed four novel load sharing schemes and a core migration scheme. The first two load sharing schemes neither partition the receivers among the different core-trees nor they build a unique tree consisting of multiple cores [23, 27]. In the first one, a primary core-based tree spans over all groups while each of the other core trees span over all members of an existing group only. However, a non-primary core tree may span over multiple groups based on the number of active groups and the number of selected cores. In the second scheme, a single core tree will never span over all groups. In our approaches, a sender does not decide which core to send packets to unlike in [27]; rather any source must send its first packet to the primary core and the primary core then decides if the sender needs to send the rest of its packets to any other core. We have discussed their relative merits.

The third and the fourth schemes have considered low latency load shared schemes for delay-sensitive applications. We have considered maximum path (*depth)* of a core-based tree as a parameter in addition to the *pseudo diameter* to achieve low latency load share in multicasting. To the best of our knowledge, there does not exist any work in this direction that considers the above two parameters together. The major difference between these two with the first two schemes is that in these two schemes a sender selects a core for multicasting its packets unlike in the first two; different senders belonging to the same group or not may select different cores. Therefore, from load sharing viewpoint the last two schemes may outperform the first two; however, these last two schemes have the extra overheads in that every core tree must span over all groups. Finally, we have presented a simple yet efficient core migration scheme which is based on both *depth* and *pseudo diameter*.

The organization of the paper is as follows. In Section 2, we state briefly the existing concept of pseudo diameter [13–15]. We also state the multicore selection scheme [7] used in this paper for a better understanding of the proposed load sharing schemes. In Section 3, we present the proposed multi-core group-based load-sharing multicast schemes. In Section 4, we present the two low latency schemes along with brief comparisons of all the four multicast schemes. In Section 5, we present the core migration scheme. Finally, Section 6 draws the conclusion.

2 Preliminaries

Two widely used unicast routing protocols are distance vector routing (DVR) and link state routing (LSR). In the former one, routers do not have the knowledge of network topology, whereas in the later routers have this knowledge. The concept of pseudo diameter is independent of the underlying unicast routing protocol. We denote the unicast routing table by UCT_i for some router r_i; it can be either the DVR table or the LSR table of the router depending on the unicast protocol used. Pseudo diameter of a router r_i denoted as $P_d(r_i)$ is defined as follows [13-15].

$P_d(r_i) = \max \{c_{i,j}\}$, where $c_{i,j} = \text{cost}(r_i, r_j)$,

$[1 \le j \le n, j \ne i]$ and $c_{i,j} \in UCT_i$ and

n = number of routers in the network

In words, pseudo diameter of router r_i denoted as $P_d(r_i)$, is the maximum value among the costs (as present in its routing table UCT_i) to reach from r_i all other routers in a network. The implication of pseudo-diameter is that any other router is reachable from router r_i within the distance (i.e., no. of hops/delay etc.) equal to the pseudo diameter $P_d(r_i)$ of router r_i,. It thus directly relates to the physical location of router r_i. Pseudo diameter is not the actual diameter of the network, because it depends on the location of router r_i in the network. So different routers in the network may have different values for their respective pseudo diameters. Therefore, pseudo diameter P_d is always less than or equal to the actual diameter of a network.

As an example, consider the network shown in Figure. 1. Without any loss of generality, we have considered DVR protocol as the underlying unicast protocol used in the network. The respective DVR tables of the routers appear in Figure. 2. Note that the diameter of the network is 90. From router A's table, its pseudo diameter is 90, which is equal to the network diameter; whereas for router C it is 70 as is seen from C's DVR table. It means that if C is the source of a communication, the maximum cost to reach any other router will be 70, which is less than the network diameter of 90. In this context, the following observations [15] are worth mentioning.

2.1 Multicore Selection Scheme

We briefly present a systematic approach [7] to select the cores to be used for multicore multicasting. Cores are selected statically using the routing information of all routers in the underlying unicast domain. This core selection is independent of any multicast group. For multicore multicast, each router r_i executes the following steps to select the required number of cores, m.

```
Multicore selection process

1. Router $r_i$ determines its $P_d(r_i)$ from its unicast
   routing table.
2. It broadcasts $P_d(r_i)$ in the network using pseudo
   diameter based broadcasting [15].
3. Router $r_i$ receives all $P_d(r_j)$ from every other router
   $r_j$, $1 \le j \le n, j \ne i$
4. Router $r_i$ creates a list of m routers out of all n
   routers, sorted in ascending order based on their $P_d$
   values.
   // the first one in the m-router list is the primary
   core.
```

Remark 1. *Every router creates identical sorted list of the m cores.*

Remark 2. *Router with minimum P_d is the primary core.*

Remark 3. *Message complexity of the core selection process is $O(n^2)$.*

Remark 4. *In the core selection process, a router does not need to know the entire topology.*

Observation 1. *For fault-tolerant single-core multicast, there are (m-1) number of redundant (stand by) cores.*

Observation 2a. *To guard against any possible core failures while using the m cores for multicasting, a total of 2m cores can be selected by the Multicore selection process.*

Observation 2b. *According to the positions in the sorted list all odd-numbered cores can be used for multicasting. Every odd-numbered core will have its standby as the even numbered core that immediately follows it in the sorted list of the cores; this standby core selection utilizes the proximity between these two cores from the viewpoint of their pseudo diameter values.*

2.2 An Example

Consider the example network of Figure.1. Primary core, in our approach, is a router that has the least pseudo diameter value compared to all other routers in the network. From the DVR tables (Figure 2) of the network, pseudo-diameter of all routers in the network can be obtained. Pseudo diameters of routers A, B, C, D, E, F, G, and H, are 90, 90, 70, 80, 60, 90, 80, and 80, respectively.

Each router broadcasts its P_d value to all other routers in the network. At the end of the broadcast, each router has the same sorted list of the routers based on their respective pseudo diameters. It is shown in Figure 3. Observe in the Figure that there is more than one router with the same P_d value. Routers D, G, and H have the same P_d value, viz., 80. If this situation arises, these routers are sorted in descending order of their router Id's (IP addresses). Without any loss of generality, we assume that router H has the highest router Id, followed by routers G and D respectively. So, router H has the priority to be selected as a possible candidate core over routers G and D. According to Figure 3, for the given network in Figure 1, router E is the primary core as it is the one with the least pseudo-diameter value, 60. For multicore multicasting, the first few routers from the list will be selected as needed. To incorporate core redundancy in case of single core multicast, router C can be chosen as the secondary core (standby core) as it has the next least pseudo-diameter. It will make the single-core scheme fault tolerant. More cores can act as standby for a larger degree of fault-tolerance.

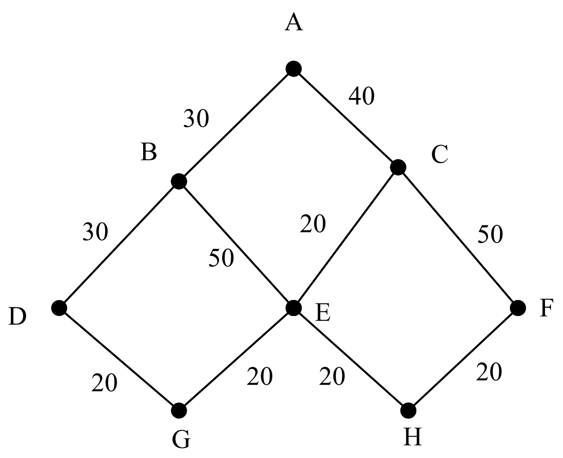


Figure 1: An 8-router network

A		
Dest.	Next	Delay
A	A	0
B	B	30
C	C	40
D	B	60
E	C	60
F	C	90
G	C	80
H	C	80

B		
Dest.	Next	Delay
A	A	30
B	B	0
C	A	70
D	D	30
E	E	50
F	E	90
G	D	50
H	E	70

C		
Dest.	Next	Delay
A	A	40
B	A	70
C	C	0
D	E	60
E	E	20
F	F	50
G	E	40
H	E	40

D		
Dest.	Next	Delay
A	B	60
B	B	30
C	G	60
D	D	0
E	G	40
F	G	80
G	G	20
H	G	60

E		
Dest.	Next	Delay
A	C	60
B	B	50
C	C	20
D	G	40
E	E	0
F	H	40
G	G	20
H	H	20

F		
Dest.	Next	Delay
A	C	90
B	H	90
C	C	50
D	H	80
E	H	40
F	F	0
G	H	60
H	H	20

G		
Dest.	Next	Delay
A	E	80
B	D	50
C	E	40
D	D	20
E	E	20
F	E	60
G	G	0
H	E	40

H		
Dest.	Next	Delay
A	E	80
B	E	70
C	E	40
D	E	60
E	E	20
F	F	20
G	E	40
H	H	0

Figure 2: DVR tables of the routers

Primary core →

r_i	$P_d(r_i)$
E	60
C	70
H	80
G	80
D	80
F	90
B	90
A	90

Figure 3: Sorted list based on the pseudo diameters

2.3 Performance

The multicore selection scheme needs only one broadcast independent of the number of cores to be used for multicasting. Therefore, the message complexity is $O(n^2)$, where n is the total number of nodes in the network. However, in this approach a router does not need to have the complete topological information. Note that to incorporate the effect of any changes in the network, e.g., router failure, this core selection process can run periodically. Besides, the core selection scheme is independent of any multicast groups unlike most existing works because it is a static core selection approach. A detailed discussion of the performance of the presented core selection method has appeared in [7]. The core selection scheme has been compared with some important existing core selection algorithms, mainly Maximum Path Count (MPC) [12], Delay Variant Multicast Algorithm (DVMA) [21], Minimum Average Distance (MAD) method, and OptTree method [12]. Results of the comparisons for randomly generated networks of different sizes are shown in the following Figures. The superiority of our approach from the viewpoint of message complexity to these other approaches is evident from the results. Note that unlike in OptTree method, in our approach a router does not need to have the complete topological information. Experimental setup has used NS2 simulator, BRITE topology generator, and Waxman's probability model for interconnection of the nodes.

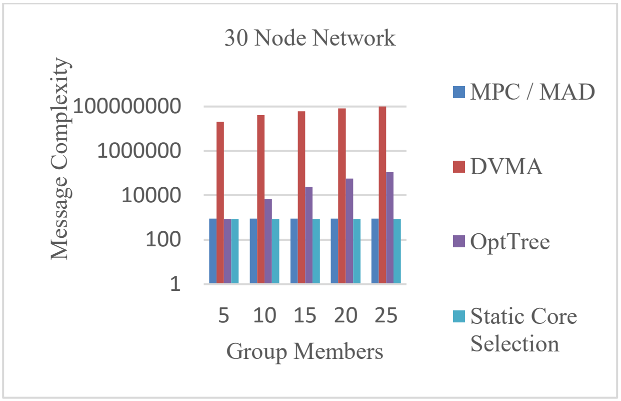


Figure 4: 30-router network

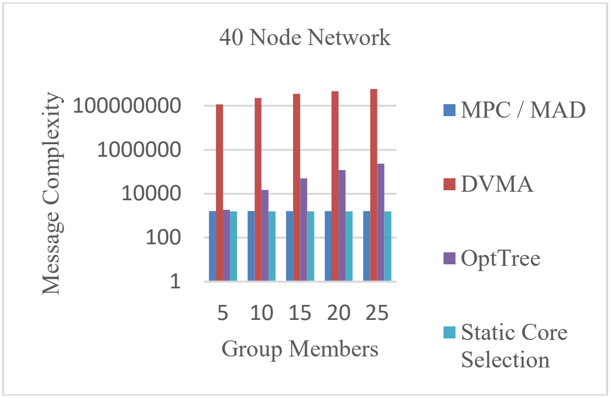


Figure 5: 40-router network

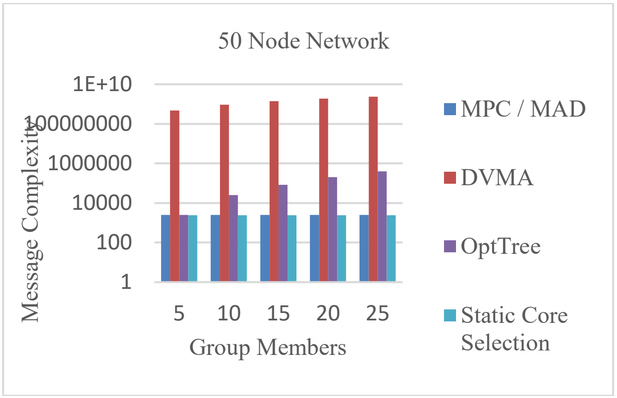


Figure 6: 50-router network

3 Group-Based Load Shared Multicore Multicoast

In this section, we present two simple and yet very effective group-based load-shared multicast schemes, GLSM-cast1 and GLSM-cast2; we have considered distribution of groups to different cores. Each core tree spans all members of a single existing group. It may also span over multiple groups based on the number of active groups and the number of selected cores. However, a single core tree will never span over all groups. Cores are determined statically using the routing information of all routers in the underlying unicast domain [7]. It is independent of any underlying unicast protocol active in the domain. The best core, i.e. the core with the minimum pseudo diameter, is called the primary core. In our approach, a sender does not decide which core to send packets to unlike in [27]; rather any source must send its first packet to the primary core and the primary core then decides if the sender needs to send the rest of its packets to any other core.

We use the following notations in the proposed schemes. We assume that there are **m** cores present in the system. C_0 denotes the primary core, and C_i denotes the $(i+1)^{th}$ core; S^{new} denotes a new sender for some group. In addition, **P** denotes the number of distinct multicast groups at a given time and N is an integer variable initialized with 0. We assume that the primary core C_0 maintains a dynamically growing linear array of integers, say A[]. At any given time the number of elements in the array is the number of active multicast groups in the network; A[1] is 0 and it corresponds to the first active group. The k^{th} entry of the array corresponds to the k^{th} newly active group. Primary core C_0 sets A[k] to (k-1) when it receives the first multicast packet for the k^{th} newly active group. We present now the two different schemes.

Let us now estimate the extra traffic load on the primary core. Without any loss of generality, let the average number of senders per group be n. So, the total number of senders is nP. Primary core always has to send the message '*continue with core C_j*'. So, there are nP unicasts to the sources from the primary core. In addition, we need to consider the following different cases as well. Let $P = km + n'$, k is an integer and $0 \leq k$ and $n' < m$.

Observe that for *GLSM-Cast-1* to work correctly, only primary core (C_0) tree spans all members of all groups.

In the next, we state the second scheme.

Let us now estimate the extra traffic load on the primary core. As in GLSM-cast1, primary core always has to send the message '*continue with core C_j*'. Hence, there are nP unicasts to the sources from the primary core. In addition, we need to consider the following as well.

Relative Merits: We now discuss briefly the effect on bandwidth consumption in the network caused by the two schemes. We also discuss the load on the primary core caused by the two schemes. We have shown above that in *GLSM-Cast1*, primary core has to execute extra multicasts which is absent in *GLSM-Cast2*. Note that total number of multicasts

```
GLSM-Cast1

Sender $S^{new}$ (executed by each new sender)
    1. sends the 1st multicast packet to $C_0$
    2. receives the message 'continue with core $C_j$'
    3. sends rest of the packets to $C_j$

Primary core $C_0$ (executed by the primary core)
    1. receives the 1st multicast packet from $S^{new}$
    2. if   corresponding ith group entry exists in A[]
            N = A[i]            // A[i] = i-1
            $C_j$ = N mod m    // selects the core for $S^{new}$
            multicasts the packet in the $C_0$-based tree
            unicasts 'continue with core $C_j$' to $S^{new}$
       else
            A[i] = i-1          // A[] grows dynamically
                                // ith newly active group
            N = A[i]
            $C_j$ = N mod m // selects the core for $S^{new}$
            multicasts the packet in the $C_0$-based tree
            unicasts 'continue with core $C_j$' to $S^{new}$

Core $C_j$
    1. multicasts the packets received from $S^{new}$
```

Case 1: $P \leq m$
number of extra single-packet multicasts by C_0 is n(P-1)
Case 2: $P > m$
number of extra single-packet multicasts by C_0 is
k(m-1)n + (n'-1)n

remains the same in both methods. Therefore, overall bandwidth consumption in the network, strictly due to multicasting, remains the same in both schemes. However, in *GLSM-Cast2* the number of extra unicasts done by C_0 compared to *GLSM-Cast1* is [k(m-1)n + (n'-1)n]. Therefore, the overall bandwidth consumption in the network is more in *GLSM-Cast2* than in *GLSM-Cast1*. However, actual load on C_0 in *GLSM-Cast2* is less, as it is not involved in any extra single packet multicast. In addition, primary core tree does not span over all groups unlike in the first one.

4 Low-Latency Load Shared Approach

In GLSM-Cast1 and GLSM-Cast2 schemes, the underlying assumption is that the cost (delay/number of hops etc.) between the root router (core) and a leaf router (a group member) is the pseudo diameter of the core. In reality, this assumption may or may not be true; it depends on the physical locations of the routers connected to the group members. The above two schemes have not considered the cost between a source router r_i (i.e., connected to a multicast source) and the core C_i which r_i uses for multicasting its packets. For cost (especially delay) sensitive applications, this last factor should be considered for selecting the core to achieve low latency in multicasting, in addition to the core's pseudo diameter. In this section, we have proposed two such schemes in which a sender itself selects a core for multicasting instead of taking help of the primary core unlike the schemes presented in the previous section. In fact, different senders belonging to the same group will select possibly different cores; the objective is to achieve low latency in multicasting with load sharing on the cores. Therefore, the following two approaches are more general in nature and have

```
GLSM-Cast2

Sender $S^{new}$ (executed by each new sender)
    1. sends the 1st multicast packet to $C_0$
    2. receives the message 'continue with core $C_j$'
    3. sends rest of the packets to $C_j$

Primary core $C_0$ (executed by the primary core)
    1. receives the 1st multicast packet from $S^{new}$
    2. if  corresponding ith group entry exists in A[]
            N = A[i]            // A[i] = i-1
            $C_j$ = N mod m     // selects the core for $S^{new}$

            if $C_j \neq C_0$
                $C_0$ unicasts the packet to $C_j$
                unicasts 'continue with core $C_j$' to $S^{new}$
            else  $C_0$ multicasts the packet
                unicasts 'continue with core $C_j$' to $S^{new}$

       else
            A[i] = i-1          // A[] grows dynamically
                                // ith newly active group
            N = A[i]
            $C_j$ = N mod m     // selects the core for $S^{new}$

            if $C_j \neq C_0$
                $C_0$ unicasts the packet to $C_j$
                unicasts 'continue with core $C_j$' to $S^{new}$
            else  $C_0$ multicasts the packet
                unicasts 'continue with core $C_j$' to $S^{new}$

Core $C_j$
    1. multicasts the packet received from $C_o$
    2. multicasts the packets received from $S^{new}$
```

Case 1: $P \leq m$
number of unicasts by C_0 to other cores is n(P-1)
Case 2: $P > m$
number of unicasts by C_0 to other cores is
k(m-1)n + (n'-1)n

the potential of outperforming the previous two schemes. Before we state the proposed approaches, we define the following.

Definition a. $\text{Cost}^{\alpha}(S^{new}, C_j) = \text{Cost}(S^{new}, C_j) + P_d(C_j)$
Definition b. $\text{Cost}^{\beta}(S^{new}, C_j) = \text{Cost}(S^{new}, C_j) + d_j(C_j)$

In the second definition, d_j denotes the *longest path* of the core tree rooted at C_j; we name d_j as the *depth* of the tree; note that $d_j \leq P_d(C_j)$. We shall elaborate on it further when we present the second low latency based approach in the next subsection.

We already have established the fact that every router has the same sorted list of *m* candidate cores. Let us denote this list as *L*

and $L = < C_0, C_1, C_2, \ldots, C_{m-1} >$, such that $P_d(C_i) \geq P_d(C_{i-1})$, $0 \leq i \leq m-1$.

4.1 Low Latency Approach 1

We assume that $P_d(C_j) = d_j$ for each candidate core. The algorithm is executed by each new source.

```
LLM-Cast1
  Sender $S^{new}$ (executed by each new sender)
      1. $S^{new}$ computes $\text{Cost}^{\alpha}(S^{new}, C_j)$ for all $C_j$ in $L$
      2. $S^{new}$ identifies the core $C_i$ such that $\text{Cost}^{\alpha}(S^{new}, C_i)$ is minimum
      3. $S^{new}$ unicasts packets to $C_i$ for multicast
```

4.2 Low Latency Approach 2

In the second approach, we consider the general case, that is, the depth of a core tree rooted at a core C_i is less than or equal to the pseudo diameter of the core; that is, $P_d(C_i) \geq d_i$.

We assume that each leaf router r_j unicasts a hello packet to its core C_i. Core C_i then determines Cost (r_j, C_i) from its unicast routing table. It computes *max {Cost* (r_j, C_i)*}* considering all its leaf routers. This is its *depth* d_i. The proposed second approach (*LLM-Cast2*) needs each core to create a list *L'* and it is done in the following way.

```
Each core $C_i$ executes the following algorithm.
    1. $C_i$ unicasts its $d_i$ to all $C_j$ in $L$, $j \neq i$
    2. $C_i$ receives each $d_j$
    3. $C_i$ creates the list $L' = < d_r, d_{r+1}, d_{r+2} \ldots, d_s >$,
       where $d_r$ is the depth of core tree rooted at $C_r$
       and $d_r \leq d_{r+1} \leq d_{r+2} \leq \ldots \leq d_s$; $|L'| = m$
```

***Remark 5**. Every core C_i creates identical sorted list of the depths (L').*

We observe that *LLM-Cast2* is likely to offer better low latency compared to *LLM-Cast1*; however, in *LLM-Cast2*, some extra information/computation is needed to form the list *L'*. Relative merits of the above two approaches are stated in more detail in the following subsection. In subsection 4.4, we have considered all the four multicast approaches to discuss their relative merits.

```
LLM-Cast2
    Sender $S^{new}$ (executed by each new sender)
        1. $S_{new}$ unicasts a request packet to $C_j$ to get the list $L'$
              such that Cost $(S^{new}, C_j)$ is minimum for all $C_j$ in $L$
                                    // it can be done via anycast as well
        2. $S^{new}$ determines the core $C_k$, such that
              $\text{Cost}^{\alpha}(S^{new}, C_k) = \min\{\text{Cost}^{\alpha}(S^{new}, C_j)\}$ for all $C_j$ in $L$
        3. $S^{new}$ determines $C_r$, such that
              $\text{Cost}^{\beta}(S^{new}, C_r) = \min\{\text{Cost}^{\beta}(S^{new}, C_j)\}$ for all $C_j$ in $L'$
        4. if    $\text{Cost}^{\alpha}(S^{new}, C_k) \leq \text{Cost}^{\beta}(S^{new}, C_r)$
                      $S^{new}$ unicasts packets to $C_k$
           else  $S^{new}$ unicasts packets to $C_r$
```

4.3 Relative Merits of LLM-Cast1 and LLM-Cast2

In the *LLM-Cast1* approach, one underlying assumption is that there will exist receiver(s) in the core-based tree at a distance from the core equal to the pseudo diameter of the core. This assumption may or may not be true; it solely depends on the geographical locations of the receivers. However, it may lead to not so accurate computation of the costs. Hence, multicast communication speed may be negatively affected.

The *LLM-Cast2* approach attempts to fine-tune this aspect of the *LLM-Cast1* approach from the viewpoint of achieving more accurate computation for the costs resulting in speedier multicast communication than that in the *LLM-Cast1* approach. For this, it needs to determine the depth of each core tree and therefore, it needs to consider some more unicasts compared to *LLM-Cast1*. The number of such unicasts, say Y is computed as follows.

$Y = m(m-1) + 2X$, where m is the number of cores and X is the number of senders (multicast sources).

Assuming that m is given, it is seen that Y varies linearly with X. It also means that bandwidth consumption also varies linearly with X. Observe that this is the extra amount of bandwidth consumption compared to that in *LLM-Cast1*.

***Remark 6**. LLM-Cast2 offers more accurate computation of cost than LLM-Cast1 resulting in speedier multicast communication than the LLM-Cost1 approach.*

***Remark 7**. LLM-Cast1 is slower but needs more bandwidth efficient than the LLM-Cast2 approach.*

4.4 Relative Merits of the Four Approaches

We consider the following aspects for comparison: load shared per core, bandwidth consumption, and multicast communication speed.

Load shared per core: In the *LLM-Cast1* and *LLM-Cast2* approaches, a source (sender) decides which core it will use for multicasting independent of any other sender. It adds an element of randomness in selecting a core for multicasting which is absent in the *GLSM-Cast1* and *GLSM-Cast2* approaches; therefore, both *LLM-Cast1* and *LLM-Cast2* approaches offer better load balancing than both *GLSM-Cast1* and *GLSM-Cast2* approaches.

Bandwidth consumption: We discuss this aspect from two viewpoints: number of trees rooted at the same core and number of unicasts needed in each approach.

(a) We start with the first one. In the *GLSM-Cast1* approach, the primary core spans over all existing groups and all other cores span over only some subsets of the groups. In the *GLSM-Cast2* approach, all cores including the primary one span over only some subsets of the groups. In the *LLM-Cast1* and *LLM-Cast2* approaches, each core spans over all existing groups. It causes more bandwidth consumption than in the two *GLSM* approaches. Besides, it also causes a larger number of memory states to be created and maintained by the routers present in the different trees compared to the two *GLSM* approaches.

(b) We now consider the effect on bandwidth consumption due to unicasts. Let N_1, N_2, N_3, and N_4 denote the respective total number of unicasts in the four approaches, namely *GLSM-Cast1, GLSM-Cast2, LLM-Cast1,* and *LLM-Cast2*. Then from the four algorithms we observe that
$N_1 = 2N_3$ and $N_2 = 3N_3$. Also, N_4 is approximately the same as N_1 when the number of cores is small and is less than N_2 as long as the number of senders is much larger than the number of cores, which is usually the case. This simple analysis suggests that the last two have better bandwidth utilization; however, the tradeoff is the extra computation done by every sender to determine the best cost.

Multicast communication speed: Faster communication is achieved in the *LLM-Cast1* and *LLM-Cast2* approaches than the two *GLSM* approaches, because they consider reduction of delay in addition to load sharing.

5 Core Migration

We extend the low-latency multicast schemes to allow migration of the cores. Such migration is needed when the performance at an alternate core is 'substantially' better than that at the current core. Below, we have stated the working principle of our proposed core migration approach before its formal presentation.

The *depth* d_i of a core tree with core C_i can change due to new member(s) joining or existing member(s) leaving. Therefore, to maintain low latency in multicasting, an existing source, say S* may need to migrate to a different core if it finds its 'new cost' has increased substantially from its '*current cost'*, both measured with respect to the current core, say C_i. It is understood that these costs can be either Cost $^{\alpha}$ (S^*, C_i) or Cost $^{\beta}$ (S^*, C_i) as determined by S* (following *LLM-Cast2* in subsection 4.2). To prevent frequent migration, which otherwise increases delay further in multicasting, a threshold δ can be set up so that *when new_cost – current_cost ≥ δ* , source S* migrates to an alternate core, say C_j, which offers the minimum cost compared to any other candidate core. For the proposed core migration algorithm to work correctly, it is required that every core C_i monitor its *depth* d_i; the regularity of this probing and this threshold value are dictated by both application behavior and the overhead cost associated with moving to a different core. We now state the algorithm.

```
Algorithm Core-Migration
Executed by each core $C_i$
  1.  $C_i$ unicasts its new depth $d_i$ to each $C_j$ in $L$
          // new_cost – current_cost ≥ δ
  2.  $C_i$ creates the new list $L'$
  3.  $C_i$ unicasts the list $L'$ to $S^*$
          // $C_i$ is the current core for $S^*$
Executed by source $S^*$
 1.  S* receives the list $L'$
 2.  S* determines the core $C_l$ such that
        new_cost ($C_l$) = min {current_cost ($C_r$)} for any
      $C_r$ in $L$, $r \neq l$
          // identical computations as in steps 2 to 4 in
  LLM-Cast2
  3.  S* unicasts packets to $C_l$ for multicasting
          // core migration takes place
```

It may be noted that joining/leaving of group members may cause changes in the depths of more than one core simultaneously. It may take some time for the convergence, i.e., when every core will have identical L'. Therefore, we consider the following in the above algorithm to tackle the problem of convergence: After building a new L' , a core C_i waits for a small time interval (Δt) to incorporate any further possible updating of L' caused by some other cores. After Δt, the core C_i unicasts the final list L' to S^* (step 3 of the pseudocode *Executed by each core C_i*).

6 Conclusion

In this paper, we have used an existing multicore selection scheme [7] for designing our proposed four novel load shared approaches. This multicore selection scheme has been the choice because of its ease of implementation [7]. The proposed load sharing schemes neither partition the receivers among the different core-trees nor they build a unique tree consisting of multiple cores [23-25, 27]. In the first two schemes, a sender belonging to any group does not decide which core to send packets to unlike in [27]; rather any source must send its first packet to the primary core and the primary core then decides if the sender needs to send the rest of its packets to some other core. In the other two, we have considered low latency load shared schemes, which are suitable for delay-sensitive applications. We have considered maximum path (*depth)* of a core-based tree as a parameter in addition to the *pseudo diameter* to achieve low latency. To the best of our knowledge, there does not exist any work in this direction that considers the above two parameters together. The two low latency load shared schemes can outperform the first two ones from the viewpoint of better load sharing since in the last two, any source can send to any core based on cost; thereby effectively adding an element of randomness in selecting a core for multicasting; however, it incurs extra overhead in that every core tree must span over all groups. Finally, we have extended the low latency approach to design a simple yet efficient core migration scheme, which considers both *depth* of a core tree and *pseudo diameter* of the core.

From the viewpoint of fault tolerance, the degree of fault-tolerance can be enhanced from covering single point-failure to any number of core failures as is evident from the *Observations 1, 2a, and 2b.*

References

[1] J. Nicholas Adams, and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM)", Internet Engineering Task Force (IETF), RFC-3973, January 2005.

[2] Tony A. Ballardie, "Core Based Tree Multicast Routing Architecture", Internet Engineering Task Force (IETF), RFC 2201, September 1997.

[3] S. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs", *ACM Transactions on Computer Systems (TOCS)*, 8(2):85-110, May 1990.

[4] M. Donahoo and E. Zegura, "Core Migration for Dynamic Multicast Routing", *Int'l Conf. Computer Comm. and Networks'*, June 1996.

[5] D. Estrin, M. Handley, A. Helmy, and P. Huang, "A Dynamic Bootstrap Mechanism for Rendezvous-Based Multicast Routing", *Proc. IEEE INFOCOM*, pp. 1090-1098, March 1999.

[6] M. Garey and D. Johnson, *Computers and Ins-tractability*, W. H. Freeman and Co., 1999.

[7] B. Gupta, A. Alyanbaawi, S. Rahimi, N. Rahimi, and K. Sinha, "An Efficient Approach for Load- Shared and Fault-Tolerant Multicore Shared Tree Multicasting", *Proc. IEEE INDIN*, Emden, Germany, pp. 937-943, July 2017.

[8] M. Handley Fenner, H. Holbrook, and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM)", Internet Engineering Task Force (IETF), RFC-4601, August 2006.

[9] H. Harutyunyan and M. Terzian, "A Multi-Core Multicast Approach for Delay and Delay Variation Multicast Routing", *Proc. IEEE 19th Int'l Conf. High Performance Computing and Communications*, Bangkok, Thailand, pp. 154-161, December 2017.

[10] W. Jia, W. Zhao, D. Xuan, and G. Xu, "An Efficient Fault-Tolerant Multicast Routing Protocol with Core-Based Tree Techniques", *IEEE Trans. on Parallel and Distributed Systems*, 10(10):984-1000, October 1999.

[11] W. Jia, W. Tu, W. Zhao and G. Xu, "Multi-Shared-Trees Based Multicast Routing Control Protocol Using Anycast Selection", *The International Journal of Parallel, Emergent and Distributed Systems*, 20(4):69-84, March 2005.

[12] Ayse Karaman and H. Hassanein, "Core Selection Algorithms in Multicast Routing – Comparative and Complexity Analysis", *J. Computer Communications*, 29(8):998-1014, May 2006.

[13] S. Koneru, B. Gupta, S. Rahimi, and Z. Liu, "Hierarchical Pruning to Improve Bandwidth Utilization of RPF-Based Broadcasting", *IEEE Symposium on Computers and Communications (ISCC)*, Split, Croatia, pp. 96-100, July 2013.

[14] S. Koneru, B. Gupta, and N. Debnath, "A Novel DVR Based Multicast Routing Protocol with Hierarchical Pruning", *International Journal of Computers and Their Applications (IJCA)*, 20(3):184-191, September 2013.

[15] S. Koneru, B. Gupta, S. Rahimi, Z. Liu, and N. Debnath, "A Highly Efficient RPF-Based Broadcast Protocol Using a New Two-Level Pruning Mechanism", *Journal of Computational Science (JOCS)*, 5(3):645-652, March 2014, SpringerLink, Berlin, Heidelberg, 2345:1045-1056, 2002.

[16] H. Lee and C. Youn, "Scalable Multicast Routing Algorithm for Delay Variation Constrained Minimum Cost Tree", *Proc. IEEE ICC*, New Orleans, LA, USA, 3:1343-1347, June 2000.

[17] H. Lin and S. Lai, "A Simple and Effective Core Placement Method for the Core Based Tree Multicast Routing Architecture", *Proc. IEEE Int. Conf. Performance, Computing, and Communications*, Phoenix, AZ, USA, pp. 215-219, February 2000.

[18] "Load Splitting IP Multicast Traffic over ECMP." Cisco, www.cisco.com/c/en/us/td/docs/ios/12_4t/, ip_mcast/configuration/guide/mctlsplt.html, Mar 2015.

[19] Adel Ben Mnaouer, M. Aissa, and A. Belghith, "An Efficient Core Selection Algorithm for Delay and Delay-Variation Constrained Multicast Tree Design", *Proc. Second international conference on Global Information Infrastructure Symposium (IEEE GIIS'09)*, pp. 281-285, June 2009.

[20] T. Pusateri, "Distance Vector Multicast Routing Protocol", Juniper Networks, Internet Engineering Task Force (IETF), draft-ietf-idmr-dvmrp-v3-11.txt, October 2003.

[21] G. Rouskas and I. Baldine, "Multicast Routing with End-to-End Delay and Delay Variation Constraints", *IEEE Journal on Selected Areas in Communications*, 15(3):346-356, April 1997.

[22] P. Sheu and S. Chen," A Fast and Efficient Heuristic Algorithm for the Delay and Delay Variation-Bounded Multicast Tree Problem", *Computer Communications*, 25(8):825-833, May 2002.

[23] C. Shields and J.Garcia-Luna-Acevez, "The Ordered Core Based Tree Protocol", *Proc. IEEE INFOCOM'97*, Kobe, Japan, pp. 884-891, April 1997.

[24] Y. Shim and S. Kang, "New Center Location Algorithms for Shared Multicast Trees", *Lecture Notes in Computer Science, SpringerLink*, Berlin, Heidelberg, 2345:1045-

1056, 2002.

[25] D. G. Thaler and C. Ravishankar. "Distributed Center-Location Algorithms", *IEEE Journal on Selected Areas in Communication*, 15(3):291-303, April 1997.

[26] C. Partridge Waitzman and S. Deering, "Distance Vector Multicast Routing Protocol (DVMRP)", Internet Engineering Task Force (IETF), RFC 1075, November 1988.

[27] D. Zappala, A. Fabbri, and V. Lo, "An Evaluation of Shared Multicast Trees with Multiple Active Cores", *Journal of Telecommunication Systems*, 19:461-479, March 2002.

Bidyut Gupta is a Professor at the Department of Computer Science, Southern Illinois University at Carbondale. He is a senior member of the IEEE. His current research interest includes fault-tolerant mobile computing, design of communication protocols, design of P2P federation architecture, and network security.

Ashraf Alyanbaawi was born in Madinah, The Kingdom of Saudi Arabia. Ashraf is a PhD Candidate in Southern Illinois University, Carbondale, IL, USA. After finishing his undergraduate degree at the University Tenaga Nasional, Malaysia, he started his career as a Teaching Assistant at Taibah University, Yanbu, Saudi Arabia. In 2016, he received his MS degree in Computer Science from Southern Illinois University. His research focus is in the area of computer networks.

Nick Rahimi is an Assistant Professor in the Computer Science Department at Southeast Missouri State University. His main research interests revolve around Computer and Network Security, Distributed Systems, Peer-to-Peer Networks and their Privacy, and Data Communication. He has earned two Bachelor of Science degrees in Software Engineering and Information Systems Technologies. Nick obtained his Master and Ph.D. degrees in Computer Science from Southern Illinois University.

Koushik Sinha is currently an Assistant Professor in the Department of Computer Science at Southern Illinois University, Carbondale. He is the co-author of the book Wireless Networks and Mobile Computing published by CRC Press, Taylor and Francis Group, USA in 2015. Prior to joining SIU, he was with the *Social Computing Group* of *Qatar Computing Research Institute*, Qatar from 2013 to 2015. Previously, he was a Research Scientist at *Hewlett-Packard Labs*. He is a Senior Member of the IEEE. His current research focus is in the areas of 5G, wireless sensor networks, IoT and Fog Computing.

Ziping Liu is a Professor in the Department of Computer Science at Southeast Missouri State University. Her research interests include wireless ad hoc network/sensor network, distributed computing, cloud computing, wireless network security, game development, game AI, mobile computing and web development.

Instructions For Authors

The International Journal of Computers and Their Applications is published multiple times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration:

1. Email your manuscript to the Editor-in-Chief, Dr. Fred Harris, Jr. Fred.Harris@sce.unr.edu.
2. Illustrations should be high quality (originals unnecessary).
3. Enclose a separate page for (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.

B. Manuscript Style:

1. The text should be, **double-spaced** (12 point or larger), **single column** and **single-sided** on 8.5 X 11 inch pages.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. Figures should be captioned and referenced.

C. Submission of Accepted Manuscripts:

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-chief.
2. The submission may be on a CD/DVD, or as an email attachment(s). **The following electronic files should be included:**
   - Paper text (required)
   - Bios (required for each author). Integrate at the end of the paper.
   - Author Photos (jpeg files are required by the printer)
   - Figures, Tables, Illustrations. These may be integrated into the paper text file or provided separately (jpeg, MS Word, PowerPoint, eps). title of the paper.
3. Specify on the CD/DVD label or in the email the word processor and version used, along with the title of the paper.
4. Authors are asked to sign an ISCA copyright form (http://www.isca-hq.org/j-copyright.htm), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges:

After a manuscript has been accepted for publication, the author will be invoiced for publication charges of $50 **USD** per page (in the final IJCA two-column format) to cover part of the cost of publication. For ISCA members, $100 of publication charges will be waived if requested.

January 2014