Bedrock Computer Technologies, LLC v. Softlayer Technologies, Inc. et al

EXHIBIT E

Dockets.Justia.com

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

Bedrock Computer Technologies LLC, Plaintiff, v. Softlayer Technologies, Inc., et al. Defendants.	Case No. 6:09-CV-269-LED JURY TRIAL DEMANDED

DECLARATION OF DR. KEVIN JEFFAY
IN SUPPORT OF

DEFENDANTS' MOTION FOR SUMMARY JUDGMENT REGARDING INVALIDITY OF U.S. PATENT NO. 5,893,120

I, Dr. Kevin Jeffay, declare as follows:

1. I am over 18 years of age. The statements made herein are true and correct and are of my own personal knowledge. I make this declaration in connection with Defendants' Motion for Summary Judgment of Invalidity of U.S. Patent No. 5,893,120.

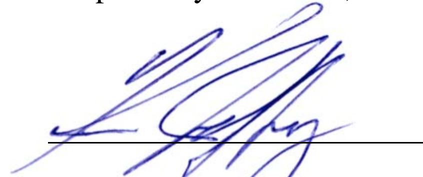
2. A detailed record of my professional qualifications, including a list of publications, awards, professional activities, and recent testimony either at trial or at deposition, is attached as Exhibit B to the Invalidity Expert Report I submitted in this action (attached as Exhibit A hereto).

Invalidity

3. I wrote a report, dated January 25, 2011, detailing my opinions regarding the invalidity of U.S. Patent No. 5,893,120 (the '120 Patent), as well as the basis and reasons therefore, a true and correct copy of which is attached as Exhibit A. This report accurately reflects my true opinion.

4. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct. Executed on February 8, 2011 in Chapel Hill, N.C.

Respectfully submitted,



Kevin Jeffay, Ph.D.

EXHIBIT A

INVALIDITY EXPERT REPORT OF DR. KEVIN JEFFAY, PH.D

Bedrock Computer Technologies, LLC vs. Softlayer Technologies, Inc., et al.

Case No. 6:09-cv-269-LED (E.D. Tex.)

January 25, 2011

I. INTRODUCTION

1. My name is Kevin Jeffay. I am a tenured professor in the Department of Computer Science at the University of North Carolina at Chapel Hill where I currently hold the position of Gillian T. Cell Distinguished Professor of Computer Science.

2. I have been retained on behalf of Google Inc., and Match.com, LLC, to offer an expert opinion on the validity of claims 1-8 of U.S. Patent No. 5,893,120 (hereafter, "the '120 Patent").[1] I submit this report to describe my opinions on these matters.

3. This report is structured as follows. The remainder of this section presents a synopsis of my background and relevant expertise that qualify me as an expert in this matter. Section II summarizes my opinions that the asserted claims of '120 Patent are invalid in light of the prior art that existed at the time of filing on the application for the '120 Patent. Section III provides some background material on the technology disclosed in the '120 Patent. Section IV discusses my understanding of the legal standards I am to apply when assessing the validity of patent claims. Section V presents my opinion on the definition of a person of ordinary skill in the art at the time of the application for the '120 Patent. Section VI presents my analysis of the validity of claims 1-8 of the '120 Patent.

4. In reaching the conclusions described herein, I have considered the documents and materials identified in Exhibit A that is attached to this report. My opinions are also based upon my education, training, research, knowledge, and personal and professional experience.

5. If asked at hearings or trial, I am prepared to testify on issues pertaining to the materials alleged to be prior art to the '120 Patent, how those materials relate to the asserted

---

[1] Though I have not currently been retained to render an opinion by any other defendants in this case, I may ultimately render an opinion at trial on behalf of other defendants.

claims of the ‘120 Patent, and on the validity of the claims of the ‘120 Patent. I am further prepared to testify on the operation of the systems and methods of the ‘120 Patent, relevant background material such as computer programming and data structures.

6. The details of my analysis, and conclusions that form the basis for any testimony I may give, are provided below. To support or summarize my opinions, any testimony I give may include appropriate visual aids, some or all of the data or other documents and information cited herein or identified in Exhibit A, and additional data or other information identified in discovery.

7. I reserve the right to supplement the opinions in this report based on any subsequent testimony or facts revealed through discovery, as well as any subsequent reports issued by the Plaintiff's experts and the Court's formal claim construction of the disputed terms in this matter.

A. <u>Qualifications</u>

8. A detailed record of my professional qualifications, including a list of publications, awards, professional activities, and recent testimony either at trial or at deposition, is attached as Exhibit B to this report and summarized below.

9. I have a Ph.D. in computer science from the University of Washington, a M.Sc. degree in computer science from the University of Toronto, and a B.S. degree with Highest Distinction in mathematics from the University of Illinois at Urbana-Champaign.

10. I have been involved in the research and development of networked computing systems for nearly 30 years. I have been a faculty member at North Carolina since 1989 where I have performed research and taught in the areas of operating systems, computer networking, anomaly detection, distributed systems, the world-wide web, multimedia computing and networking, and real-time and embedded systems, among others. I consider myself an expert in these areas as well as others.

11. I have authored or co-authored over 100 articles in peer-reviewed journals, conference proceedings, texts, and monographs in the aforementioned areas of computer science and others. I am currently an Associate Editor for the journal *Real-Time Systems* and have previously served as the Editor-in-Chief for the journal *Multimedia Systems*. In addition, I have edited and co-edited numerous published proceedings of technical conferences and have edited a book of readings in multimedia computing and networking (with Hong-Jiang Zhang) published by Morgan Kaufman.

12. I have served on numerous proposal review panels for the National Science Foundation and other international funding agencies in the aforementioned areas of computer science. I have served as a program chair or member of the technical program committee for over 100 professional, international, and technical conferences, workshops, and symposia.

13. I have worked for, and had research collaborations with, companies such as Cisco Systems, CloudShield, VMware, Lucent, Cabletron/Aprisma, IBM, Dell, Sun, Intel, DEC, and Hewlett Packard, among others. All of these collaborations have involved computer networking, distributed systems, and the Internet.

14. I am a named inventor on two U.S. Patents and I have a third application for a patent pending. These patents relate generally to computer networking.

15. In my research and teaching I have considered problems of the design and implementation of operating systems, computer networks, network communication protocols, and distributed systems and services. I have also considered problems of network measurement and the evaluation of network and server performance. Since 1992, I have served as the Director of Undergraduate Studies for the Department of Computer Science at UNC. In this role I have been responsible for curriculum development and student advising for the Department.

16. I have served as an expert witness and technical consultant in litigation matters concerning operating systems, computer networks, distributed systems, telecommunication networks, and telecommunications systems for cellular, wireline, and voice over IP (VoIP) telephony, among others. This work has been performed on behalf of entities such as Cisco Systems, AT&T, Lucent, Alcatel-Lucent, Enterasys, Nortel Networks, AOL, Verizon Wireless, Cox Communications, Motorola, Green Hills Software, Akamai Technologies, Dell, and Toshiba, among others. I have testified in several trials, arbitrations, and claim construction hearings as an expert witness.

B. Compensation

17. I am being compensated for my work in this matter at the rate of $450 per hour plus expenses. My compensation is in no way tied to the outcome of this matter.

II. SUMMARY OF OPINIONS

18. I understand that this suit has been brought by Bedrock Computer Technologies LLC ("Bedrock"), the current owner of the '120 Patent. I further understand that Bedrock has asserted claims 1-8 of the '120 Patent (collectively, "the asserted claims") in this matter. I have studied the '120 Patent and its file history as well as the prior art references cited below and listed in Exhibit A. Having reviewed these materials, it is my opinion that:

- NRL BSD key management computer source code files *key.c* and *key.h* by Bao Phan, Randall Atkinson*, and* Dan *McDon*ald, US Naval Research Laboratory, 1995 (hereafter, "the NRL BSD source code") in combination with itself or with "The Art of Computer Programming," Volume 3, "Searching and Sorting," by D.E. Knuth, Addison-Wesley, 1973 (hereafter, "the Knuth reference" or simply "Knuth"), Robert L. Kruse, *Data Structures and Program Design,* Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1987 (hereafter "Kruse"), or Daniel F. Stubbs and Neil W. Webre*, Data Structures with Abstract Data Types and Pascal*, Brooks/Cole Publishing Company, Monterey, California, 1985 (hereafter "Stubbs") renders obvious claims 1, 5, and 7. Further, the NRL BSD code in combination with U.S. Patent 6,119,214, "Method For Allocation of Address Space in a Virtual Memory

System," to Dirks, filed April 25, 1994, and issued September 12, 2000 (hereafter, "the Dirks '214 Patent," or simply "Dirks") renders obvious claims 2, 4, 6, and 8.

- Xinu Operating System computer source code file *gcache.c* by Douglas Comer and Shawn Ostermann, Purdue University, October 1991 (hereafter "GCache source code"), and the companion documentation "GCache: A Generalized Caching Mechanism," Douglas Comer and Shawn Ostermann, Technical Report CSD-TR-91-081, Department of Computer Science, Purdue University, November 1991 (revised, March 1992) (hereafter "GCache report") anticipates all claims of the '120 Patent and, in combination with Dirks, renders obvious claims 2, 4, 6, and 8.

- Linux computer source code *route.c*, version 1.0.14, by Ross Biro *et al.*, dated May, 31, 1993, part of the Linux operating system kernel, version 2.0.1, dated July 3, 1996 (hereafter, "Linux v2.0.1") anticipates all claims of the '120 Patent and, in combination with Dirks, renders obvious claims 2, 4, 6, and 8.

- U.S. Patent 5,121,495, "Methods and Apparatus For Information Storage and Retrieval Utilizing Hashing Techniques," to Nemes, issued June 9, 1992 (hereafter, "the Nemes '495 Patent"), in combination with Knuth, Kruse, or Stubbs renders obvious claims 1, 3, 5, and 7 of the '120 Patent and, in combination with Knuth, Kruse, or Stubbs, and Dirks, NRL BSD, Linux v2.0.1, or GCache, renders obvious claims 2, 4, 6, and 8.

19. The bases for these opinions are presented in detail below.

III. BACKGROUND ON THE TECHNOLOGY OF THE '120 PATENT

20. The technology at issue in this matter is a particular set of methods and systems for storing and retrieving information in a computer system. The methods revolve around the use of a basic technique in computer programming known as "hashing." This section provides a brief background on computer programming generally, and some specific ways that programmers arrange data in the memory of computer so as to efficiently store and retrieve data. This includes a discussion of the well-known technique of hashing as well as an overview of the specific technology disclosed in the specification and claims of the '120 Patent.

A. Background on Computer Programs and Data Structures

21. Computer programs are sequences of instructions that cause the processor of the computer to perform some useful task. The instructions perform various manipulations of data that are stored in the memory of the computer. (The term “memory,” as used in this report, refers to the *random access memory* [RAM] of the computer. This is a form of storage wherein the computer’s processor can read or write any storage location in the memory in a single step at any time.)

22. Abstractly, a computer program is the combination of one or more *algorithms*, and one or more *data structures*. An algorithm is a set of specific steps (a “procedure”) to be carried out by the processor to generate some specific result. For example, in the computing field, there exist well-known algorithms to sort a sequence of numbers, to efficiently search a set of data values for the presence or absence of a particular value, or to perform some mathematical calculation such as to determine if a given value is a prime number, or to compute the greatest common divisor of two integer numbers.

23. Data structures refer to the manner in which data is arranged in the computer’s memory. Just as one can imagine numerous schemes for organizing a set of paper files in a filing cabinet, a programmer can similarly chose from a number of differing ways of organizing data in memory. And just as the scheme used for organizing paper files in a filing cabinet can effect the effort required to store a new file or retrieve an existing file, so to can the scheme a programmer chooses to use to arrange data in a computer’s memory effect the effort required by a program to store and retrieve data.

1. Arrays

24. The simplest data structure for storing and retrieving data is known as an *array*. An array consists of a contiguous sequence of memory locations. An array is used to store a

sequence (a collection) of related data values. Arrays have the advantage that any element of the array can be accessed in a single step. If x is the location in memory of the first element in the array (x is the "starting address" of the array), and y is the number of memory locations required to store an element of the array, then the kth element in the array can be accessed in a single step by simply accessing memory location $x + (k–1)y$. The figure below shows an array of nine elements where locations 2-4 of the array presently store data values v_1 through v_3.

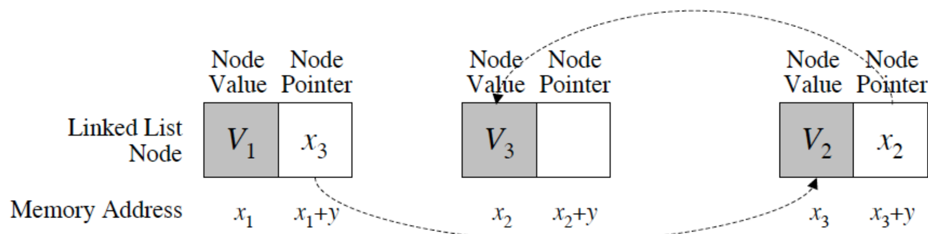
Array Index	1	2	3	4	5	6	7	8	9
Array Contents		V_1	V_2	V_3					
Memory Address	x	$x+y$	$x+2y$	$x+3y$	$x+4y$	$x+5y$	$x+6y$	$x+7y$	$x+8y$

25. Arrays have the disadvantage that once the array is created, the size of the array is fixed. That is, arrays occupy a fixed amount of memory (a fixed number of contiguous memory locations). If additional storage is required, a new, larger array must be created and the contents of the old array copied into the new, larger array. If the old array is large, the time required to copy the old array to the new array can be large and thus undesirable.

2. Linked Lists

26. Another simple data structure for storing and retrieving data is known as a *linked list*. A linked list consists of a set of *nodes* (also called "records") that can be randomly distributed throughout the computer's memory. A node is a set of adjacent memory locations (a small "block" or "region" of memory). In the simplest case, each node consists of two memory locations (two "fields"): a first location storing a *value* and a second location storing a *pointer* to another node. The value field of a node is the data item that is stored in the node (*i.e.*, the data value that the programmer wishes to store in memory). The pointer field of a node is the address of (an indication of the location in memory of) the next node in the list.

27. The figure below shows a linked list of three nodes where y is the number of memory locations required to store a data value in a node. The nodes of the linked list are stored in memory at (unrelated) addresses x_1, x_2, and x_3. In this example, the first node of the linked list contains the data value v_1, the second node contains the data value v_2, and the third node contains the data value v_3. The pointer field in the first node of the linked list contains the starting address of ("points to") the second node in the linked list, and the pointer field in the second node of the linked list contains the starting address of the third node in the linked list. As the list contains only three nodes, the pointer field in the last node of the linked list (the third node of the linked list [the node containing the data value v_3]), contains an indication that the pointer field is "empty" or invalid (does not point to any node in the linked list).



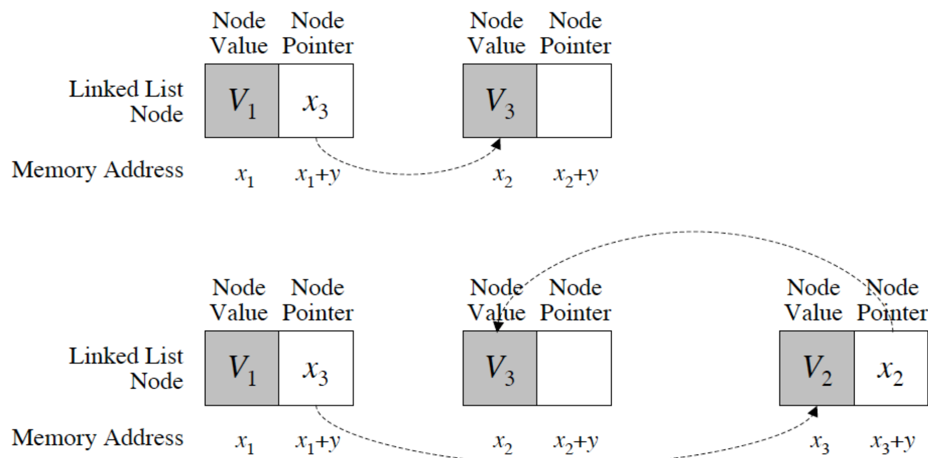
28. Using the data values stored in a linked list is more complex than using the data values stored in an array. In order to use a data value stored in a linked list (or in an array), the data value must first be located within the linked list (or within the array). Whereas in an array, the kth element in the array can be located in a single step, in a linked list, k steps are required to locate the kth node. To locate the kth node of a linked list, the linked list must be traversed. To do this, the first node in the linked list (called the "head" or "root" of the list) is located. The data stored in the first node is read to find the pointer to (to find the location in memory of) the second node in the list. Next, data stored in the second node in the list is read to find the pointer to (to locate) the third node in the list. The data stored in the third node in the list is then read,

and so on until the kth node in the list is located. At this point the data stored in the kth node of a linked list can be used.

29. This process of accessing a linked list is most commonly performed with a program structure known as a *loop*. A loop consists of a series of instructions (called the "body" of the loop) that are executed over and over until some condition is met. The execution of the loop starts with a consideration of the head of the linked list and each subsequent execution of the loop body (each "iteration" of the loop) considers the next node in the list. Thus, accessing a linked list consists of starting with the head of the linked list and iteratively considering the nodes in the list in turn until some condition is met. The notion of iteration is fundamental to the process of accessing a linked list. Inserting data into a linked list, removing data from a linked list, and searching for a data value in a linked list, all require an iterative procedure to be followed (and this iterative procedure is most commonly realized with a loop structure).

30. Because accessing a linked list (for example, to locate a node in the linked list) is an iterative process, it can require significantly more steps (and thus require more time) than accessing an array (for example, locating an element in an array). However, the advantage of linked lists is that the size of the list (the number of nodes in the list) can easily grow or shrink over time without requiring any copying of data items. To increase the number of nodes in a linked list, a new node can be created anywhere in memory and added to ("linked" into) the list. A new node is added to the list by changing the pointer of a node that is already in the list to point to (to store the location in memory of) the newly allocated node, and to set the pointer of the newly allocated node to point to the node that the pointer of preceding node in the list previously pointed to.

31. For example, the two figures below illustrate the steps of adding a node to a linked list. In this example, a linked list initially contains two nodes containing data values v_1 and v_3. A new node containing the data value v_2 is inserted into the linked list after the node containing the data value v_1. In this case, the pointer in the node containing data value v_1 is adjusted to point to the new node containing data value v_2, and the pointer in the new node containing data value v_2, is set to point to the node containing data value v_3. Thus, adding a new node to a linked list requires at most two pointer modification operations.



32. Deleting[2] a node from a linked list requires one pointer modification operation. To delete a node from a linked list, the pointer in the predecessor node of the node to be deleted is simply adjusted to "bypass" the node to deleted. For example, the figure below illustrates the step of deleting the node containing the data value v_3 from the linked list above. Deletion is accomplished by locating the node in the linked list that precedes the node containing the data value v_3 in the linked list, and adjusting the pointer in the predecessor node to bypass the node containing the data value v_3. In this case, since the node containing the data value v_3 was the last

[2] The Court's claim construction order construes "removing" a record from the linked list as merely adjusting the pointers in the linked list to bypass the record, and not to include deallocation of the record. My discussion of deletion from a linked list applies that same construction.

node in the linked list, the pointer in the predecessor node (the node containing data value v_2) is adjusted to indicate that there are no additional nodes in the linked list.

33. When a node is deleted from a linked list, most typically the storage associated with the node is reclaimed for future use by the program.

34. Whether or not a programmer chooses to use an array or a linked list in a program to store and retrieve data will depend on whether the programmer chooses to optimize the execution of their program for low data retrieval times or the ability to easily manipulate the amount of memory available to store data. The former concern would argue for use of an array data structure while the latter concern would argue for the use of a linked list data structure.

3. Knowledge of Arrays and Linked Lists was Common in 1997

35. Arrays and linked lists are among the most very basic and elementary data structures used in computer programs. By 1997, it was common for students desiring a college degree in computer science to have started their study of computer programming in high school and to have already encountered, and used, arrays and linked lists in programs they wrote while in high school. In college, a computer science major (or students in other majors requiring a knowledge of programming) would typically take a formal course in data structures in either their freshman or sophomore year. However, such a data structures course would consider structures significantly more advanced than arrays and linked lists and would in fact assume students already possessed knowledge of arrays and linked lists.

36. For example, presently, at the University of Texas at Tyler (UT Tyler), students desiring formal training in computer science would take the computer science course COSC 2336, "Data Structures and Algorithms," in the first semester of their sophomore (second) year. As expected, based on information from UT Tyler's web site, this class provides a sound grounding in the fundamentals of data structures and algorithms and includes material on hashing and collision resolution schemes (see, *http://cs.uttyler.edu/CourseSyllabi/Undergraduate/ COSC%202336.pdf*). While this information is contemporary, it comports with my personal knowledge of and experience with data structure and algorithms classes going back nearly 30 years.

B. Background on Searching and Hashing

1. Searching

37. While individual elements stored in an array can be accessed in a single step, the problem of determining whether or not a particular data value is currently stored in an array, or where a particular data value is stored in an array, may require several steps. This is because, in general, determining whether or not a particular data value is currently stored in an array, or where a data value is stored in an array, will require the use of an algorithm for searching the contents of the array.

38. For example, to determine if a data value v is currently stored in an array, the simplest search algorithm, *linear search* (also called *sequential search*), simply traverses each array location in sequence, checking the value stored in each location against the value v. If an array location is found that stores the value v, the algorithm produces as its result an indication of the location in (the address, or "index," of the location within) the array storing value v. If the

array does not store the value *v*, then the algorithm produces as its result an indication that the desired value was not found.

39. In the best case, the linear search algorithm can determine if a value is stored in an array in one step. This occurs when the sought after data value happens to be stored in the first element in the array. In the worst case, the linear search algorithm will require n steps to determine if a data value is stored in an array, where n is the length of (the number of elements in) the array. The worst case occurs if the sought after data value happens to be stored in the last, or n^{th}, element in the array. (If the sought after data value is not stored in the array then n steps are required to make this determination because the entire array must be searched.) On average, the linear search algorithm can determine where a data value is stored in an array in $(n+1)/2$ steps.

40. These performance results also hold for a linear search for a data value stored in a linked list. That is, in the best case, the linear search algorithm can determine if a value is stored in a linked list in one step; in the worst case, the linear search algorithm will require n steps to determine if a data value is stored in a linked list (where n is the number of nodes in the linked list); and in the average case, the linear search algorithm can determine where a data value is stored in a linked list array in $(n+1)/2$ steps.

41. In either case, linear search in either an array or linear search in a linked list, requires a traversal of the respective data structure (an iteration through the elements of either the array or the linked list).

2. Hashing

42. Hashing is a class of techniques for storing and retrieving data in a computer system that are more efficient than linear searching. Hashing attempts to minimize the average number of steps required to search for a data value in memory. In particular, data storage and

retrieval systems using hashing attempt to ensure that most data values stored in the system can be located in a small number of steps (ideally one step), independent of where the data values are stored in memory.

43. Hashing algorithms use a data structure abstractly called a "hash table." In the simplest case, a hash table is just an array. Locations in the array are colloquially referred to as "buckets." Data values are stored in the array via the use of a *hash function*. A hash function is a mathematical function that takes a value (also called a "key" or a "search key") and computes an integer in the range from 1 to n, where n is the number of elements in the hash table array (the number of "buckets" in, or the size of, the hash table). In the simplest case, data values are stored in the hash table by computing the hash function of the data value (by "hashing the value") to produce an index into the table (an integer value from 1 to n[3]). If a data value v "hashes" to the value h (*i.e.*, if applying the hash function to data value v produces the result h), then the data value v is stored in the hth location in the hash table (*i.e.*, the data value is stored in the hth element in the hash table array or, equivalently, the hth bucket in the hash table). To later determine if the data value v is stored in the hash table, one need only compute the hash of v and check the location in the hash table given by the hash function for the presence of value v.

a. Collisions and Collision Resolution

44. Generally, the domain of the hash function (the set of possible values to be stored in the hash table) will be considerably larger than the size of the hash table. For example, consider storing social security numbers for employees of a small company in a hash table. If the company has 100 employees then, in principle, the hash table would require no more than 100

[3] Hash functions would actually generate hash table indices in the range from 0 to n–1. Similarly, indices in hash tables with n elements would range from 0 to n–1. However, to simplify the presentation, here it is assumed hash functions, and hash table indices, range from 1 to n.

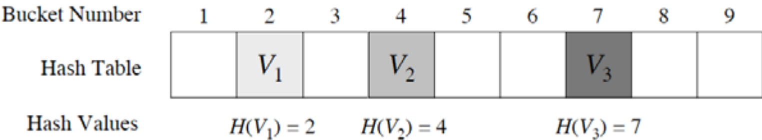
storage locations. However, the domain of the hash function, the set of all possible social security numbers (hundreds of millions of numbers), will be considerably larger than the size of the hash table (100 elements). This necessarily means that it will be possible for different social security numbers to hash to the same hash table location. That is, it is possible for the hash of one employee's social security number to equal the hash of another employee's social security number. More generally, in any hash table it will be possible for data values v_1 and v_2 to hash to the same hash table location (*i.e.*, it is possible for the hash of v_1 and v_2 to produce the same number). Thus, when trying to store a particular value v_1 in the hash table, it may be the case that the location in the hash table corresponding to v_1 (the location in the hash table given by applying the hash function to the value v_1), may be occupied by another value v_2 that was previously stored in the hash table. In such a situation, a *collision* is said to occur.

45. For example, consider a hash function where the hash of a data value v_1 equals the hash of another data value v_2. Assume that the hash of data value v_1 equals h (and thus the hash of data value v_2 also equals h). If the data value v_2 is inserted into the hash table, v_2 will be stored in the hash table at location h (v_2 will be stored in the hth bucket). If one later attempts to store the data value v_1 in the hash table, they will attempt to insert the data value v_1 into location h of the hash table but find that the hth bucket of the hash table already contains a value (the value v_2). Thus, the insertion of data value v_1 into the hash table will "collide" with the previous insertion of data value v_2. Collisions are a natural consequence of storing data items with a wide range of values in tables of fixed size.

46. Hashing algorithms deal with collisions via a *collision resolution scheme*. There are two basic forms of collision resolution schemes: *open addressing* and *chaining* (also called *external chaining*). In open addressing, when a data value v_1 to be stored in a hash table at

location h collides with a data value v_2 (*i.e.*, when the data value v_2 was previously stored in the hash table at location h), the insertion procedure "probes" (searches) the hash table for an empty location in which to store the data value v_1. The probing of the hash table for an empty location follows a *probe sequence*. In the simplest case, known as *linear probing*, the probing proceeds in a "linear" fashion by checking hash table locations h+1, h+2, h+3, …, n in sequence, looking for an empty location. If an empty location is found, then data value v_1 is stored in the first such empty location. If an empty location is not found, the "probing" continues with locations 1, 2, 3, … h–1, of the hash table. If no empty locations are found, then the hash table is "full" and some remedial action must be taken. (Note that after location h–1 of the hash table is probed, the entire table has been searched.)

47. For example, consider the following graphical illustration of hashing with collision resolution based on linear probing. The figure below shows the first nine elements of a hash table (the first nine elements of an array) that currently stores data values v_1, v_2, and v_3.



In this example, applying the hash function H to the data value v_1 yields the result 2 (*i.e.*, $H(v_1) = 2$). Since the hash of v_1 equals 2, data value v_1 is stored in the second bucket of the hash table. Similarly, in the figure above, the hash of v_2 equals 4 ($H(v_2) = 4$) and thus data value v_2 is stored in the fourth bucket of the hash table, and the hash of v_3 equals 7 ($H(v_3) = 7$) and thus data value v_3 is stored in the seventh bucket of the hash table.

48. Now consider inserting data value v_4 into the hash table. Assume data value v_4 hashes to the value 2 (*i.e.*, assume $H(v_4) = 2$, the same hash value as data value v_1). An attempt to insert data value v_4 into the second bucket of the hash table will result in a collision because data

value v_1 is already stored in the second bucket of the hash table. The collision will be resolved by linear probing starting with hash bucket 2+1 = 3. In this case, bucket 3 is empty and hence data value v_4 would be stored in the third location of the hash table as shown below.[4]

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table		V_1	V_4	V_2			V_3		
Hash Values		$H(V_1) = 2$	$H(V_4) = 2$	$H(V_2) = 4$			$H(V_3) = 7$		

49. Next, consider inserting data value v_5 with hash value 4 ($H(v_5) = 4$) into the hash table. The insertion of data value v_5 will also result in a collision because data value v_2 is already stored in the fourth bucket of the hash table. Linear probing will again be used and data value v_5 will be stored in the fifth bucket of the hash table as shown below.

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table		V_1	V_4	V_2	V_5		V_3		
Hash Values		$H(V_1) = 2$	$H(V_4) = 2$	$H(V_2) = 4$	$H(V_5) = 4$		$H(V_3) = 7$		

50. Next, consider inserting data value v_6 with hash value 2 ($H(v_6) = 2$) into the hash table. The insertion of data value v_6 will result in a collision with data value v_1 that also has a hash value of 2 (and is stored in the second bucket of the hash table). Once again linear probing is used to find an empty hash table location. As before the probe sequence would start with the 2+1 = 3rd hash table bucket. In this case, because buckets 3 through 5 are full (each contains a data value), the linear probing algorithm would generate the probe sequence of buckets 3, 4, 5, and 6. The linear probing algorithm would stop at the 6th bucket because this is the first bucket

[4] For convenience, in the following figures, hash table entries with data values that hash to the same hash bucket are illustrated with the same shade of grey. Thus, for example, hash table entries 2 and 3 have the same shading because the data values they contain, v_1 and v_4, hash to the same value ($H(v_1) = H(v_4) = 2$).

encountered that was empty. As a result of this probing, data value v_6 will be stored in the sixth bucket of the hash table as shown below.

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table		V_1	V_4	V_2	V_5	V_6	V_3		

Hash Values: $H(V_1) = 2$ $H(V_4) = 2$ $H(V_2) = 4$ $H(V_5) = 4$ $H(V_6) = 2$ $H(V_3) = 7$

51. Notice that if a seventh data value were to be added to the hash table, and this seventh data value hashed to any value in the range 2 through 7, the seventh data value would be stored in the eighth bucket of the hash table. This behavior would occur because buckets 2 through 7 of the hash table are full and the eighth bucket is the first empty bucket. Thus, for any collision that occurs with buckets 2 through 7, linear probing would terminate at the eighth bucket.

52. Note further that when probing for an empty bucket to store a data value v, a probe sequence can encounter buckets containing data values that collide with v as well as buckets containing data values not colliding with v. That is, not every hash table location considered in a probe sequence for data value v collides with data value v. For example, if data value v_7 with a hash value of 2 is stored in the hash table above, data value v_7 will collide with data value v_1 stored in location 2 of the hash table and, as a result, the linear probing algorithm will place data value v_7 into the eighth bucket of the hash table as previously described. However, in the course of probing for the first free bucket after the second hash bucket (the first free bucket after the bucket where data value v_7 hashed to), the linear probing algorithm will consider hash table locations 3, 4, 5, 6, 7, and 8, of which locations 4, 5, and 7 contain data values that do not collide with data value v_7. This observation will be important when discussing the process of deleting data values from a hash table with collision resolution based on open addressing.

53. The second collision resolution scheme, external chaining, uses a different data structure for the hash table and a different procedure for searching the hash table. When external chaining is used, the hash table still employs an array, however, the elements of the array (the buckets of the hash table) no longer store data values. Instead, each array location stores a pointer to a linked list (a pointer to the head of a linked list). That is, each array element now stores an indication of the location of the head of a linked list rather than a data value.

54. In external chaining, there will be a separate linked list associated with (pointed to by) each hash table array location. The linked list for the hth location in the array (the linked list pointed to by the hth location in the array) is used to store all the data values that hash to the value h. In this arrangement, the "hash table" is now a hybrid structure consisting of an array of pointers and a set of linked lists. When storing a data value v in this new hash table, the hash of data value v is computed as before and the resulting hash value h is still used to locate the hth element of the hash table array. However, the hth location of the array is now read to find the head of a linked list of data values that hash to the hth location of the array. The head of the appropriate linked list is found by reading the hth location in the array of the hash table and following the pointer stored at that location. Once the head of the linked list is found,[5] a new node for the linked list is created, the data value v is stored in the new node, and the new node is inserted into the linked list using the procedure previously described. (Precisely where the new node is inserted into the linked list is up to the programmer.)

[5] Before inserting the data value into the linked list, one common implementation choice to prevent the insertion of duplicate values is to search for the data value being inserted and, if there, not inserting the new data value. This is commonly done when there are concerns with duplicate records being stored in the linked list/hash table.

b. Searching For Data Values in a Hash Table

55. The process of searching for data values stored in a hash table follows many of the same steps as the process for storing elements in a hash table. To determine if a data value v is stored in a hash table, the hash of v is first computed. If the hash of v is the value h, then in the case of hashing with collision resolution based on open addressing, the hth hash bucket (location h of the hash table array) is accessed. If the hth hash bucket is empty, then data value v is not stored in the hash table and the search procedure would return an indication of this fact. If the hth hash bucket is not empty, then the data value stored in location h is compared to data value v. If the values match, then data value v is stored in the hash table (at location h) and the search procedure would return an indication of this fact. If location h of the hash table array is not empty but the data value stored in location h does not match data value v, the search procedure probes the hash table according to the probe sequence. In the case of linear probing, this is done by checking hash table locations (buckets) $h+1$, $h+2$, *etc.*, until either an empty location is found or a location storing the value v is found (or until the entire hash table has been probed [searched]). If an empty location is found, or if the entire table is probed and data value v is not found, then data value v is not stored in the hash table. If the value v is found then data value v was stored in the hash table. In either case, the search procedure would return the appropriate indication of success or failure.

56. Note that when probing a hash table for a particular data value v, as was the case when inserting data values into a hash table, one may encounter both data values that collide with data value v (*i.e.*, data values whose hash is the same as the hash of data value v), as well as data values that do not collide with data value v.

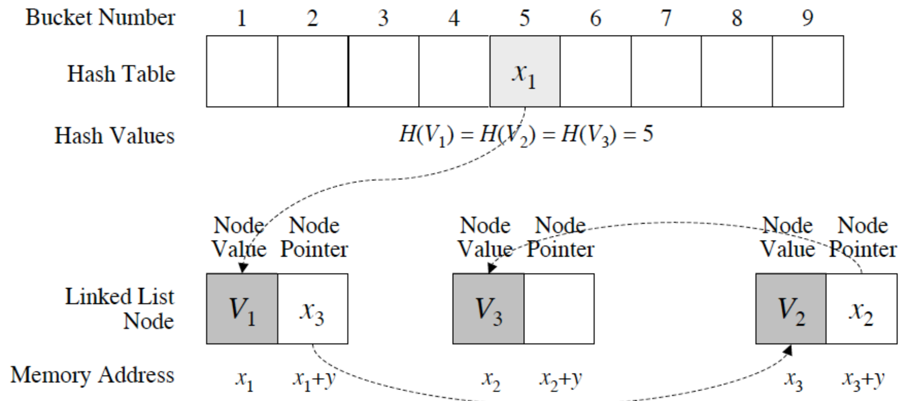
57. In the case of hashing with collision resolution based on external chaining, to determine if a data value v is stored in a hash table, the hash h of data value v is computed and

used to find the head of the hth linked list. This is done by reading the hth location of the hash table array and following the pointer stored there to locate the head of the hth linked list. If the hth linked list is empty, then data value v is not stored in the hash table and the search procedure would return an indication of this fact. If the hth linked list is not empty, then the linked list is traversed (searched) in a linear fashion to determine if the data value v is stored in the list. If the hth linked list contains the value v, then the search procedure would return an indication of this fact. If the hth linked list does not contain the value v, then the search procedure would return an indication of this fact.

c. Deletion of Data Values From a Hash Table

58. The process of deleting a data value stored in a hash table first requires that the data value to be deleted is located in the hash table. This requires that the search procedure appropriate for the collision resolution scheme in use is first used to locate the data value to be deleted. Once located, the deletion process will also depend on the collision resolution scheme in use.

59. In the case of deleting a data value from a hash table with collision resolution based on external chaining, the deletion process is equivalent to deleting an element from a linked list and thus quite simple. Consider a simple linked list associated with the hth bucket of a hash table (*i.e.*, this linked list holds data values that hash to the value h). Consider further that this linked list presently consists of three nodes storing the data values v_1, v_2, and v_3 (in that order) (*i.e.*, in this hash table, the three data values, v_1, v_2, and v_3, all hash to the value h and thus collide with one another and are stored in the external chain for the hth bucket of the hash table). The figure below illustrates this situation where the data values, v_1, v_2, and v_3, all hash to the value 5.

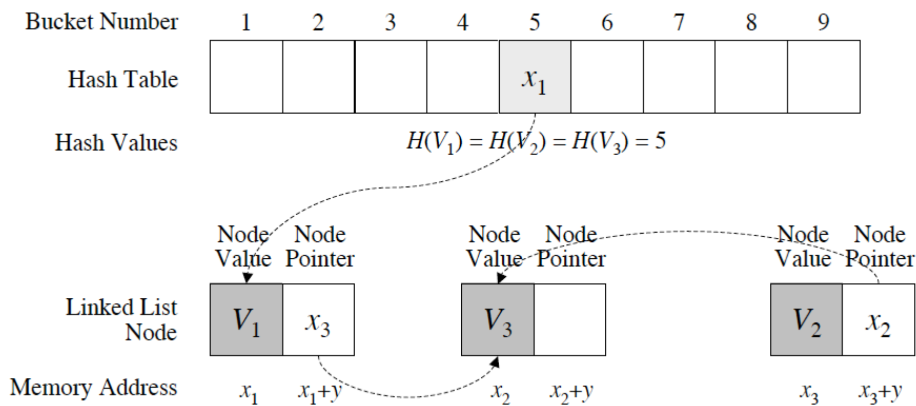


60. Assume that data value v_2 is the data value to be deleted. As a first step, the node in the linked list containing the data value v_2 would have to be located. This would be accomplished by using the search procedure described above to first compute the hash of data value v_2 (compute $H(v_2) = 5$), then access the fifth bucket of the hash table to locate the fifth linked list, and then follow the procedure for deleting a node from a linked list. This deletion procedure would consist of first traversing the fifth linked list to locate the node containing the data value v_2. During the process of locating the node in the linked list containing the data value v_2, the deletion procedure would record the location of the node in the linked list that is the predecessor in the list of the node containing the data value v_2. In this example, this means that as part of the process of locating the node containing the data value v_2, the deletion procedure would also record the location of the node containing the data value v_1.

61. Having located the node containing the data value v_2, in order to delete[6] the node, the pointer in the predecessor node in the linked list to the node containing the data value v_2 (*i.e.*, the pointer in the node containing the data value v_1) is simply set to point to the node in the linked list pointed to by the pointer in the node containing the data value v_2. That is, the pointer

[6] The Court's claim construction order construes "removing" a record from the linked list as merely adjusting the pointers in the linked list to bypass the record, and not to include

in the node containing the data value v_1 is changed to point to the node containing the data value v_3 (the node pointed to by the pointer in the node containing the data value v_2 [the node to be deleted]) as shown below. At this point the node containing the data value v_2 is no longer in the linked list because no node in the list points to it. At this point the storage occupied by the node containing the data vale v_2 would be reclaimed or otherwise made available for further use by the program. Apart from the process of locating the node containing the data value to be deleted, the process of deleting a node from a linked list can typically be performed in a single statement (a single "instruction") in a computer programming language.



62. In contrast, deleting a data value from a hash table with collision resolution based on open addressing is significantly more complex. In particular, the seemingly obvious means of deleting a data value from a hash table, namely deleting the data value stored in a bucket, will not work.[7] This is because when collision resolution is based on open addressing, a hash table location may appear in one or more probe sequences for data values that have collided other data

---

deallocation of the record. My discussion of deletion from a linked list in a hash table applies that same construction.

[7] Knuth commented on the surprising subtly and complexity of deleting from a hash table with collision resolution based on open addressing, stating "[m]any computer programmers have great faith in algorithms, and they are surprised to find that the obvious way to delete records from a scatter table [a hash table with collision resolution based on open addressing] doesn't work" (see Knuth, p. 526).

values in the hash table (appear in probe sequences for data values other than the data value to be deleted). As a result, if a data value is simply deleted from the hash table and its bucket marked as empty, a “gap” can appear in a probe sequence that will potentially result in other data values in the hash table becoming inaccessible.

63. For example, consider again the hash table with collision resolution based on linear probing from the discussion above on collisions. The table, reproduced below, stores six data values where two sets of values collide with one another (data values v_1, v_4, and v_6 collide with one another, and data values v_2 and v_5 collide with one another).

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table		V_1	V_4	V_2	V_5	V_6	V_3		

Hash Values: $H(V_1) = 2$, $H(V_4) = 2$, $H(V_2) = 4$, $H(V_5) = 4$, $H(V_6) = 2$, $H(V_3) = 7$

64. If data value v_2 were to be deleted, the location of data value v_2 would first have to be determined. Having found data value v_2 in the fourth bucket of the hash table, data value v_2 cannot be deleted by simply erasing the contents of the fourth bucket of the hash table. If a deletion procedure simply erased the contents of the fourth bucket of the hash table as shown below, then if collision resolution was based on linear probing, data values that collided with v_2, for example data value v_5, would become inaccessible. Data value v_5 would be inaccessible because a search for data value v_5 would start with location 4 of the hash table (since the hash of data value v_5 is 4). The search procedure would examine location 4 of the hash table, find that location to be empty, and, as described above, conclude (erroneously) that data value v_5 was not in the hash table.

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table		V_1	V_4		V_5	V_6	V_3		

Hash Values: $H(V_1) = 2$, $H(V_4) = 2$, $H(V_5) = 4$, $H(V_6) = 2$, $H(V_3) = 7$

65. Note further that if a deletion procedure deleted the data value v_2 by simply erasing the contents of the fourth bucket of the hash table, then data values hashing to either the second or third buckets of the hash table could similarly become inaccessible. That is, data values stored in the hash table unrelated to the deleted data value v_2 can also become inaccessible. For example, if the contents of the fourth bucket of the hash table were erased, data value v_6 in the figure above would become inaccessible. This is because the location in the hash table occupied by the (deleted) data value v_2, location 4, is on the probe sequence for data values colliding with data value v_1 (the probe sequence for data values hashing to the second bucket of the has table). If the contents of the fourth bucket of the hash table are erased, a “gap” appears in the probe sequence for data values hashing to second bucket in the hash table. Note that these are data values that did not collide with data value v_2 and thus, from a hashing perspective, are unrelated to v_2. In this case, when searching for data value v_6, the search procedure would consider hash table locations 2, 3, and 4 in turn. Upon finding location 4 of the hash table to be empty, the search procedure would again erroneously conclude that data value v_6 was not in the hash table.

66. There are two basic approaches to deleting data values in hash tables with collision resolution based on open addressing. The simplest approach is to store more information about the state of hash table locations in the hash table. Whereas previously, hash table locations could be in either one of two states, “empty” or “full,” now a hash table location can be either “empty,” “full,” or “deleted.” A hash table location being in the “deleted” state means that for purposes of inserting data values into the hash table, the location is empty and available for storage of a new data value, but that for purposes of searching for data values stored in the hash table, the location is “full,” but it contains invalid (deleted) data. That is, for purposes

of searching for data values stored in the hash table, a deleted location should simply be skipped and the probe sequence continue with the next location in the hash table (in the case of linear probing). Under this new scheme of labeling hash table locations, the figure below illustrates the state of the hash table above after the deletion of data value v_2.

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table	"empty"	V_1	V_4	"deleted"	V_5	V_6	V_3	"empty"	"empty"

Hash Values: $H(V_1) = 2$ $H(V_4) = 2$ $H(V_5) = 4$ $H(V_6) = 2$ $H(V_3) = 7$

Now, when searching for data value v_6, a linear probing search procedure will not terminate at hash table location 4 and return an indication that data value v_6 is not stored in the table. Instead, when probing hash table location 4, the search procedure will skip over hash table location 4 and continue the probe sequence. Eventually, the search procedure will probe hash table location 6, find data value v_6 and return an indication that data value v_6 is indeed present in the hash table.

67. An obvious downside of this approach to deletion is that probe sequences are now longer than need be. As a result it will take longer to find certain data values stored in the hash table. For example, when searching for data value v_5, two hash table locations, locations 4 and 5, must be probed (recall that data value v_5 hashes to the fourth bucket of the hash table and hence any search for data value v_5 necessarily would start with the fourth location in the hash table).

68. A seemingly obvious solution to the problem of probe sequences being longer than need be, is to "fill in the gap" in the probe sequence for data values hashing to the fourth bucket of the hash table by simply placing data value v_5 in the fourth bucket of the hash table (since data value v_5 hashes to the fourth bucket of the hash table), and marking the bucket in the hash table previously occupied by data value v_5 as empty. The result would be the hash table shown below.

Bucket Number	1	2	3	4	5	6	7	8	9
Hash Table	"empty"	V_1	V_4	V_5	"empty"	V_6	V_3	"empty"	"empty"
Hash Values		$H(V_1) = 2$	$H(V_4) = 2$	$H(V_5) = 4$		$H(V_6) = 2$	$H(V_3) = 7$		

69. This process of combining deletion with the moving (copying) of data values colliding with the deleted data value, solves the problem of probe sequences for data values colliding with the deleted data value being longer than they otherwise need to be. However, note that in moving data values colliding with the deleted data value, unrelated data values can now become inaccessible. For example, moving data value v_5 from location 5 in the hash table to location 4, shortens the probe sequence required to locate v_5 in the future. However, now data value v_6 that did not collide with the deleted data value v_2 is now inaccessible. Data value v_6 is now inaccessible because data value v_6 hashes to location 2 of the hash table and, under linear probing, would be searched for by probing locations 2, 3, 4, and 5 of the hash table in turn. Upon encountering empty location 5 of the hash table (the location made empty by moving data value v_5 from location 5 in the hash table to location 4) a search procedure would erroneously conclude that data value v_6 is not stored in the hash table.

70. Solving this problem is vexing because although the deletion procedure originally sought to delete data value v_2 — a data value that hashed to the fourth bucket of the hash table — the deletion procedure must identify all the data values stored in the hash table that are reached via a probe sequence that includes the fourth location in the hash table. This is a complex procedure because such data values are not easily identifiable because they are not related to data value v_2. To be sure, algorithms to delete data values from a hash table with collision resolution based on open addressing, and at the same time reduce the length of probe sequences, are well known. However, these algorithms are typically quite complex and costly in terms of execution

time. The algorithms are costly because they involve rehashing (removing and reinserting) data values stored in the hash table. Moreover, while the rehashing process is in progress, the hash table will not be available for other storage and retrieval operations (the hash table will be "offline").[8] All of this is in marked contrast to the process of deleting a data value from a hash table with collision resolution based on external chaining which simply requires a single pointer manipulation operation.

3. Hashing of Records

71. For simplicity, and ease of explanation, the forgoing has described hashing within the context of storing and retrieving individual data values. However, in practice, one typically stores collections of data in a hash table. That is, in many storage and retrieval applications, a set of related data items are stored together in a structure commonly called a *record*. For example, in a database application, an "employee record" might contain a collection of data related to a specific employee, such as the individual's name, date of birth, address, social security number, pay grade, *etc*. Just as programmers can create arrays and linked lists of individual data values, they can also create arrays and linked lists of records. Conceptually, the only difference between an array (or linked list) of individual data values and an array (or linked list) of records, is the amount of storage in memory required to store an element of the array (or a node of the linked list). Similarly, hash tables of records can be created using either an array of records (when open addressing is used) or a set of linked lists of records (when external chaining is used).

---

[8] As discussed in Section VI.A, the inventor of the '120 Patent, Richard Nemes, previously applied for a patent related to hashing with collision resolution base on open addressing via linear probing. In the earlier patent, the '495 Patent, Dr. Nemes acknowledged that then existing techniques for deletion of entries from a hash table with collision resolution based on open addressing were so expensive in time that their use would result in the hash table being unavailable for use.

72. The primary new complexity introduced when hashing records rather than individual data values is that the programmer now has to decide what to use as the input for the hash function. That is, if a record contains a set of fields (name, address, *etc*.), the programmer must select one or more of these fields to use as the input for the hash function. Whatever field or fields the programmer selects to hash on, that element (or collection of elements) is referred to as the *hash key* or a *record search key*. However, whatever is used as the hash key, as when hashing individual data values, the hash function still computes a value from 1 to n, where n is the size of the hash table (the number of locations or buckets in the hash table array).

4. State of Knowledge of Hashing in 1997

73. Search algorithms generally, and search algorithms based on hashing, were well known in the art by 1997. For example, more than twenty years earlier, in 1973, the eminent computer scientist Donald Knuth published a series of volumes on (and titled) "The Art of Computer Programming." In volume 3, "Searching and Sorting," Knuth identified and analyzed the foundational algorithms for all searching and sorting procedures, including hashing with open addressing and external chaining. Because of its comprehensive treatment of its subjects, and because of its elegant yet rigorous analyses of algorithms, ever since its publication, "The Art of Computer Programming" has been universally considered to be the seminal reference on the foundations of computer programming. For example, at the close of the twentieth century, the publication "American Scientist," published by the Sigma Xi Scientific Research Society, listed "The Art of Computer Programming" among its list of its "100 or so books that shaped a century [the twentieth century] of science."[9]

[9] "100 or so Books that Shaped a Century of Science," by Phylis and Philip Morrison, *American Scientist*, Vol. 87, No. 6, November-December 1999.

C. <u>Choosing a Collision Resolution Scheme</u>

74. Hashing is a well-known and much used technique for efficiently storing and retrieving data in a computer system. When designing a hashing scheme, a programmer must make three interrelated decisions. A programmer must decide what size hash table to use, what hash function to use, and which of the two classes of collision resolution techniques to use (collision resolution based on open addressing or collision resolution based on external chaining).

75. The following considers the problem of selecting a collision resolution scheme. Generally, the choice of collision resolution scheme can be influenced by factors such as the amount of memory to be consumed by the hash table, whether or not the hash table is expected to grow or shrink significantly over time, the complexity of the collision resolution scheme, and the frequency with which records are expected to be deleted from the hash table. Interestingly, as the following makes clear, in all cases, there are strong, clear-cut reasons for selecting the external chaining method of collision resolution over open addressing methods such as linear probing.

1. Considerations Based on Memory Usage

76. The size of the hash table is primarily determined by how many records are expected to be stored. However, if the expected number of records is either unknown or is expected to be highly variable, this may impact the choice of collision resolution scheme. External chaining allows for hash tables to grow in size with minimal effort whereas open addressing can require substantial effort to increase the size of the hash table. The difference here mirrors the difference between the effort required to increase the size of an array versus the effort required to increase the size of a linked list. Because external chaining is based on linked lists, the chains, and hence the size of the hash table, can grow by simply linking in new nodes for

new hash table records. Thus, to extend the size of a hash table with collision resolution based on external chaining requires the allocation of a new linked list node and insertion of the node into the appropriate linked list. Once the location for insertion in the linked list has been identified, the insertion process requires at most two pointer manipulations as illustrated above in Section III.A.2.

77. In contrast, increasing the size of a hash table with collision resolution based on open addressing requires allocating a new, larger array for the hash table, and then rehashing (re-inserting) each element in the old hash table into the new hash table. If the hash table is large, the overhead of rehashing every element in the old hash table can be substantial and can require that a storage and retrieval system using hashing with collision resolution based on open addressing be taken offline while the size of the hash table is increased and the contents of the old table are rehashed.

78. A second memory usage consideration concerns the size of the records to be hashed. If the records to be hashed are small, for example, if the records consist of only a single data value, then the storage required for a hash table with collision resolution based on external chaining may be greater than the storage required for a hash table with collision resolution based on open addressing. This is because nodes in the linked lists used with external chaining require memory to store the pointers to subsequent elements in the list and no such memory is required with open addressing. However, if the number of records stored in a hash table is small compared to the overall size of the table, the storage required for a hash table with collision resolution based on external chaining may actually be less than the storage required for a hash table with collision resolution based on open addressing. In this case, this is because with external chaining, nodes are only created as needed, whereas with open addressing, all of the

memory required to store the maximum number of records must be allocated when the hash table is created (even though, as discussed below, some of this memory will never be used).

79. Similarly, if the records to be hashed are large (require a lot of memory to store), then the storage required for a hash table with collision resolution based on external chaining will be less than the storage required for a hash table with collision resolution based on open addressing. This is again because with external chaining, nodes are only created as needed whereas with open addressing, all the memory required to store the maximum number of records must be allocated when the hash table is created.

80. Thus, considerations of efficient memory usage often favor performing collision resolution in a hash table using external chaining rather than open addressing.

2. Considerations Based on the Frequency of Deletions

81. If records are expected to be frequently deleted from the hash table, then collision resolution based on external chaining will be more desirable than collision resolution based on open addressing. First, as discussed above, deletions from a hash table with collision resolution based on external chaining is equivalent to deletions from a linked list and thus trivial (requiring only one pointer manipulation to bypass the deleted node in the list). In contrast, deletion from a hash table with collision resolution based on open addressing requires either the introduction and management of a new "deleted" state for hash table locations (result in longer probe sequences than necessary), or requires the use of a complex and time consuming algorithm to locate and rehash elements in the table to fill in any "gap" in probe sequences created by the deletion of a record.

82. Second, frequent deletions from a hash table with collision resolution based on open addressing, and in particular, collision resolution based on linear probing, can negatively impact the performance of searching for records in the hash table. In particular, if gaps in probe

sequences formed by the deletion of records are not filled (*i.e.*, if the complex and time consuming gap filling algorithm is not used and instead hash table locations are merely labeled as being in the "deleted" state when elements are deleted), then the time required to determine if a record is not in a hash table can be substantially increased. This is because whereas normally, when searching for a record in a hash table with collision resolution based on open addressing, upon encountering an empty hash table location, the search procedure can terminate and conclude that the sought after record is not present in the hash table. If gaps in probe sequences are not filled, and empty locations that are labeled "deleted" exist in the hash table (because those locations previously contained a record that has since been deleted), then the search procedure can no longer terminate when it encounters an empty, "deleted" location. Instead, as described above, the search procedure must continue until it finds an empty hash table location that has never contained a record. In particular, in the worst case, a search for an element not in the hash table may require a search of the entire table (*i.e.*, in effect, a linear search must be performed) even though the table may be arbitrarily close to being empty (be in a state when most hash table locations are not being used).

83. Hash tables with collision resolution based on external chaining do not suffer from the problem of degraded performance when deletions are frequent. To the contrary, because deletions from a hash table with collision resolution based on external chaining only serve to shorten the length of the external chains (the linked lists), deletions from a hash table with collision resolution based on external chaining will result in improved performance (decreased search times when searching for records not presently in the hash table).

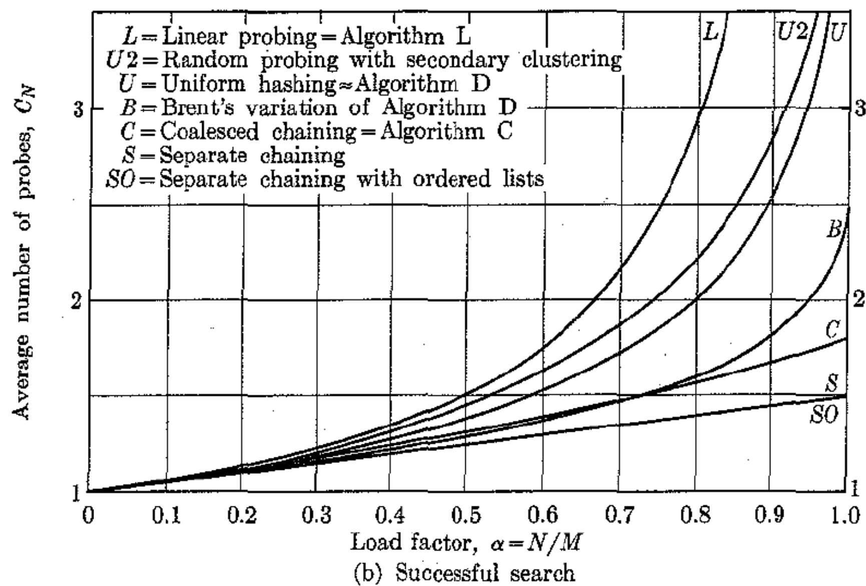
3. Considerations Based on Search Performance

84. The foregoing has shown that when deletions are frequent, collision resolution based on external chaining can provide better search performance than collision resolution based

on linear probing. However, the same is true even when records are never deleted from the hash table.

85. In his seminal work, Knuth analyzed the performance of a number of variants of the linear probing and external chaining, collision resolution schemes. In particular, Knuth analyzed the average number of probes required to find an element in a hash table as a function of collision resolution scheme and as a function of how "full" the hash table is. The "fullness" of a hash table is measured by computing the ratio of the number of records N stored in the hash table to the number of distinct values M generated by the hash function. (The value M corresponds to the number of buckets in the hash table.) This ratio, N/M, is referred to as the *load factor* of the hash table. When the load factor is 1, then in the case of open addressing, the hash table is full (every location in the hash table stores a value and no more records can be stored in the hash table). When the load factor is 0, the hash table is completely empty.

86. The plot below, taken from Knuth (see, Knuth, p. 539), shows how the average number of probes required to find a record stored in a hash table varies as a function of the load factor for a variety of collision resolution schemes.



(b) Successful search

In this figure, the leftmost line, the line labeled “L,” shows the average number of probes required to find a record stored in the hash table when collision resolution is based on linear probing. The rightmost line, the line labeled “S” (overlaid on top of the line labeled “SO”), shows the average number of probes required to find a record stored in the hash table when collision resolution is based on external chaining (called “separate chaining” in Knuth). This analysis clearly shows that for all load factors, the average number of probes required to find a record that is present in the hash table when collision resolution is based on external chaining is smaller than the average number of required probes when collision resolution is based on linear probing. More importantly, Knuth’s analysis shows that the average number of probes required to find a record present in the hash table when collision resolution is based on linear probing grows exponentially as the load factor increases, while the average number of probes required to find a record present in the hash table when collision resolution is based on external chaining grows only linearly (with very shallow slope). These data suggest that one would expect dramatically different search time performance when the load factor goes above 50% (when the has table is more than half full).

87. A tabular view of this same phenomena can be found in data structures textbooks, such as the text “Data Structures and Program Design,” Second Edition, by R.L. Kruse, Prentice Hall, Englewood Cliffs, NJ, 1987 (hereafter the “Kruse text” or simply “Kruse”). The table below, from Kruse, shows empirical data from a study of hashing 900 pseudorandom numbers and then searching for various values (see, Kruse, p. 213). Three hash tables were constructed, each using a different collision resolution method, but including external chaining (called “Chaining” in the table below) and linear probing (called “Linear probes” in the table below).

The table shows the number of probes required to search for an element as a function of the load factor of the hash table.

Load factor	0.1	0.5	0.8	0.9	0.99	2.0
Successful search						
Chaining	1.04	1.2	1.4	1.4	1.5	2.0
Open, Quadratic probes	1.04	1.5	2.1	2.7	5.2	—
Linear probes	1.05	1.6	3.4	6.2	21.3	—
Unsuccessful search						
Chaining	0.11	0.53	0.78	0.90	0.99	2.04
Open, Quadratic probes	1.13	2.2	5.2	11.9	126.	—
Linear probes	1.13	2.7	15.4	59.8	430.	–

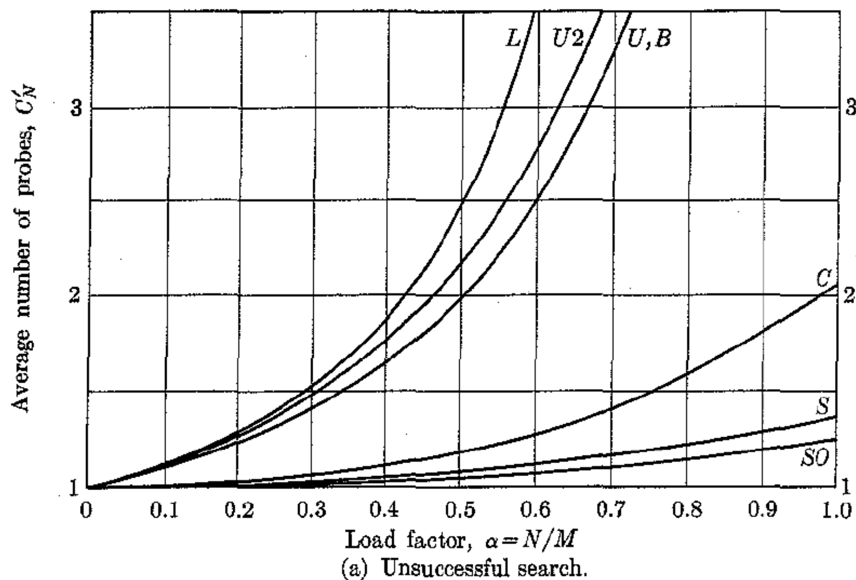
Figure 6.14. Empirical comparison of hashing methods

For a successful search (a search for a record that is present in the hash table), these data confirm that searching in a hash table with collision resolution based on external chaining always requires fewer probes than a search in a hash table with collision resolution based on linear probing. In particular, these data confirm that as the load factor of the table increases beyond 50%, the number of probes required to locate an element in a hash table with collision resolution based on linear probing grows exponentially while the number of probes required to locate an element in a hash table with collision resolution based on external chaining grows linearly (again with very shallow slope).

88. For an unsuccessful search (a search for a record that is not present in the hash table), the Kruse data show that the disparity in search performance (the disparity in the number of probes) is even more pronounced. In particular, as the load factor of the table approaches 100% (as the hash table becomes full), the number of probes required to determine if a record is not in a hash table with collision resolution based on linear probing is more than 400 times that required for a hash table with collision resolution based on external chaining. This is a profound difference.

89. The profound difference in the number of probes required to determine that a record is not stored in a hash table when collision resolution is based on linear probing versus

external chaining is also shown in Knuth's analysis. Below is a plot from Knuth showing how the average number of probes required to determine that a record is not present in a hash table varies as a function of the load factor (see, Knuth, p. 539). The interpretation of the labels on the lines of the plot are the same as in the plot above (*i.e.*, the leftmost line, the line labeled "L," shows the average number of probes required for an unsuccessful search when collision resolution is based on linear probing, and the second line from the right, the line labeled "S," shows the average number of probes required for an unsuccessful search when collision resolution is based on external chaining).



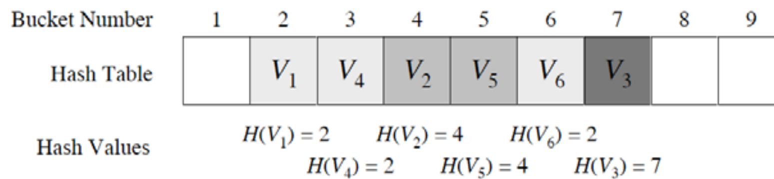
(a) Unsuccessful search.

This analysis clearly shows that the average number of probes required to determine if a record is not present in a hash table when collision resolution is based on external chaining is *always* smaller than the average number of required probes when collision resolution is based on linear probing. More importantly, Knuth's analysis shows that the average number of probes required to determine if a record is not present in the hash table when collision resolution is based on linear probing grows exponentially as the load factor increases, while the average number of probes required to determine if a record is not present in the hash table when collision resolution

is based on external chaining grows only slightly worse than linearly (but again with a very shallow slope). These data suggest that one would expect dramatically different performance (search times) when the load factor goes above 25% (when the hash table is more than one quarter full). And indeed, the Kruse study above confirms this expectation.

90. A key difference in the search performance between collision resolution based on linear probing and collision resolution based on external chaining is a phenomenon known as *clustering*. In hash tables with collision resolution based on linear probing, as the load factor increases (as the table becomes more full), it becomes likely that records fill numerous adjacent locations in the hash table. When this occurs, collision resolution requires a higher number of probes because more hash table locations now have to be examined before an empty location is found. Because more probes are required, search times increase.

91. For example, the example hash table previously considered, with collision resolution based on linear probing, illustrates an instance of clustering. As shown below, because data values hashing to the second and fourth hash table buckets collided with data values already stored in the table, and because these data values are generally hashing "close" to one another, a contiguous sequence of hash table locations (locations 2 through 7) are filled. The presence of this "cluster" of records in the hash table means that searches for records hashing to locations 2 through 6 of the table will require several probes before generating a result (an indication of success or failure).



92. Collision resolution based on external chaining is immune from the clustering effect. As a result, search time performance in hash tables with collision resolution based on

external chaining is largely unaffected by the load factor of a hash table (as is clearly evident from the data shown above from Knuth and Kruse).

93. Because of the problems caused by clustering in hash tables with collision resolution based on linear probing, the load factor for such hash tables is typically required to always be smaller than a comparable hash table with collision resolution based on external chaining. When the load factor exceeds a certain threshold, hash tables with collision resolution based on linear probing are typically taken off-line (made unavailable for searching, insertion, or deletion), the size of the table expanded, and the records stored in the table rehashed. These are all expensive and time-consuming operations. A typical load factor threshold that would trigger an expansion and reconstruction of the hash table would be a value in the range of 50-80%. Thus, potentially, when a hash table with collision resolution based on linear probing is only half full, the table may be taken off-line and expanded. In contrast, as discussed above, hash tables with collision resolution based on external chaining can always be expanded without requiring rehashing. As a result, hash tables with collision resolution based on external chaining can tolerate load factors greater than 1.0. (Load factors for hash tables with collision resolution based on open addressing probing can never exceed 1.0 [and as discussed above, typically must stay below a value in the range of 0.5 – 0.8].)

4. Considerations Based on Implementation Complexity

94. Collision resolution based on external chaining is simpler to implement than collision resolution based on open addressing. Implementing collision resolution based on external chaining simply requires implementing procedures to insert, delete, retrieve, and search a linked list (note that the insert, retrieve, and delete procedures will require first searching the list). These are elementary procedures. In fact, notably, in his description of external chaining, Knuth remarks:

> This method [hashing with external chaining] is a straightforward combination of techniques we have discussed before, so we do not need to formulate a detailed algorithm for chained scatter [hash] table. (See, Knuth, p. 513, emphasis added.)

Thus, Knuth teaches that knowledge of linked lists (a technique previously discussed in Knuth) is sufficient to implement external chaining. As a result, Knuth does not provide a reference implementation of hashing with external chaining (Knuth does provide reference implementations of forms of external chaining that are more sophisticated than the schemes considered herein). In contrast, Knuth does provide a reference implementation of collision resolution based on the simplest form of open addressing, namely linear probing. This was likely done because even open addressing based on linear probing is conceptually and algorithmically more complicated than collision resolution based on external chaining.

95. In addition, as discussed previously, deletions of records from a hash table are significantly easier to implement when collision resolution is based on external chaining than when collision resolution is based on linear probing. Deletion[10] of records from a hash table with collision resolution based on external chaining is equivalent to deletion from a linked list and thus requires only a single pointer manipulation (once the node to be deleted from the list has been located). Deletion of records from a hash table with collision resolution based on linear probing requires either introducing the notion of a "deleted" state into the hash table (and thus requires changes to the procedures to insert into and search the hash table), or requires the implementation of a complex and time consuming algorithm to fill gaps in probe sequences caused by the deletion of records. (Deletion of records from a hash table with collision resolution

[10] The Court's claim construction order construes "removing" a record from the linked list as merely adjusting the pointers in the linked list to bypass the record, and not to include deallocation of the record. My discussion of deletion from a hash table with external chaining applies that same construction.

based on linear probing also requires that the node to be deleted first be located within the hash table.)

96. In fact, authors such as Kruse often describe the complexity of supporting deletion of records under external chaining versus linear probing (or more generally open addressing) in stark terms. For example, the Kruse text teaches that the method of deleting records from a hash table with collision resolution based on open addressing by marking hash table locations as "deleted" will make the open addressing algorithms "somewhat more complicated and bit slower" (see, Kruse, p. 206). In addition, Kruse next states:

> With the methods we have so far studied for hash tables [open addressing], deletions are indeed awkward and should be avoided as much as possible. (See, Kruse, p. 206.)

In contrast, with respect to deletions from a hash table with collision resolution based on external chaining, when discussing the advantages of external chaining Kruse writes:

> Finally, deletion becomes a quick and easy task in a chained hash table [a hash table with collision resolution based on external chaining]. Deletion proceeds in exactly the same way as deletion from a simple linked list. (See, Kruse, p. 206.)

97. In summary, collision resolution based on external chaining has numerous well-known and clear-cut advantages that in many instances would lead one to select this method of collision resolution over open addressing. Collision resolution based on external chaining can use less memory, provide faster searching, allow the size of the hash table to easily grow dynamically, and be easier to implement than collision resolution based on linear probing. This is particularly true if support for deletion of records from hash tables is an important concern. In the abstract, if support for deletion of records from hash tables is an important concern then external chaining would be the collision resolution algorithm of choice. Additionally, in 1997, these facts were well known to formally trained (*e.g.*, college educated) computer programmers and would have been common knowledge among such individuals. Such individuals would have

been exposed to the material presented here no later than their freshman or sophomore year in college when they took a data structures or algorithms course.

D. <u>Overview of the '120 Patent</u>

98. U.S. Patent No. 5,893,120 is titled "Methods and Apparatus For Information Storage and Retrieval Using a Hashing Technique With External Chaining and On-the-Fly Removal of Expired Data." The application that resulted in the '120 Patent was filed on January 2, 1997. The '120 Patent was issued on April 6, 1999 to Richard Nemes.

99. The '120 Patent concerns a method and apparatus for performing storage and retrieval in an information storage system of records where at least some of the records automatically expire. The '120 Patent explains that "[s]ome forms of information are such that individual data items, after a limited period of time, become obsolete, and their presence in the storage system is no longer needed or desired" (see, '120 Patent, col. 2, lines 7-10). Expired records remaining in the storage system are a potential problem because their presence may increase the time required to locate non-expired records (see, '120 Patent, col. 2, lines 16-19). The goal of the '120 Patent is to remove such expired records to reclaim the storage occupied by the records so as to maintain fast access to the (non-expired) records (see, '120 Patent, col. 2, lines 19-21).

100. The specification of the '120 Patent teaches the use of hashing with the external chaining method for resolving collisions in the hash table, and linked lists generally, for storing and retrieving information (see, *e.g.*, '120 Patent, Abstract, col. 1, lines 20-22). As acknowledged in the '120 Patent, the use of hashing with external chaining for storing and retrieving information in an information storage system was well known prior to the filing of the

application for the ‘120 Patent (see, *e.g.*, ‘120 Patent, col. 1, lines 42-46, col. 1, line 65 – col. 2, line 6).

101. The ‘120 Patent is generally directed to information storage and retrieval systems comprising either linked lists of records, or hash tables for storing and retrieving records where collision resolution is based on external chaining. In both cases, at least some of the records automatically expire (see, *e.g.*, ‘120 Patent, col. 3, lines 4-7). When the linked list is accessed, the methods of the ‘120 Patent identify at least some of the automatically expired records, and remove at least some of the automatically expired records (see, *e.g.*, ‘120 Patent, col. 3, lines 7-11). As an example, the ‘120 Patent teaches that expiration of a record could be determined by comparing a timestamp maintained in the record to the current time-of-day value maintained by the computer (see, ‘120 Patent, col. 6, lines 9-11).

E. **The Prosecution History**

102. I have reviewed the prosecution history of the ‘120 patent and the prior art cited against the claims of the ‘120 patent by the United States Patent and Trademark Office (“USPTO”) during prosecution. During prosecution, the USPTO did not consider the prior art I address in this report, with the exception of the ‘495 Patent. Although the Knuth, Kruse, and Stubbs references were cited to the USPTO, they were not cited in any office actions by the USPTO and it is not clear whether they were ever considered by the USPTO. The USPTO also did not have the benefit of the testimony of prior art witnesses Daniel McDonald or Dr. Shawn Ostermann, nor did the USPTO have the benefit of the testimony of Dr. Richard Nemes, the inventor of the ‘120 Patent.

103. Furthermore, Dr. Nemes did not inform the USPTO that external chaining and open addressing are the two principle collision resolution techniques as described in the Knuth,

Kruse and Stubbs references. Nor did Dr. Nemes inform the USPTO that external chaining is an obvious design alternative over linear probing. This is despite the fact that Dr. Nemes testified that he did not know of any collision resolution techniques for removing expired records other than linear probing and external chaining (see, Nemes Dep. Tr., p. 194). Had Dr. Nemes told the USPTO that external chaining was an obvious design alternative to those of ordinary skill in the art in view of linear probing, I believe the USPTO would have had sufficient basis for rejecting the '120 patent on obviousness grounds.

IV. LEGAL STANDARDS

104. I understand that under Section 102 of the Patent Act, claims may be invalidated for lack of novelty or loss of rights. I have been informed by counsel that a claimed invention is invalid for anticipation or lack of novelty when all of the limitations of the claim as construed by the Court are present in a single prior art reference. I understand, however, that all limitations of the claim need not be shown directly so long as all limitations are necessarily present in the single prior art reference and thus are inherent.

105. Section 102 of the Patent Act provides that "[a] person shall be entitled to a patent unless . . . (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for patent, or . . . (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of the application for patent in the United States, or . . . (g) . . . (2) before such person's invention thereof, the invention was made in this country by another inventor who had not abandoned, suppressed, or concealed it."

106. I have been informed by counsel that the evidence must be "clear and convincing" for a claim to be found invalid.

107. I understand that a claim is obvious in light of the prior art if the difference or differences between the claimed subject matter and the prior art are such that the subject matter as a whole would have been obvious, at the time the invention was made, to a person having ordinary skill in the art.

108. I understand that that in KSR Int'l Co. v. Teleflex, Inc., 127 S. Ct. 1727 (2007), the Supreme Court provided an outline for analyzing obviousness. The Supreme Court rejected the Federal Circuit's "rigid" application of the "teaching, suggestion, or motivation" test for obviousness in favor of an "expansive and flexible approach" using "common sense." I also understand that the Supreme Court explained that under the correct analysis, any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed. I also understand that the Supreme Court explained that "[t]he combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results." I further understand that the Court pointed to other factors that may show obviousness. These factors included the following principles:

- a combination that only unites old elements with no change in their respective functions is unpatentable. As a result, the combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results,
- a predictable variation of a work in the same or a different field of endeavor is likely obvious if a person of ordinary skill would be able to implement the variation,
- an invention is obvious if it is the use of a known technique to improve a similar device in the same way, unless the actual application of the technique would have been beyond the skill of the person of ordinary skill in the art. In this case, a key inquiry is whether the improvement is more than the predictable use of prior art elements according to their established functions,

- an invention is obvious if there existed at the time of invention a known problem for which there was an obvious solution encompassed by the patent's claims.
- inventions that were "obvious to try" — chosen from a finite number of identified, predictable solutions, with a reasonable expectation of success — are likely obvious,
- known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art, and
- an explicit teaching, suggestion, or motivation in the art to combine references, while not a requirement for a finding of obviousness, remains "a helpful insight" in determining upon which a finding of obviousness may be based.

109. Finally, I understand that even if a claimed invention involves more than substitution of one known element for another or the application of a known technique to a piece of prior art ready for improvement, the invention may still be obvious. I also understand that in such circumstances courts may need to look to interrelated teachings of multiple patents; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by a person having ordinary skill in the art to determine if the claimed invention is obvious.

110. I understand that a plaintiff can rebut a showing of obviousness over the prior art by showing "secondary considerations" of non-obviousness and that it is plaintiff's burden to make a showing of secondary considerations. Because it is plaintiff's burden to show secondary considerations, I reserve my right to address any such secondary considerations in a supplemental report or at trial.

V. DEFINITION OF A PERSON OF ORDINARY SKILL IN THE ART

111. It is my opinion that a person of ordinary skill in the art would have a Bachelor of Science degree in computer science or computer engineering, including practical experience writing computer programs, or the equivalent.

112. Such a person in January 1997 would know all of the material discussed in the Background section of this report (Section III) and would have no difficulty implementing an information storage and retrieval system using hashing with collision resolution based on either open addressing or external chaining. Such a person of ordinary skill would also have no difficulty modifying or adapting an existing implementation of an information storage and retrieval system using hashing with collision resolution based on either open addressing or external chaining to change the collision resolution scheme in use.

VI. THE PRIOR ART

113. The following summarizes the prior art systems and references that are relevant to my analysis of the validity of the '120 Patent.

A. The Nemes '495 Patent

114. The inventor of the '120 Patent, Richard Nemes, had previously filed for, and received, a first patent related to an information storage and retrieval system employing on-the-fly deletion of expired records from a hash table. U.S. Patent 5,121,495 ("the '495 Patent"), entitled "Methods and Apparatus for Information Retrieval Utilizing Hashing Techniques," was filed on October 31, 1989, and issued on June 9, 1992.

115. The '495 Patent is generally directed to the same problem as the '120 Patent: using hashing techniques in an information storage and retrieval system and on-the-fly removal of automatically expiring records from the hash table. However, whereas the '120 Patent considers hashing with collision resolution based on external chaining, the '495 Patent considers hashing with collision resolution based on linear probing. As explained above, there are only two classes of collision resolution schemes: collision resolution based on open addressing and collision resolution based on external chaining. Within the class of collision resolution schemes

based on open addressing, collision resolution based on linear probing is the exemplar method. Moreover, as discussed in Section III.C above, it is widely accepted that of the two basic methods of collision resolution, (1) collision resolution based on external chaining is significantly easier to understand and to implement (and in many case provides superior memory and processing time performance), and (2) when deletions of records from a hash table are of concern, collision resolution based on external chaining provides significant advantages and in many cases is the preferred technique. For at least these reasons, as explained in more detail below, given the teachings of the '495 Patent, a person of ordinary skill in the art would have been motivated to adapt the teachings of the '495 Patent to hashing with collision resolution based on external chaining. Moreover, it would have been a trivial matter for a person of ordinary skill in the art to adapt the teachings of the '495 Patent to hashing with collision resolution based on external chaining.

116. Of note is the fact that significant portions of the '495 Patent were copied into both the specification, figures, and claims of the '120 Patent. In my opinion this is evidence that there is little fundamental difference between the '495 Patent and the '120 Patent beyond the former's use of collision resolution based on linear probing and the latter's use of external chaining. Indeed I understand that when Bell Communications Research Inc. ("Bellcore"), the assignee of the '495 Patent originally prepared the application for the '495 Patent, a version of the application from January 1988 included the paragraph:

> It is to be understood that the present invention will be described in connection with linear probing with open addressing only for convenience and because such a collision-resolution strategy is very commonly used. The techniques of the present invention can just as readily applied [sic] to such other forms of collision-resolution strategies by modifications readily apparent to those skilled in the art. (See TELECORDIA00000253.)

Thus, at some point during the preparation of the application for the '495 Patent, Bellcore understood that the techniques of the '495 Patent could be readily applied to other forms of collision resolution. I concur fully with this statement.[11] There is no question that a person of ordinary skill in the art at the time of the application for the '495 Patent (nearly 10 years before the application for the '120 Patent) could readily adapt the teachings of on-the-fly removal of automatically expiring records from a hash table in the '495 Patent to hashing with collision resolution based on external chaining.

B. The Nemes '499 Patent

117. The inventor of the '120 Patent, Richard Nemes, also had previously filed for, and received, an additional patent related to an information storage and retrieval system employing hashing. U.S. Patent 5,287,499 ("the '499 Patent"), entitled "Methods and Apparatus for Information Storage and Retrieval Utilizing a Method of Hashing and Different Collision Avoidance Schemes Depending Upon Clustering in the Hash Table," was filed on May 16, 1991, and issued on February 15, 1994. The '499 Patent is generally directed to a method and apparatus for information storage and retrieval using a hash table where at certain times collision resolution based on external chaining for resolving collisions is used and at other times the collision resolution based on linear probing for resolving collisions is used. The disclosures in the '499 Patent confirm that techniques for hashing with collision resolution based on external chaining were known to Dr. Nemes before 1997.

118. The Nemes '499 Patent confirms that deletion of records from an external chain of a hash table with is easy:

[11] I only became aware of the Bellcore draft application of the '495 Patent in late December 2010. Seeing this application confirmed my long-standing opinion that the teachings of the '495 Patent were readily adaptable to hashing with collision resolution based on external chaining.

> If it is determined in decision box 63 that the contents of the cell is a list pointer, box 64 is entered where the record to be deleted is removed from the linked list. This is easily accomplished by adjusting the pointer in the chain just before the record to be deleted to point the [sic] the record to deleted. (See, '499 Patent, col. 8, lines 53-58, emphasis added.)

C. NRL BSD Key Management Source Code

119. In 1995, the US Naval Research Laboratory (NRL) developed and publicly released a reference implementation of the IPv6 protocol (the sixth version of the internetworking protocol IP) for the open source Unix operating system known as "4.4BSD-Lite." By way of background, IP is the network protocol used by computers on the Internet to deliver data (messages) between each other. (In this context, a protocol is a set of rules for exchanging messages between computers.)

120. The NRL IPv6 reference implementation is in the form of a series of files of computer source code written in the C programming language. The NRL IPv6 reference implementation analyzed in this report is from the software release labeled *ipv6-dist-domestic* and consists primarily of the files *ipv6-dist-domestic/sys.common/netinet6/key.{h,c}*. The files are publicly available on the Internet and were obtained from the *ftp* site *ftp://ftp.ripe.net/ipv6/nrl*. I understand that these files have been produced as documents labeled DEF7942 – DEF7974. The files carry a copyright date of August 1995 and a file modification data of September 28, 1995. This is an indication that the files downloaded have not been modified since September 28, 1995. Daniel McDonald, one of the authors of the NRL IPv6 reference implementation, testified in his deposition that the code was published in late 1995 (see, McDonald Depo Tr., p. 29, line 14 to p. 30, line 14). I believe that the *key.c* and *key.h* files in the initial release of the NRL IPv6 reference implementation analyzed below were publicly available by late 1995.

121. Subsequent to the initial release of the NRL IPv6 code, the NRL published a second version of the code, known as "Alpha-2." According to Mr. McDonald, this code was released in January 1996 and provided to MIT for publication (see, McDonald Depo Tr., p. 31, line 23 to p. 33, line 15). Jeffrey Schiller, an employee of MIT, testified that he received the NRL code in January 1996 and would have published it soon thereafter (see, Schiller Depo Rough Tr., p. 39, line 18 to p. 40, line 22). I believe that the files *key.c* and *key.h* in the "Alpha-2" release of the NRL IPv6 code were available no later than January 1996. I understand that the files in the "Alpha-2" release have been produced as documents labeled YAHOO507259. I have reviewed the files in the "Alpha-2" release and have concluded that these files are substantively identical to the files in the original release with respect to the code relevant to my analysis. As a result, the analysis I have done on the original NRL IPv6 code below applies equally to the "Alpha-2" release that was available as of January 1996.

122. The NRL IPv6 implementation included an implementation of the Internet Protocol security protocol suite "ipsec" ("IP security"). Ipsec is a protocol (a set of rules for the exchange of messages) that enables a pair of computers to establish a secure, logical communications channel over the Internet by encrypting the messages exchanged over this logical channel.

123. The NRL ipsec implementation included a new operating system interface for cryptographic key management. This interface, part of a new BSD "protocol family" called PF_KEY, was to be used by programmers who developed distributed applications where security was achieved by encrypting the data in messages sent by the application. The PF_KEY interface provided a set of functions for programmers to manage the encryption keys ("encryption passwords") used in their application. The PF_KEY interface was designed in accordance with a

then proposed security architecture for the IP protocol suite (see, RFC 1825, "Security Architecture for the Internet Protocol," August 1995).

124. Part of the NRL ipsec implementation consists of a process within the BSD operating system called the "key engine." The key engine sends messages to processes (applications) that are endpoints of some secure communication channel. Specifically, the key engine sends "key acquire" messages to applications directing the applications to acquire a new encryption key (obtain a new "security association") for use in securing the communications with other processes. To perform this messaging function, the key engine maintains a linked list of records called an "acquirelist." Each record on the acquirelist represents (stores information about) a process that is associated with an encryption key. When the key engine attempts to send a key acquire message to a process, the key engine first traverses the acquirelist linked list to check whether or not a key acquire message has recently been sent to the process. In the course of traversing the acquirelist, the key engine also checks to see if any entries in the list of records have expired. Records on the acquirelist contain an expiration time (a time of day) that is set when a record is added to the acquirelist. Like the '120 Patent, the NRL ipsec implementation uses timestamps to determine if records in the linked list have expired: once the current time advances past a record's expiration time, the record in the acquirelist is considered to have expired and be obsolete. Thus, if expired records are found during the traversal of the acquirelist, the key engine, deletes those records (removes the records from the linked list and deallocates the storage occupied by the record) (see, NRL BSD source code, file *key.c*, lines 1423-1460). For these reasons, as the analysis below will show, the NRL ipsec implementation teaches the same method of on-the-fly removal of expired data from a linked list as the '120 Patent.

125. Of note is the fact that the September 1995 NRL IPv6 implementation was termed an "alpha" release by its developers (see, NRL BSD source code, file README). The developers stated that the state of the implementation was "complete enough to use for experimenting but it is not entirely complete" (see, file README). This is consistent with the commonly accepted notion of an alpha release in the software development community: the software is capable of functioning at some minimal level but it is not complete. Often it is the case that certain functions may have been implemented quickly and that further tuning or rewriting of the implementation may be necessary.

126. This situation appears to have been the case with the NRL developer's implementation of on-the-fly removal of expired data from the acquirelist linked list. In the source code for the traversal of the acquirelist, the following comment (a note to the reader) appears just prior to the code that removes expired records:

```
/*
 *  Since we're already looking at the list, we may as
 *  well delete expired entries as we scan through the list.
 *  This should really be done by a function like key_reaper()
 *  but until we code key_reaper(), this is a quick and dirty
 *  hack.
 */
```

(See, NRL BSD source code, file *key.c*, lines 1446-1452.)

As an experienced software developer, this comment tells me that the NRL developers did not view their on-the-fly deletion as anything particularly deep or novel. Their observation that "[s]ince we're already looking at the list, we may as well delete expired records as we scan through the list" indicates to me that the on-the-fly removal of expired records idea was an obvious design choice to consider. Moreover, the fact that the developers thought their implementation to be a "quick and dirty hack," further communicates that there is nothing special or complex about this implementation. (In the computer science community, as a noun,

"hack" is a colloquialism for, among other things, "the first thing that a programmer thought to do.")

127. The NRL BSD source code also contains an implementation of a hash table called keytable (see, NRL BSD source code, file *key*.c, line 122). The keytable hash table uses external chaining to resolve collisions, as evidence by the fact that the nodes of the keytable, which are defined as key_tblnode structures in *key.h*, contain a pointer to the next record in a linked list (see, NRL BSD source code, file *key.h*, lines 162-68).

128. The NRL BSD source code also contains a number of standard functions to search, insert, retrieve, and delete records from the keytable hash table. The function *key_search()* is used to search the keytable for a particular entry with a matching type, source address, destination address, and a parameter known as the security parameter index (spi) (an index into a security association database) (see, NRL BSD source code, file *key.c*, lines 648-685). If a match is found, the function returns a pointer to the matching record. Otherwise, the function returns a null pointer, indicating that the search has failed.

129. The function key_add() is used to insert a record into the keytable hash table (see, NRL BSD source code, file *key.c*, lines 728-852). Before inserting the record, the function uses the *key_search()* function to search the table to see if the record already exists in the hash table (see, NRL BSD source code, file *key.c*, lines 782-786). To do this, the *key_add()* function first calculates a hash value, then calls *key_search()*, passing the hash value into the *key_search()* function. If *key_search()* returns null, indicating that the record was not found, the *key_add()* function then adds a new entry at lines 796-818 by allocating memory for the record and copying the values into the new record using the function *key_addnode()* (see, NRL BSD source code, file *key.c*, lines 796-818, 688-725).

130. The function *key_get()* is used to retrieve a record from the keytable hash table (see, NRL BSD source code, file *key.c*, lines 855-890). The *key_get()* function uses the *key_search()* function to search the table for the record to be retrieved (see, NRL BSD source code, file *key.c*, line 883). Like the *key_add()* function, the *key_get()* function calculates a hash value to be passed into the *key_search()* function (see, NRL BSD source code, file *key.c*, lines 881-883). If *key_search()* finds the record being retrieved, it returns a pointer to the record, which is used by *key_get()* to copy the security association stored in the record. The *key_get()* function then returns 0, indicating success (see, NRL BSD source code, file *key.c*, lines 884-888). If *key_search()* fails to find the record, *key_get()* will return -1 indicating that the record was not found (see, NRL BSD source code, file *key.c*, lines 888-89).

131. The function *key_delete()* is used to delete a record from the keytable hash table (see, NRL BSD source code, file *key.c*, lines 987-1087). Like *key_add()* and *key_get(),* key_delete uses the *key_search()* function to search the table for the record to be retrieved, and calculates a hash value to be passed into the *key_search()* function (see, NRL BSD source code, file *key.c*, lines 1013-18). If *key_search()* finds the record, it is removed from the hash table by adjusting the pointers in the linked list to bypass the record and the *key_delete()* function returns 0, indicating success (see, NRL BSD source code, file *key.c*, lines 1020, 1031, 1084). Otherwise, the *key_delete()* function returns -1, indicating failure.

132. It would have been obvious to one of ordinary skill in the art that the on-the-fly removal of expired records that occurs in the *key_acquire()* function from the key_acquirelist linked list could have been combined with the *key_search()* function, resulting in an on-the-fly removal of expired records from a hash table with external chaining that would occur during insertion, retrieval, and deletion operations.

133. Such a combination would have been easily implemented by one of ordinary skill in the art. It could have been accomplished by simply importing the code in the *key_acquire()* function that checks, during the traversal of the linked list, whether each record is expired and removes it if it is, into the *key_search()* function (see, NRL BSD source code, file *key.c*, lines 1445-57). The result of such a combination would have been predictable – *i.e.* it would have resulted in the same on-the-fly removal of expired records in a linked list in a hash table instead of a free-standing linked list.

134. A person of ordinary skill in the art would have been motivated to combine these two techniques if faced with the need to implement an information storage and retrieval system that was expected to deal with a large number of records (on the order of hundreds or more) and that contained expiring records that needed to be removed from the system. For example, if one had expected the key_acquirelist to need to store hundreds of records, the obvious solution would have been to use a hash table instead of a linked list because of the benefits in search speed that come with a hash table.

135. The result of importing the on-the-fly removal code contained in the key_acquirelist into the *key_search()* function would have been a search function that is essentially identical to the Alternate Version of the Search Table Procedure found in the pseudo-code of the '120 Patent (see, '120 Patent, cols. 11-14). That *key_search()* function would then be utilized by the *key_add(), key_get(),* and *key_delete()* functions to insert, retrieve and delete records and, at the same time, remove expired records from the accessed linked list of records.

136. The technique of hashing with external chaining that was implemented in the keytable structure and the *key_search(), key_add(), key_get(),* and *key_delete()* functions is also disclosed in any number of basic computer science textbooks dealing with data structures and

algorithms. For example, Knuth discloses algorithms for hashing with "collision resolution by 'chaining'" in Section 6.4 on Hashing (see, Knuth pp. 506-518). Kruse discloses hashing in Section 6.5, including a discussion of collision resolution by chaining in Section 6.5.4 (see, Kruse, pp 198-215). This section includes Pascal algorithms for retrieving and inserting into a hash table with external chaining (see, Kruse p. 208). The textbook "Data Structures with Abstract Data Types and Pascal" by D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing Co., Monterey, California, 1985, hereafter "Stubbs," also discloses hashing with external chaining (see Stubbs pp. 310-36, Section 7.4, "Hashed Implementations," describing hashing). This section includes a section on "External Chaining" that describes the algorithm for collision resolution by external chaining (see Stubbs. pp. 324-25).

137. Just as combining the on-the-fly removal of expired records disclosed in the *key_acquire()* code and the hash table implementation of keytable would have been obvious, it would have been obvious to combine the NRL BSD *key_acquire()* code with the methods of hashing with external chaining that were well known in the art and disclosed in the Knuth, Kruse, and Stubbs textbooks for the same reasons.

D. <u>The GCache Software from Purdue University</u>

138. In 1991, Doug Comer and Shawn Ostermann of Purdue University developed and released software, known as "GCache," for general-purpose cache management as part of the Xinu operating system. The Xinu operating system is a Unix-like operating system developed for educational and research purposes at Purdue University in the late 1980s and early 1990s (Xinu is "Unix" spelled backwards and allegedly stands for "Xinu Is Not Unix").

139. The GCache software is single file of computer source code written in the C programming language. The version of GCache analyzed in this report is from the XINU software release publicly available on the Internet at the *ftp* site *ftp://ftp.cs.purdue.edu*

/pub/comer/XINU-SPARC.TAR.Z. In this distribution, the GCache software is found at the path *XINU-SPARC/xinu/src/sys/sys/archindep/gcache.c* and *XINU-SPARC/xinu/src/sys/h/gcache.h*. The file *gcache.c* carries an internal date stamp of October 21, 1991, and a file modification date of December 26, 1993. This is an indication that the files downloaded have not been modified since December 26, 1993. I understand that this file was produced in this litigation as the document labeled RHT-BR00015824. The GCache software was also described in a Purdue University, Department of Computer Science Technical Report titled "GCache: A Generalized Caching Mechanism," by D. Comer and S. Ostermann (the developers of the GCache source code), dated November 1991 and revised March 1992 (the "GCache paper"). I understand that the GCache paper was produced in this litigation as RHT-BR00015539. Dr. Shawn Ostermann, the primary author of GCache testified in his deposition that the software was available publicly at least as of 1994 (see, Ostermann Depo. Tr. p. 49, lines 2-20). I believe that the computer source code file *gcache.c* analyzed below was publicly available no later than 1994 and thus I understand qualifies as prior art to the '120 Patent.

140. The GCache software provided general facilities to create and manage caches. In an operating system, a cache is a copy of data, typically data read from a device, that is held in main memory (RAM) where it can be accessed more efficiently. The GCache software supports the creation of caches that are organized as hash tables. The hash tables are accessed via a hash function and collisions are resolved via the external chaining technique.

141. Each entry in the cache has an associated "lifetime." The "lifetime" of a cache entry is expressed as a duration (a number of seconds). The lifetime of a cache entry serves as a limit on the number of seconds a cache entry can remain in the cache without being accessed. Cache entries that have not been accessed in the last lifetime number of seconds are considered

to have expired and are eligible for removal from the cache. However, these expired records are removed from the cache in a lazy manner. Expired cache entries remain in the cache until they are searched for, at which time the expired record is identified and removed from the cache.

142. Figure 1 from the GCache report, reproduced below, illustrates the general organization of a cache created with the GCache software. A GCache cache is implemented as a hash table with a specified number of hash buckets. Each bucket contains a pointer to the head of a (doubly) linked list of nodes (called "cache entries"). Each cache entry is characterized by a "key" and a "result." The key is the data value that is used to identify a cache entry (the value that is hashed to insert, delete, or locate cache entries). The result is the data value that is returned as the result of searching for and locating a cache entry in the cache. Cache entries do not store keys and results but instead store pointers to a key and a result (*i.e.*, the key and result for a cache entry are stored in memory separately from the cache entry).

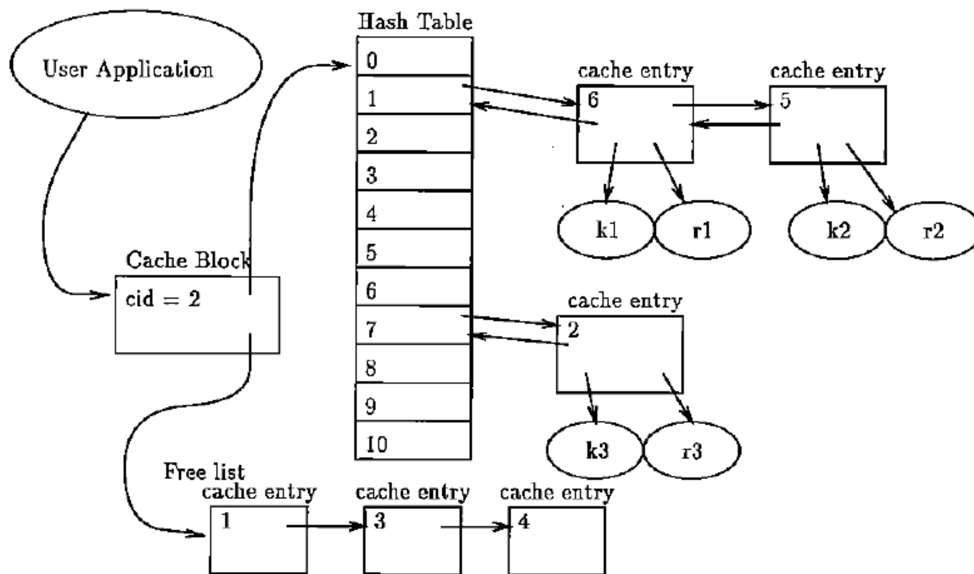


Figure 1: GCache Data Structures

143. The GCache software maintains an array of cache descriptors, called the "cache block." Each descriptor in the cache block points to a cache that has been created by an

application. Applications access GCache caches indirectly through the cache block. In the example illustrated in Figure 1 above from the GCache report, a user application is accessing a cache whose cache id is "2." Figure 1 also shows the hash table for the cache with cache id 2. The hash table consists of a number of buckets, each bucket pointing to a doubly linked list of cache entries, each cache entry containing pointers to a key and a result. Figure 1 also illustrates the fact that the GCache software maintains a maximum number of cache entries for each cache and entries not currently in use are stored in a free list accessed via the cache block entry for the cache.

E. Linux Operating System Kernel Version 2.0.1 *route.c* Software

144. I understand that Bedrock has accused at least versions 2.6.9, 2.6.11, 2.6.18, and 2.6.26 of the Linux operating system kernel of infringing the '120 Patent. Within these versions of the Linux kernel, I further understand that Bedrock has focused on the computer source code file *route.c*.

145. Prior to the release of versions 2.6.9, 2.6.11, 2.6.18, and 2.6.26 of the Linux kernel, version 2.0.1 of Linux was publicly available. Version 2.0.1 of the Linux operating system kernel also contained a version of the *route.c* computer source code that provides similar functions to those provided in to the newer versions of *route.c* that I understand Bedrock alleges infringe the asserted claims of the '120 Patent. Version 2.0.1 of the Linux kernel is publicly available on the Internet at the *ftp* site *ftp://ftp.kernel.org/pub/linux/kernel/v2.0/linux-2.0.1.tar.gz*. The version 2.0.1 distribution carries a file modification date of July 3, 1996. The computer source code file *route.c* analyzed here, found with the version 2.0.1 distribution at the path *linux-2.0.1/net/ipv4/route.c*, carries a file modification date of May 31, 1993. (The version of *route.c* analyzed here relies on the header file *linux-2.0.1/include/net/route.h* which carries a file modification date of July 3, 1996.) Combined, these dates are an indication that the files

downloaded from the *kernel.org ftp* site have not been modified since July 3, 1996. Further, Alexey Kuznetsov, one of the authors of the *route.c* code in Version 2.0.1, has confirmed that this version was publicly available on or about July 3, 1996 (see, Declaration of Alexey Kuznetsov, DEF00009284-85). I believe that the computer source code files *route.c* and *route.h* analyzed below were publicly available no later than July 3, 1996 and thus I understand qualify as prior art to the '120 Patent.[12]

146. The computer source code file *route.c* is part of the implementation of the TCP/IP protocol suite in the Linux operating system kernel. TCP/IP refers to a collection of communication protocols used by all computers on the Internet to communicate with one another. The computer source code in the file *route.c* concerns support for the routing of data packets (messages) between computers. Generally, the process of routing a message from one computer to another consists of consulting a table, called a routing table, to learn the information required to transmit the message towards its final destination. Because a computer may have multiple network interfaces (because a computer may be attached to multiple distinct networks at the same time), or because a computer operating system may support multiple types of protocols (because an operating system may support multiple different ways of communicating with other computers), the computer's operating system (Linux in this case) may have multiple separate routing tables. Thus, when the Linux operating system receives a message from an application for transmission to another computer, the Linux operating system must determine the appropriate

---

[12] I understand that versions 1.3.52 and 1.3.51 of the Linux kernel, both released in December 1995 (see, *http://www.kernel.org/pub/linux/kernel/v1.3,* Declaration of Alexey Kuznetsov, DEF00009285), have also been identified as prior art to the '120 Patent. The portions of version 2.0.1 of the Linux kernel that I analyze in this report also appear in version 1.3.52 and 1.3.51 of the Linux kernel and are substantively identical. Therefore, my discussion and analysis of version 2.0.1 of the Linux kernel applies equally to versions 1.3.52 and 1.3.51.

routing table to consult, and then access that table to learn the information required to transmit the message towards its intended destination.

147. The processing steps of locating and accessing the appropriate routing table are generically referred to as "routing" a message within the operating system. To speed up the process of routing a message, the Linux operating system maintains a cache of recently used data from recently accessed routing tables. This cache, called the "routing table cache," or simple the "route cache," is implemented as a hash table with collision resolution based on external chaining.

148. The implementation of the route cache in version 2.0.1 of the Linux kernel is substantially similar to the implementation of the cache appearing in later versions of the Linux kernel. For concreteness, the following describes the operation of a portion of the route cache in version 2.0.1 of the Linux kernel by comparison with the equivalent portions of the route cache implementation in version 2.6.18 of the Linux kernel. The same comparison can be made between version 2.0.1 of the Linux kernel and version 2.6.18 of the kernel.

149. In version 2.0.1 of the Linux kernel, the route cache is implemented as a hash table called ip_rt_hash_table ("the IP route hash table"). In version 2.6.18 of the Linux kernel the route cache is implemented as a hash table called ip_rt_hash_table. Each element of the hash table in version 2.0.1 of the Linux kernel (each bucket of the hash table) contains a pointer to record called an rtable (a "route table entry"). Each element of the hash table in version 2.6.18 of the Linux kernel similarly points to a record called an rtable. In both versions 2.0.1 and 2.6.18 of the Linux kernel the route table entries are linked to one another to form a linked list (an external chain for a hash table bucket). In version 2.0.1 of the Linux kernel the route hash table is accessed via a hash function called ip_rt_hash_code. In version 2.6.18 of the Linux kernel the

route hash table is accessed via a hash function called rt_hash_code. In both versions 2.0.1 and 2.6.18 of the Linux kernel this hash function is used to locate a bucket in the route hash table to find (or insert, or delete) a specific route table entry.

150. In version 2.0.1 of the Linux kernel, to add an entry to the route cache, the route hash table is used to locate an external chain (a linked list of route table entries) that is then accessed to perform the appropriate task. Each route table entry maintains a number of data values that describe the state of the entry. These data values include a reference count that indicates the number of references to the route table entry, and a form of timestamp that indicates when the route table entry was last accessed. When a route table entry has not been referenced (when its reference counter is zero), and when the entry has not been used for a specified period, the route table entry is considered to have expired (as Bedrock interprets expiration in its Infringement Contentions) and is eligible for deletion. When a route table entry is added to the route cache (when a route table entry is added to an external chain of the IP route hash table), the linked list that comprises an external chain of the IP route hash table is accessed to identify and remove expired route table entries.

F. The Dirks '214 Patent

151. U.S. Patent 6,119,214 to Dirks, titled "Method For Allocation of Address Space in a Virtual Memory System," was filed April 25, 1994, and issued September 12, 2000. Given these dates, I understand that the Dirks '214 Patent qualifies as prior art to the '120 Patent.

152. Generally, the Dirks '214 Patent discloses a method of managing the physical memory allocated to processes in a computer system. The patent is directed to the virtual memory system within the computer's operating system and hardware memory management unit. Virtual memory is the logical view of memory maintained by processes (programs executing on the computer). Virtual memory is distinguished from physical memory which is the

(physical) memory that is actually used to store executing programs and their data. Virtual memory is a means of accessing physical memory wherein executing programs reference memory by generating abstract virtual addresses that must be translated ("mapped") to actual physical addresses before memory can be accessed. In modern computer systems, the mapping from virtual addresses to physical addresses is performed by a hardware memory management unit that is controlled by the computer's operating system.

153. The memory management unit typically maintains a table, called a "page table," that maps virtual memory "pages" to physical memory pages. A page is the basic unit of memory allocation (some fairly large number of bytes). Conceptually, all of physical memory is partitioned into a number of equal-sized physical memory pages numbered from 0 to some large number. The largest page number depends on how much physical memory is present in the computer.

154. When a program commences execution, it must first be allocated a number of physical memory pages. These memory pages are used to hold the executable instructions of the program and any data the program requires for execution. The operating system of the computer will allocate a number of physical pages (a set of physical page numbers) for the program and assign them to the program. When the program executes, the program will then reference these physical pages as virtual pages (reference these physical pages by virtual page numbers). The page table in the memory management unit will keep track of the correspondence between the virtual page numbers and physical page numbers. Thus, when a process references a particular virtual page, the memory management unit will access the page table using the virtual page number (and possibly other data) to locate the appropriate physical page. Once located, the physical page corresponding to the requested virtual page can then be accessed.

155. The Dirks '214 Patent considers a particular type of page table commonly known as an "inverted page table" or a "reverse mapped page table."[13] As an example of an inverted page table, the Dirks '214 Patent references the Motorola PowerPC 601 processor (see, Dirks, col. 3, lines 64-66). An inverted or reverse mapped page table contains a number of entries that is proportional to the number of physical memory pages in the computer system (proportional to the amount of memory in the computer). Each page table entry contains one or more "page mappings" (or simply "mappings"). A mapping is a pairing of a virtual page number with its corresponding physical page number. A mapping tells the memory management unit where in physical memory (in which physical memory page) a process's virtual page can be found. In the PowerPC 601 processor each page table entry contained eight virtual page number to physical page number mappings. In the PowerPC architecture, the collection of eight mappings is referred to as a "page table entry group" (a "PTEG").

156. In an inverted page table, when a process references a particular virtual page, the memory management unit must search the page table to find the appropriate virtual page number to physical page number mapping (to find the number of the appropriate physical page to access). This searching process is performed via the use of hashing techniques. Specifically, the inverted page table is organized as a hash table (an array of hash locations). When the memory management unit receives a request for a virtual page from a process (from the program currently executing), the memory management unit will apply a hash function to data in the request to generate an index into the page table. Each entry in the page table (*e.g.*, each PTEG)

[13] The Dirks '214 Patent references the Motorola 601 processor for its background on memory management (see, Dirks, col. 3, lines 64-66). The Motorola 601 processor was an implementation of the Apple/IBM/Motorola PowerPC processor architecture. The description of inverted page tables here comes from both the Dirks '214 Patent itself as well as publicly

stores a number of virtual page number to physical page number mappings (see, Dirks, col. 9, lines 36-40). Using the terminology of the Nemes '495 Patent, an entry in the page table can thus be viewed as an external "chain" of records (see, *e.g.*, Nemes '495 Patent, col. 1, lines 60-63).

157. Once a hash value for a virtual page request is computed, the corresponding location in the page table (the corresponding PTEG) is accessed and searched to see if a mapping for the requested virtual page is present (see, Dirks, col. 9, lines 40-45). If such an entry is present, the physical page number corresponding to the requested virtual page number is returned and used to access physical memory. If an entry for the requested virtual page is not present in the page table then the requested virtual page is not present in physical memory (a physical page has not been allocated for this virtual page) and the operating system is invoked to resolve this situation.

158. The Dirks '214 Patent considers a virtual memory scheme that is similar as that found in the PowerPC 601 processor architecture. Specifically, each process, or sub-process ("thread"), is allocated a region of virtual memory called a "segment." A segment is identified by a number called a "virtual segment identifier" ("VSID"). A segment is composed of a number of equal sized virtual pages. When a process or thread accesses memory in some virtual page, the process or thread's VSID (its virtual segment number) and the virtual page number (called the "page index" in Dirks) within the segment are used to access the page table. The VSID and page index are used as inputs to a hash function to compute a location in the page table (a hash table) to search for the mapping to a physical page number.

159. When processes or threads terminate (are "deleted"), the VSID assigned the process or thread can be reallocated to some other process or thread. However, before any such

---

available information on the PowerPC architecture such as the reference "PowerPC Operating

reallocation can occur, all of the entries in the page table corresponding to deleted process' or thread's VSID must be deleted (because the process or thread has been deleted, all page mappings for the deleted process or thread are now invalid). Finding all of the page mappings corresponding to a VSID of a deleted process or thread is not easy because the virtual pages used by the deleted process or thread may be spread all over the page table (because page mappings are placed in the page table according to a hash function). Each location in the page table can be examined in turn to determine if it contains a page mapping for a VSID of a deleted process or thread, however, this will be a time consuming, and thus undesirable, operation.

160. To remedy this problem, the Dirks '214 Patent teaches a method of on-the-fly removal of page table entries that correspond to VSID's of deleted processes or threads. When a process or thread is created or deleted, or in response to some regularly occurring event, a number of entries in the page table are examined (see, Dirks, col. 10, lines 20-24). The Dirks '214 Patent refers to the process of examining a number of entries in the page table at the occurrence of an event as a "sweeping process." In the sweeping process, for each page table entry to be "swept," it is determined whether or not any page mappings in the page table entry correspond to VSIDs of deleted processes or threads. Any such page mappings found are removed from the page table because they are obsolete and no longer needed or desired (see, Dirks, col. 7, lines 2-7, col. 8, lines 44-46, col. 9, lines 11-14).

161. In more detail, the Dirks system maintains three lists of VSIDs: a "free list" of VSIDs that have not been allocated to any process or thread and thus are available for use, an "inactive list" of VSIDs of deleted processes or threads whose page mappings are still present in the page table, and a "recycle list" of VSIDs of deleted processes or threads whose page

---

Environment Architecture," Book III, Version 2.02, January 2005.

mappings are in the process of being deleted from the page table. During the sweeping process, a number of entries in the page table are examined. Any page mapping corresponding to a VSID on the inactive and/or recycle list that is found in a page table entry is deleted.

162. The Dirks system limits the number of entries that are examined on any given sweep to limit the amount of garbage collection that is done at any given moment (see, Dirks, col. 6, lines 9-15, col. 7, lines 14-46). One embodiment of the invention disclosed calculates the number of entries to examine based on "the total number of entries in the page table and the number of threads and applications that are allowed to be active at any given time" (see, Dirks, col. 7, lines 14-37). Dirks derives a formula for this calculation (see, Dirks, col. 8, lines 29-32):

$$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$

163. Dirks further discloses that "[a]ny other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process" (see, Dirks, col. 7, lines 38-40).

VII. ANALYSIS OF THE VALIDITY OF THE '120 PATENT

164. Here I assess the validity of the asserted claims of the '120 Patent. In performing this analysis I have used the claim constructions rendered by the Court in its January 10, 2011 Memorandum Opinion and Order concerning claim construction. For completeness, these claim constructions are listed in the table below. For terms and claim limitations where no construction has been provided, I analyze those elements using the plain and ordinary meaning of the terms as would have been understood by a person of ordinary skill in the art as of January 1997.

Claim Term	Court's Construction
"a linked list to store and provide access to records"	a list, capable of containing two or more records, in which each record contains a pointer to the next record or information

	indicating there is no next record
"automatically expiring"/ "expired"	becoming obsolete and therefore no longer needed or desired in the storage system because of some condition, event, or period of time / obsolete and therefore no longer needed or desired in the storage system because of some condition, event, or period of time
"removing . . . from the linked list"	adjusting the pointer in the linked list to bypass the previously identified expired records
"dynamically determining"	making a decision based on factors internal or external to the information storage and retrieval system
"maximum number"	no construction necessary, but not limited to a single number
"external chaining"	a technique for resolving hash collisions using a linked list(s)
"when the linked list is accessed"	both identification and removal of the automatically expired record(s) occurs during the same access of the linked list
Ordering of Method Steps – Claim 7	the "inserting, retrieving or deleting" step must follow, at least in part, "the step of removing"
Ordering of Method Steps	identifying must begin before removing can begin
"a record search means utilizing a search key to access the linked list"	Function: utilizing a search key to access the linked list Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions as described in Boxes 31-36 and Boxes 39-41 of FIG. 3 and in col. 5 line 53-col. 6 line 4 and col. 6 lines 14-20, and/or programmed with software

	instructions as described in the pseudo-code of Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14), and equivalents thereof
"the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed"	Function: identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions as described in Boxes 33-42 of FIG. 3 and in col. 5 line 53-col. 6 line 34, and/or programmed with software instructions as described in the pseudo-code of Search Table Procedure (cols. 11 and 12) or Alternate Version of Search Table Procedure (cols. 11, 12, 13, and 14), and equivalents thereof
"means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list"	Function: utilizing the record search means, accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions that provide the insert, retrieve, or delete record capability as described in the flowchart of FIG. 5 and col. 7 line 65-col. 8 line 32, FIG. 6 and col. 8 lines 33-34, or FIG. 7 and col. 8 lines 45-59, respectively, and/or programmed with software instructions that provide the insert, retrieve or delete record capability as described in the pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), or

	Delete Procedure (cols. 11 and 12), respectively, and equivalents thereof
"a hashing means to provide access . . ."	Function: to provide access to records stored in a memory of the system and using an external chaining technique to store records with same hash address at least some of the records automatically expiring Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions to provide a hash table having a pointer to the head of a linked list of externally chained records as described in col. 5 lines 16-26 and/or programmed with software instructions as described in pseudo-code of Definitions, definition number 4, and equivalents thereof
"means for dynamically determining maximum number"	Function: "dynamically determining maximum number of records for the record search means to remove in the accessed linked list of records" Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions to dynamically determine a maximum number of records to remove by choosing a search strategy of removing all expired records from a linked list or removing some but not all of the expired records as described in col. 6 line 56-col. 7 line 15 and/or programmed with software instructions to dynamically determine a maximum number of records to remove by choosing between the pseudo-code of the Search Table Procedure (cols. 11 and 12) or Alternative Version of Search Table Procedure (cols. 11, 12, 13,

	and 14), and equivalents thereof
"mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting from the system and, at the same time, removing at least some of the expired ones of the records in the accessed linked list of records"	Function: utilizing the record search means, inserting, retrieving, and deleting records from the system and, at the same time, removing at least some of the expired ones of the records in the accessed linked list of records Structure: CPU 10 and RAM 11 of FIG. 1 and col. 3 lines 52-56 and portions of the application software, user access software or operating system software, as described at col. 4 lines 22-48, programmed with software instructions that provide the insert, retrieve, and delete record capability as described in the flowchart of FIG. 5 and col. 7 line 65-col. 8 line 32, FIG. 6 and col. 8 lines 33-44, or FIG. 7 and col. 8 lines 45-59, respectively, and/or programmed with software instructions that provide the insert, retrieve and delete record capability as described in the pseudo-code of Insert Procedure (cols. 9 and 10), Retrieve Procedure (cols. 9, 10, 11, and 12), and Delete Procedure (cols. 11 and 12), respectively, and equivalents thereof

A. **<u>The NRL BSD Source Code Anticipates and/or Renders Obvious All Claims of the '120 Patent</u>**

165. It is my opinion that the NRL BSD code anticipates claims 3 and 4 of the '120 Patent and, in combination with itself or with various textbooks disclosing hashing with external chaining (including Knuth, Kruse, or Stubbs), renders obvious claims 1, 5, and 7. Further, the NRL BSD code in combination with itself or Knuth, Kruse, or Stubbs, and Dirks renders obvious claims 2, 4, 6, and 8.

1. The NRL BSD source code in combination with itself or Knuth, Kruse, or Stubbs, renders obvious claims 1 and 5 of the '120 Patent

166. The NRL BSD source code discloses an information storage and retrieval system wherein a variety of data items related to security associations are stored in, and retrieved from, data structures such as a linked list or a hash table (see, NRL BSD source code, *e.g.*, *key.c*, lines 648-1087, 1545-1560).

a. The NRL BSD code discloses the "linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring" limitation of claim 1 and the "hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" limitation of claim 5

167. The NRL BSD source code discloses a linked list to store and provide access to records stored in a memory of the system. The NRL BSD source code uses a linked list of records called a key_acquirelist (see, NRL BSD source code, *key.c*, line 129). The definition of a key_acquirelist record (expressed in the syntax of the C programming language) is:

```
struct key_acquirelist {
   u_int8 type;                 /* secassoc type to acquire */
   struct sockaddr_in6 target;  /* destination address of secassoc */
   u_int32 count;               /* number of acquire messages sent */
   u_long expiretime;           /* expiration time for acquire message */
   struct key_acquirelist *next;
};
```

(See NRL BSD source code, *key.h*, lines 188-194.)

168. The key_acquirelist structure includes a pointer to the next record in the linked list called "next" on line 193.

169. The NRL BSD code also discloses a hash table with external chaining called keytable (see NRL BSD source code, key.c, line 122). The keytable structure is defined as an array of linked lists, which is a hash table. The function key_gethashval() is used to hash the

search key to a hash table entry corresponding with the appropriate linked list (see NRL BSD source code, key.c, lines 425-50, 782, 881, 1014).

170. The definition of a node of the keytable is:

```
struct key_tblnode {
  int alloc_count;            /* number of sockets allocated to secassoc */
  int ref_count;              /* number of sockets referencing secassoc */
  struct socketlist *solist;    /* list of sockets allocated to secassoc */
  struct ipsec_assoc *secassoc; /* security association */
  struct key_tblnode *next;     /* next node */
};
```

(See NRL BSD source code, *key.h*, lines 162-168.)

171. The keytblnode structure includes a pointer to the next record in the linked list called "next" on line 167.

172. Knuth, Kruse, and Stubbs also disclose hashing with external chaining (see Knuth, pp. 513-518, Kruse pp. 206-08, Stubbs pp. 324-25).

173. It would have been obvious to one of ordinary skill in the art that the linked list used for the key_acquirelist could have been implemented as a hash table like the keytable hash table or as described in Knuth, Kruse, or Stubbs. One of ordinary skill in the art would have been motivated to replace the linked list with a hash table in a system where the number of records expected to be stored in the structure was large, since the hash table would result in more efficient searching.

174. Included in each key_acquirelist record is an expiration time "expiretime." Records in the key_acquirelist expire a set amount of time after they are added to the list. As shown in the following source code, by default, this duration is 15 seconds:

```
#define MAXACQUIRETIME 15;   /* Lifetime of acquire message */
```

(See NRL BSD source code, *key.c*, line 116.)

```
u_long maxacquiretime = MAXACQUIRETIME;
```

(See NRL BSD source code, *key.c*, line 132.)

```
DPRINTF(IDL_EVENT,("Updating acquire counter and expiration time\n"));
ap->count++;
ap->expiretime = time.tv_sec + maxacquiretime;
```
(See NRL BSD source code, *key.c*, function *key_acquire*, lines 1557-1559.)

175. Records in the key_acquirelist become obsolete and are therefore no longer needed in the storage system after their expiration time (a limited period of time):

```
} else if (ap->expiretime < time.tv_sec) {
  /*
   *  Since we're already looking at the list, we may as
   *  well delete expired entries as we scan through the list.
   *  This should really be done by a function like key_reaper()
   *  but until we code key_reaper(), this is a quick and dirty
   *  hack.
   */
  DPRINTF(IDL_MAJOR_EVENT,("found an expired entry...deleting it!\n"));
  prevap->next = ap->next;
  KFree(ap);
  ap = prevap;
}
```
(See NRL BSD source code, *key.c*, function *key_acquire*, lines 1445-1457.)

176. Thus, under the Court's claim constructions, the NRL BSD source code discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. The NRL BSD code in combination with itself or Knuth, Kruse, or Stubbs discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring.

b. The NRL BSD code discloses the "record search means utilizing a search key to access the linked list" limitation of claim 1 and the "record search means utilizing a search key to access a linked list of records having the same hash address" limitation of claim 5

177. Further, the NRL BSD source code discloses a record search means utilizing a search key to access the linked list. The NRL BSD source code discloses a key_acquirelist linked list of records that is accessed using a search key consisting of the combination of the destination address of the target security association and the type of the association.

178. The NRL BSD code discloses getting the head of the key_acquirelist and traversing the key_acquirelist in search of a record with a key that matches the search key:

```
for(ap = key_acquirelist->next; ap; ap = ap->next) {
  if (addrpart_equal(dst, (struct sockaddr *)&(ap->target)) &&
    (etype == ap->type)) {
    DPRINTF(IDL_MAJOR_EVENT,("acquire message previously sent!\n"));
    if (ap->expiretime < time.tv_sec) {
      DPRINTF(IDL_MAJOR_EVENT,("acquire message has expired!\n"));
      ap->count = 0;
      break;
     }
    if (ap->count < maxkeyacquire) {
    DPRINTF(IDL_MAJOR_EVENT,("max acquire messages not yet exceeded!\n"));
    break;
    }
    return(0);
  } else if (ap->expiretime < time.tv_sec) {
    /*
     *  Since we're already looking at the list, we may as
     *  well delete expired entries as we scan through the list.
     *  This should really be done by a function like key_reaper()
     *  but until we code key_reaper(), this is a quick and dirty
     *  hack.
     */
    DPRINTF(IDL_MAJOR_EVENT,("found an expired entry...deleting it!\n"));
    prevap->next = ap->next;
    KFree(ap);
    ap = prevap;
  }
  prevap = ap;
}
```

(See NRL BSD source code, *key.c*, function *key_acquire*, lines 1430-1459.)

179. When the NRL BSD code identifies a record with a key that matches the search key, the code exits the for loop that is traversing the linked list (see, NRL BSD source code, *key.c*, lines 1432-1442).

180. If a match is found during the traversal of the linked list, the variable ap will point to the matching record. If ap is not null, this indicates success of the search. If ap is null, this indicates that the search failed. On lines 1546 to 1556, a new record will be inserted into the key_acquirelist if the search failed.

181. The NRL BSD code also discloses the function *key_search()*, which searches the keytable hash table for a record with a key matching the search key (see, NRL BSD source code,

key.c, lines 648-685). The *key_search()* function is called by the function *key_add(), key_get(),* and *key_delete(),* each of which hashes the search key and passes the resulting value to the function *key_search()* as the parameter indx (see, NRL BSD source code, *key.c*, lines 782-86, 881-83, 1014-16). The *key_search()* function then gets the head of the target linked list corresponding with that hash value in the key_tblnode and traverses the linked list in search of a record with a key matching the record search key (see, NRL BSD source code, *key.c*, lines 676-85). All of the records in that linked list will have the same hash address. When the *key_search()* function finds a key match, it saves a pointer to the list element in the variable keynode, stops traversing the linked list, and returns a pointer to the matching record to the calling function (indicating success of the search). If no match is found, *key_search()* returns a null value, which indicates failure of the search. (see NRL BSD source code, key.c, lines 677-81, 683-84).

182. Though the *key_acquire()* code does not hash the search key before getting the head of the target list, it would have been obvious to one of ordinary skill in the art that a hash table could have been used instead of a linked list to store and provide access to records.

183. One of skill in the art would have been motivated to use a hash table like the one used in the NRL BSD code for keytable or as described in Knuth, Kruse, or Stubbs instead of the linked list used for the key_acquirelist where a large number of records were expected to be stored because a hash table provides for more efficient searching of records. In such a case, the linked list accessed by the *key_acquire()* function would be a linked list of records having the same hash address.

184. The NRL BSD source code in combination with itself or Knuth, Kruse, or Stubbs discloses using a search key to access records stored in the list that is equivalent to the structure identified by the Court in its Claim Construction Order (see, NRL BSD source code, *key.c*, lines

1430-1563), including Figure 3, boxes 32-36 and 39-41 and the Alternate Version of the Search Table Procedure in the pseudo-code of the '120 Patent.

c. The NRL BSD code discloses the "record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed" limitation of claims 1 and 5

185. Further, the NRL BSD source code discloses a means for identifying and removing at least some of the expired records from the key_acquirelist linked list when the linked list is accessed. The NRL BSD source code discloses a key_acquirelist linked list of records that is accessed using a search key consisting of the combination of the destination address of the target security association and the type of the association. That list is accessed to search for a record that is being inserted. During that access to the key_acquirelist linked list, expired records are identified and removed (see, NRL BSD source code, *key.h*, lines 188-194, *key.c*, lines 1431-1459). This is made clear by the NRL developers comment in the source code to access the key_acquirelist linked list:

```
/*
 *  Since we're already looking at the list, we may as
 *  well delete expired entries as we scan through the list.
 *  This should really be done by a function like key_reaper()
 *  but until we code key_reaper(), this is a quick and dirty
 *  hack.
 */
```

(See, NRL BSD source code, *key.c*, lines 1446-1452.)

186. The NRL BSD source code discloses a structure for identifying and removing at least some of the expired records from the linked list that is equivalent to the structure identified by the Court's claim construction order (see, NRL BSD source code, *key.c*, lines 1431-1459).

d. The NRL BSD code discloses the "means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" limitation of claim 1 and the "means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records" limitation of claim 5

187. Further, the NRL BSD source code discloses using the record search means to access the key_acquirelist linked list to insert a record and at the same time, remove at least some of the expired records from the key_acquirelist linked list (see, NRL BSD source code, *key.h*, lines 188-194, *key.c*, lines 1431-1459). The *key_acquire()* function uses the code at lines 1431-59 to search for a record matching the search key and, at the same time, remove expired records from the linked list as described above. If a record is not found during that search, a new record is inserted into the linked list on lines 1546-60.

188. Line 1548 allocates memory for the new list element by calling the function *K_Malloc()*. Line 1552 copies the record into the new list element and lines 1554-55 adjust the pointers in the key_acquirelist linked list to insert the new record into that linked list (see, NRL BSD source code, *key.c*, lines 1548-1555).

189. The NRL BSD code further discloses that the *key_acquire()* function is called by the function *getassocbysocket()* (see NRL BSD source code, key.c, lines 1779, 1835).

190. The NRL BSD code also discloses the functions *key_add(), key_get(),* and *key_delete()*, all of which utilize the function *key_search()* to access the linked lists in the key_table hash table to search for the records being inserted, retrieved and deleted, respectively.

191. The function *key_add()* adds a record to the keytable hash table. The *key_add()* function calculates a hash value based on the search key and calls *key_search()* to search for the record being added, passing in the hash value as one of the parameters (see, NRL BSD source

code, *key.c*, lines 782-786). If *key_search()* finds a record, *key_add()* returns a value indicating that the record already exists in the table. If *key_search()* does not find a record, *key_add()* allocates memory for a new record in lines 802-808 and then adds a new record by calling the function *key_addnode()* on line 812. The function *key_add()* copies the record to be inserted into the new list element and inserts that element into the keytable hash table by adjusting pointers. (See, NRL BSD source code, *key.c*, lines 720-23).

192. The function *key_get()* retrieves a record stored in the keytable hash table. The function *key_get()* calculates a hash value based on the search key of the record being searched for and utilizes *key_search()* to search for the record being retrieved (see, NRL BSD source code, *key.c*, lines 881-883). If *key_search()* finds the record, the security association stored in the record is copied into the structure secassoc and the function returns 0 indicating success (see, NRL BSD source code, *key.c,* lines 884-88). If no record is found, the function returns -1, indicating failure (see, NRL BSD source code, *key.c,* lines 888-89).

193. The function *key_delete()* deletes a record from the keytable hash table. The function *key_delete()* calculates a hash value based on the search key of the record being searched for and utilizes *key_search()* to search for the record being deleted (see, NRL BSD source code, *key.c,* lines 1014-16). If *key_search()* finds the record being deleted, the record is removed from the keytable hash table by adjusting pointers to bypass the record (see, NRL BSD source code, *key.c,* lines 1020, 1031). The function then returns 0, indicating success (see, NRL BSD source code, *key.c,* line 1084). Otherwise, the function returns -1, indicating failure (see, NRL BSD source code, *key.c,* line 1086).

194. To the extent the key_acquire function does not disclose the structure for this limitation, the *key_acquire()* function in combination with the *key_search(), key_add(),*

key_get(), and *key_delete()* functions does disclose this structure. As previously discussed, it would have been obvious to one of skill in the art that the linked list in the key_acquirelist could have been replaced by a hash table as implemented in keytable or as described by Knuth, Kruse, or Stubbs, and that insertions, retrievals, and deletions from that hash table could have been implemented in the same manner as for the keytable hash table.

195. For these reasons, the NRL BSD source code in combination with itself or Knuth, Kruse, or Stubbs discloses a structure for utilizing the record search means to access the linked list or insert, retrieve, and delete records from the system and, at the same time, remove at least some of the expired records from the linked list that is equivalent to the structure identified by the Court's claim construction order.

2. The NRL BSD source code renders obvious claims 2 and 6 of the '120 Patent

196. The NRL BSD source code discloses a means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.

197. In Bedrock's Infringement Contentions, Bedrock has taken the position that the determination of whether or not to delete one expired record falls within the meaning of this claim. While I do not agree with this interpretation, if the Court were to agree with this interpretation, the NRL BSD code anticipates this claim.

198. The NRL BSD code contains a conditional statement that determines whether or not to delete an expired record based on a comparison of the record's expiration time with the current time (see, NRL BSD source code, *key.c,* lines 1445-1457). This determination of whether or not to delete an expired record falls within Bedrock's interpretation of this claim as I understand it.

199. Additionally, it is a fundamental concept in computer programming to make decisions in a program based on factors internal or external to the system being implemented. Thus, arguably, virtually any determination that is made in a computer system is a dynamic determination as that term has been construed. If processing load were a concern in the NRL IPv6 system, it would have been obvious to one of ordinary skill in the art to introduce a dynamic determination of the number of expired records to remove from the key_acquirelist to limit the amount of clean up work performed during the traversal of the list. That is, since deletion of expired records from the key_acquirelist was an afterthought ("since we're already looking at the list, we may as well delete expired entries as we scan through the list" [see, NRL BSD source code, lines 1430-1459]), it would have been obvious to limit the amount of time spent deleting expired records to, for example, limit the time spent in any one instance of the *key_acquire()* function.

200. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Dr. Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one" (see, '120 Patent, col. 7, lines 10-15).

201. Furthermore, the NRL BSD code in combination with the Dirks '214 Patent renders these claims obvious. The disclosures of the Dirks '214 Patent are described in Sections VI.F and VII.E. The Dirks '214 Patent teaches a simple formula for bounding the number hash

table entries considered when deleting expired entries as a fraction of the total number of entries in the hash table and the number of active processing entities. A person or ordinary skill in the art would recognize that this simple bounding ratio could be applied to the bounding the number of removals in the key_acquirelist processing by, for example, computing the ratio of the number of entries in the key_acquirelist to the number of active processes.

3. The NRL BSD source code anticipates claim 3 and renders obvious claim 7 of the '120 Patent

202. The NRL BSD source code discloses a method for storing and retrieving information records using a linked list and/or a hash table to store and provide access to records. (see, NRL BSD source code, *e.g.*, *key.c*, lines 648-1087, 1545-1560). The NRL BSD code also discloses that at least some of the records in stored in the linked list are automatically expiring, as discussed above in Section VII.A.1.a. Though the NRL BSD code does not disclose automatically expiring records in a hash table, one of ordinary skill in the art would have found it obvious to store the automatically expiring records in a hash table instead of a linked list for the reasons set forth above.

a. The NRL BSD code discloses accessing the linked list of records and accessing a linked list of records having the same hash address

203. The NRL BSD source code discloses accessing a linked list of records in the *key_acquire()* function, as discussed above in Section VII.A.1.b. It also discloses accessing a linked list of records having the same hash address in the *key_search()* function, as discussed above in Section VII.A.1.b.

204. As previously discussed above, it would have been obvious to one of ordinary skill in the art that a hash table like the keytable hash table or as described in Knuth could be used instead of a linked list in the *key_acquire()* function.

b. The NRL BSD code discloses identifying at least some of the automatically expired ones of the records

205. The NRL BSD code discloses identifying at least some of the automatically expired ones of the records when the linked list is accessed.

206. As discussed above in Section VII.A.1.c, the *key_acquire()* function identifies automatically expired records while the linked list is being traversed in search of the record to be inserted.

c. The NRL BSD code discloses removing at least some of the automatically expired ones of the records from the linked list when the linked list is accessed

207. The NRL BSD code discloses removing at least some of the automatically expired ones of the records during the same access of the linked list as the one in which the automatically expired records are identified.

208. As discussed above in Section VII.A.1.c, the *key_acquire()* function identifies and removes the automatically expired records from the linked list while the linked list is being traversed in search of the record to be inserted.

d. The NRL BSD code discloses inserting, retrieving or deleting one of the records from the system following the step of removing

209. The NRL BSD code discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

210. As discussed above in Section VII.A.1.d, if the record searched for during the traversal of the linked list in the *key_acquire()* function is not found, the record will be inserted. This insertion occurs after the expired records have been identified and removed (see, NRL BSD source code, *key.c*, lines 1543-1560).

211. Further, as discussed above, it would have been obvious to one skilled in the art that the linked list in the key_acquirelist could have been replaced by a hash table, and that insertions, retrievals, and deletions from that hash table could have been implemented in the same manner as for the keytable hash table or as described in Knuth, Kruse, or Stubbs. The resulting code would insert, retrieve, or delete a record after identifying and removing automatically expired records in the *key_search()* function.

4. The NRL BSD source code anticipates and/or renders obvious claim 4 and renders obvious claim 8 of the '120 Patent

212. The NRL BSD source code discloses the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.

213. As discussed above in Section VII.A.2, under Bedrock's interpretation of this claim, the NRL BSD code dynamically determines a maximum number of records to remove.

214. As discussed above in section VII.A.2, it would have been obvious to one of ordinary skill in the art to combine the NRL BSD code with the dynamic determination of a maximum number that occurs in Dirks.

B. <u>The GCache Source Code Anticipates and/or Renders Obvious All Claims of the '120 Patent</u>

215. It is my opinion that the GCache code anticipates all claims of the '120 Patent and, in combination with Dirks, renders obvious claims 2, 4, 6, and 8.

1. The GCache source code anticipates claims 1 and 5 of the '120 Patent

216. The GCache source code discloses an information storage and retrieval system wherein data values to be cached are stored in, and retrieved from, caches organized as hash tables with collision resolution based on external linked lists (see, GCache source code, *e.g.*, *gcache.c*, structures *cacheblk* [line 22], *cacheentry* [line 53], and functions *cainsert* [line 246], *caremove* [line 355], and *calookup* [line 312]).

217. Each cache entry has an associated "lifetime." When a cache is created, the maximum lifetime for each cache entry is specified (see, GCache source code, functions *cacreate* [line 135], see also, GCache report, p. 2 "Timeouts," p. 3, "User Interface"). When a cache entry is accessed, the time since the last access of the entry (or the time since the cache entry was added to the cache if the cache entry has never been accessed) is compared against the current time to determine if the time since the last access is greater than the lifetime of the cache entry. If so, the cache entry is identified as expired and the cache entry is removed from the cache (see, GCache source code, *e.g.*, functions *calookup* [line 312], *cagetindex* [line 643], *caisold* [line 622], and GCache report, pp. 9-10, "Timeouts").

218. Thus, the GCache source code discloses a method of on-the-fly removal of automatically expiring records from an information storage and retrieval system. As the following analysis shows, the GCache system for on-the-fly removal of automatically expiring records from a linked list/hash table is the same as that disclosed in the '120 Patent.

a. The GCache source code discloses the "linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring" limitation of claim 1 and the "hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" limitation of claim 5.

219. The GCache source code discloses a linked list and a hashing means to store and provide access to records stored in a memory of the system. The GCache source code discloses a hash table with external chaining, *cb_hash*, that is part of the structure *cacheblk* (see, GCache source code, structure *cacheblk* [line 31], see also, GCache report, p. 3, "Data Structures"). The GCache source code also discloses a linked list of cache entries (see, GCache source code, structure *cacheentry* [lines 53-64], see also, GCache report, p. 3, "Data Structures"). As

described above, cache entries store and provide access to records such as keys and results that are stored in a memory. Each cache entry contains two pointers, *ce_prev* and *ce_next,* which point to the previous and next nodes in the linked lists (see, GCache source code, structure *cacheentry* [lines 62-63]).

220. The GCache source code discloses that at least some of the records stored in the linked list of GCache entries automatically expire. Cache entries automatically become obsolete and therefore no longer desired in the storage system because of the condition that the time since the last access of the cache entry exceeds the lifetime of the cache entry (see, GCache source code, functions *cacreate* [line 135], *calookup* [line 312], *cagetindex* [line 666-70], *caisold* [line 617-634], see also, GCache report, p. 2 "Timeouts," p. 3, "User Interface," pp. 9-10, "Timeouts").

b. The GCache source code discloses the "record search means utilizing a search key to access the linked list" limitation of claim 1 and the "record search means utilizing a search key to access a linked list of records having the same hash address" limitation of claim 5

221. The GCache source code discloses a record search means utilizing a search key to access the linked list of records having the same hash address. The GCache source code discloses a search key *pkey* that is used as an input to a hash function, the result of which is passed into the function *cagetindex* to access the linked list to locate a record corresponding to the search key (see, GCache source code, functions, *e.g.*, *calookup* [line 332-333], *cahash* [line 516-27], *cagetindex* [line 643-78], see also, GCache report, p. 3, "Data Structures"). For example, *cainsert* calculates a hash value at line 272 by calling the function *cahash* with *pkey* as one of its parameters. The function *cahash* then outputs a hash value that is passed into the function *cagetindex* on line 275. The function *cagetindex* then accesses the linked list in search of a record with a search key that matches the target record (see, GCache source code, functions,

cagetindex [lines 643-78]). If one is found, and that record is not expired, *cagetindex* returns a pointer to that record (see, GCache source code, functions, *cagetindex* [line 671]). Otherwise, a null pointer is returned indicating that the search failed (see, GCache source code, functions, *cagetindex* [line 669, 677]).

c. The GCache source code discloses the "record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed" limitation of claims 1 and 5

222. The GCache source code discloses that the record search means includes a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. The GCache source code discloses that during the access to the linked list of cache entries to locate a cache entry in *cagetindex*, the GCache source code both identifies and removes the cache entry if the entry has expired (see, GCache source code, functions *calookup* [line 312], *cagetindex* [line 643], *caisold* [line 622], *caunlink* [line 742], see also, GCache report, p. 2 "Timeouts," p. 3, "User Interface," pp. 9-10, "Timeouts"). More specifically, when *cagetindex* finds a record that matches the target record, it will call the function *caisold* to determine whether the record is expired (see, GCache source code, functions, *cagetindex* [line 661-672], *caisold* [622-634]). If the record is expired, the function *caunlink* is used to adjust the pointers in the linked list to bypass the expired record (see, GCache source code, functions, *cagetindex* [line 668], *caunlink* [lines 757-59]).

d. **The GCache source code discloses the “means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list” limitation of claim 1 and the “means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records” limitation of claim 5**

223. The GCache source code discloses a means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The GCache source code discloses that the *cagetindex* function is called by the *cainsert*, *calookup*, and *caremove* functions to insert, retrieve, and delete a record and, at the same time, remove expired records from the hash table (see, GCache source code, functions *cainsert* [line 241-304], *caremove* [lines 350-76], and *calookup* [line 307-47]).

224. The *cainsert* function calculates a hash function using *cahash*, then passes that hash value along with the search key for the record to be inserted into the *cagetindex* function to search for whether that record already exists in the hash table (see, GCache source code, functions *cainsert* [line 272-75]). If the record is found in the table, no new record is inserted, but rather the existing record is used to store the newly inserted record. If the record is not found, a new record is inserted into the head of the linked list. Either way, the contents of the record to be inserted are copied into the existing/newly inserted record (see, GCache source code, functions *cainsert* [line 275-89]).

225. The *calookup* function similarly calculates a hash function using *cahash* and passes that hash value and the search key for the record to be retrieved into the *cagetindex* function to search for the target record (see, GCache source code, functions *calookup* [lines 332-33]). If the record is found in the table, the information in the record is copied and the function

returns success (see, GCache source code, functions *calookup* [lines 333-43]). Otherwise, the function returns failure (see, GCache source code, functions *calookup* [line 346]).

226. The *caremove* function also calculates a hash value and calls *cagetindex* using that value and the search key for the record to be deleted in order to search for the target record (see, GCache source code, functions *caremove* [lines 369-70]). If the search is successful, *cagetindex* will return a pointer to the record to be removed, which is then passed into *caunlink* to adjust the pointers to bypass the record, thus removing it from the linked list (see, GCache source code, functions *caremove* [lines 370-72], *caunlink* [lines 757-59]).

2. The GCache source code anticipates and/or renders obvious claims 2 and 6 of the '120 Patent

227. The GCache source code discloses a means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.

228. In Bedrock's Infringement Contentions, Bedrock has taken the position that the determination of whether or not to delete one expired record falls within the meaning of this claim. While I do not agree with this interpretation, if the Court were to agree with this interpretation, the Linux v2.0.1 code anticipates this claim.

229. The GCache code contains a conditional statement that determines whether or not to delete an expired record based on a comparison of the record's expiration time with the current time (see, GCache source code, function *cagetindex* [lines 666-68]). This determination of whether or not to delete an expired record falls within Bedrock's interpretation of this claim as I understand it.

230. Additionally, it is a fundamental concept in computer programming to make decisions in a program based on factors internal or external to the system being implemented. Thus, arguably, virtually any determination that is made in a computer system is a dynamic

determination as that term has been construed. If processing load were a concern in the GCache system, it would have been obvious to one of ordinary skill in the art to introduce a dynamic determination of the number of expired records to remove from cb_hash to limit the amount of clean up work performed during the traversal of the list. It would have been obvious to limit the amount of time spent deleting expired records to, for example, limit the time spent in any one instance of the *cagetindex* function.

231. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Dr. Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one" (see, '120 Patent, col. 7, lines 10-15).

232. Furthermore, the NRL BSD code in combination with the Dirks '214 Patent renders these claims obvious. The disclosures of the Dirks '214 Patent are described in Sections VI.F and VII.E. The Dirks '214 Patent teaches a simple formula for bounding the number hash table entries considered when deleting expired entries as a fraction of the total number of entries in the hash table and the number of active processing entities. A person or ordinary skill in the art would recognize that this simple bounding ratio could be applied to the bounding the number of removals in the *cagetindex* processing by, for example, computing the ratio of the number of entries in the *cagetindex* to the number of active processes.

3. The GCache source code anticipates claims 3 and 7 of the '120 Patent

233. As previously discussed, the GCache source code discloses a method for storing and retrieving information records using a linked list and/or a hash table with external chaining to store and provide access to records, some of which automatically expire based on whether the record's lifetime has passed.

a. The GCache source code discloses accessing the linked list of records and accessing a linked list of records having the same hash address

234. The GCache source code discloses accessing a linked list of records having the same hash address in the *cagetindex* function, as discussed above in Section VII.B.1.b.

b. The GCache source code discloses identifying at least some of the automatically expired ones of the records

235. The GCache code discloses identifying at least some of the automatically expired ones of the records when the linked list is accessed.

236. As discussed above in Section VVI.B.1.c, the *cagetindex* function identifies an automatically expired record while the linked list is being accessed in search of a target record.

c. The GCache source code discloses removing at least some of the automatically expired ones of the records from the linked list when the linked list is accessed

237. The GCache code discloses removing at least some of the automatically expired ones of the records during the same access of the linked list as the one in which the automatically expired records are identified.

238. As discussed above in Section VII.B.1.c, the *cagetindex* function identifies and removes the automatically expired record that was previously identified from the linked list while the linked list is being accessed in search of a target record.

d. The GCache source code discloses inserting, retrieving or deleting one of the records from the system following the step of removing

239. The GCache code discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

240. As discussed above in Section VII.B.1.d, the *cagetindex* function is utilized by the functions *cainsert*, *calookup*, and *caremove* to search for a record to be inserted, retrieved, or deleted from the hash table and remove expired records. The cagetindex function returns a pointer to that record, which is then used by *cainsert*, *calookup*, or *caremove* to insert, retrieve or delete the record returned.

4. The GCache source code anticipates and/or renders obvious claims 4 and 8 of the ‘120 Patent

241. The GCache source code discloses the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.

242. As discussed above in Section VII.B.2, under Bedrock’s interpretation of this claim, the GCache code dynamically determines a maximum number of records to remove.

243. As discussed above in section VII.B.2, it would have been obvious to one of ordinary skill in the art to combine the GCache code with the dynamic determination of a maximum number that occurs in Dirks.

C. The Linux Version 2.0.1 Source Code Anticipates and/or Renders Obvious All Claims of the ‘120 Patent

244. It is my opinion that the Linux v2.0.1 source code anticipates all claims of the ‘120 Patent and, in combination with Dirks, renders obvious claims 2, 4, 6, and 8.

1. The Linux version 2.0.1 source code anticipates claims 1 and 5 of the '120 Patent

245. The Linux v2.0.1 source code discloses an information storage and retrieval system wherein route table entries are stored in, and retrieved from, a route cache that is organized as a hash table with collision resolution based on external linked lists (see, Linux v2.0.1 source code, file *route.h*, structure *rtable* [line 65], file *route.c*, *e.g.*, structure *ip_rt_hash_table* [line 151], function *rt_cache_add* [line 1299]).

246. Each route table entry has an associated "lifetime," after which the entry is considered to have expired (as Bedrock interprets expiration in its Infringement Contentions) (see, Linux v2.0.1 source code, file *route.h*, structure *rtable* [line 65], file *route.c*, *e.g.*, function *ip_rt_check_expire* [line 968], *rt_cache_add* [line 1299]). A record expires when its reference count drops to zero, indicating that the record is no longer being used, and its lifetime passes. When a route table entry is inserted into the linked list of the hash table (added to an external chain of the IP route hash table), the *rt_cache_add* function traverses the linked list to search for any entries matching the entry just inserted and deletes those duplicate entries if they are found. During this traversal in search of the inserted entry, the *rt_cache_add* function also identifies and removes expired records linked list that it is traversing (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add*, lines 1361-1383).

247. Thus, the Linux version 2.0.1 source code discloses a method of on-the-fly removal of automatically expiring records from an information storage and retrieval system. As the following analysis shows, the Linux version 2.0.1 system for on-the-fly removal of automatically expiring records from a linked list/hash table is the same as that disclosed in the '120 Patent.

a. The Linux version 2.0.1 source code discloses the "linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring" limitation of claim 1 and the "hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring" limitation of claim 5

248. The Linux v2.0.1 source code discloses a linked list to store and provide access to records stored in a memory of the system. The Linux v2.0.1 source code discloses a hash table *ip_rt_hash_table*. The *ip_rt_hash_table* structure is defined at line 151 of the *route.c* code as a hash table with external chaining. This hash table consists of *rtable* structures, which are defined at lines 65-81 of the *route.h* code as nodes of a linked list and include a pointer to the next record in the linked list of route table entries at line 67.

249. Under Bedrock's interpretation of expiration, the Linux v2.0.1 source code further discloses that at least some of the records stored in the hash table of route table entries automatically expire. Routing table entries automatically become obsolete and therefore no longer needed or desired in the storage system because of the condition that the time since the last use of the route table entry exceeds the timeout value for the route table entry and the route table entry is not currently being referenced (see, Linux v2.0.1 source code, file *route.h*, structure *rtable* [line 65], file *route.c*, function *ip_rt_check_expire* [line 968], *rt_cache_add* [lines 1369-1380]).

b. The Linux version 2.0.1 source code discloses the "record search means utilizing a search key to access the linked list" limitation of claim 1 and the "record search means utilizing a search key to access a linked list of records having the same hash address" limitation of claim 5

250. The Linux v2.0.1 source code discloses a record search means utilizing a search key to access a linked list of records having the same hash address. The Linux v2.0.1 source code

discloses a search key, a form of destination address, that is used as an input to a hash function, the result of which is further used to access the linked list. For example, the function *rt_redirect_1* calculates a hash value at line 1039 based on the destination address of the record being inserted, and then calls the *rt_cache_add* function to insert that record into the route cache (see, Linux v2.0.1 source code, file *route.c*, function *rt_redirect_1* [lines 1039, 1061]). To calculate the hash value, *rt_redirect_1* calls the function *ip_rt_hash_code*, which is defined in *route.h* at lines 116-120. The resulting value is passed into the *rt_cache_add* function and is used to access the appropriate linked list in the hash table *ip_rt_hash_table* (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [line 1345]). That linked list is then traversed in search of a record matching the search key (the destination address) and, if found, deletes it from the linked list (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [line 1365-83]).

251. The Linux 2.0.1 source code thus discloses using a search key to access records stored in the list that is equivalent to the structure identified by the Court in its Claim Construction Order.

c. The Linux version 2.0.1 source code discloses the "record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed" limitation of claims 1 and 5

252. The Linux v2.0.1 source code discloses that the record search means includes a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. The Linux v2.0.1 source code discloses that during the access to the linked list of route table entries to search for a duplicate entry to delete, the Linux v2.0.1 source code both identifies and removes route table entries that have expired (see, Linux v2.0.1 source code, file *route.h*, structure *rtable* [line 65], file *route.c*, function *rt_cache_add* [lines 1365-1383]).

253. In particular, the Linux v2.0.1 source code identifies an expired record (according to Bedrock's interpretation) on lines 1369-1370 based on whether the reference count is zero and the lifetime of the record has passed:

```
if ((!rth->rt_refcnt && rth->rt_lastuse + RT_CACHE_TIMEOUT < now)
     || rth->rt_dst == daddr)
```

254. After an expired record is identified, it is then removed from the linked list by adjusting the pointers in the linked list to bypass the record (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [line 1372]).

d. The Linux version 2.0.1 source code discloses the "means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" limitation of claim 1 and the "means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records" limitation of claim 5

255. The Linux v2.0.1 source code discloses a means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The Linux v2.0.1 source code discloses that the *rt_cache_add* function is called by the *ip_rt_slow_route* and *rt_redirect_1* functions, which use *rt_cache_add* to insert a record into the hash table, retrieve a record from the table to print, and deletes any duplicate records from the hash table, and at the same time, remove expired records from the hash table (see, Linux v2.0.1 source code, file *route.c*, function *rt_redirect_1* [line 1061], *ip_rt_slow_route* [line 1490], *rt_cache_add* [line 1299-1385]).

256. The function *rt_cache_add* inserts, retrieves, and deletes records from the hash table at the same time that it removes at least some of the expired ones of the records from the accessed linked list of records. First, *rt_cache_add* inserts a record into the linked list at lines 1356-57. Though this happens before the removal of expired records from the accessed linked

list of records that occurs on lines 1365-83, this structure is functionally equivalent to, and insubstantially different from, performing the insertion after the linked list of records.

257. Moreover, to the extent it is not equivalent, performing the insertion after the traversal of the linked list of records would have been an obvious variation. Whether the insertion is performed before or after the removal of expired records is an obvious and trivial design choice.

258. After the insertion of the record into the linked list, the *rt_cache_add* function traverses the remainder of the linked list in search of the record just added so that it can delete any duplicate records to the record just added. During that traversal in search of the duplicate record to delete, any expired records are identified and removed (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [line 1361-1383]).

259. When a duplicate record or an expired record are found, the *rt_cache_add* function then retrieves the record that was just removed so that it can print it using the printk function at line 1376 (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [line 1376]).

260. Further, Bedrock's apparently construes this claim element to include the deletion of a record in a separate traversal of the linked list that occurs prior to the removal of expired records (see, *e.g.*, Bedrock's Jan. 12, 2011 P.R. 3-6 Infringement Contentions – Google 2.6.18 at 12-13). While I do not agree with Bedrock's position, if the Court adopts this position, then both the insertion described above and deletions that occur during the call to function *rt_garbage_collect* at line 1342 disclose this claim element (see, Linux v2.0.1 source code, file *route.c*, function *rt_cache_add* [lines 1342, 1356], *rt_garbage_collect* [lines 1289-1297], *rt_garbage_collect_1* [lines 1107-1137]). The function *rt_garbage_collect* calls

rt_garbage_collect_1, which traverses the linked list to remove expired records (see, Linux v2.0.1 source code, file *route.c*, *rt_garbage_collect* [lines 1289-1297], *rt_garbage_collect_1* [lines 1107-1137]).

2. The Linux v2.0.1 source code anticipates and/or renders obvious claims 2 and 6 of the '120 Patent

261. The Linux v2.0.1 source code discloses a means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.

262. In Bedrock's Infringement Contentions, Bedrock has taken the position that the determination of whether or not to delete one expired record falls within the meaning of this claim. While I do not agree with this interpretation, if the Court were to agree with this interpretation, the Linux v2.0.1 code anticipates this claim.

263. The Linux v2.0.1 code contains a conditional statement that determines whether or not to delete an expired record based on a comparison of the record's expiration time with the current time (see, Linux v2.0.1 source code, *route.c,* lines 1369-80). This determination of whether or not to delete an expired record falls within Bedrock's interpretation of this claim as I understand it.

264. Additionally, it is a fundamental concept in computer programming to make decisions in a program based on factors internal or external to the system being implemented. Thus, arguably, virtually any determination that is made in a computer system is a dynamic determination as that term has been construed. If processing load were a concern in the Linux v2.0.1 system, it would have been obvious to one of ordinary skill in the art to introduce a dynamic determination of the number of expired records to remove from the ip_rt_hash_table to limit the amount of clean up work performed during the traversal of the list. It would have been

obvious to limit the amount of time spent deleting expired records to, for example, limit the time spent in any one instance of the *rt_cache_add* function.

265. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Dr. Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one" (see, '120 Patent, col. 7, lines 10-15).

266. Furthermore, the Linux v2.0.1 code in combination with the Dirks '214 Patent renders these claims obvious. The disclosures of the Dirks '214 Patent are described in Sections VI.F and VII.E. The Dirks '214 Patent teaches a simple formula for bounding the number hash table entries considered when deleting expired entries as a fraction of the total number of entries in the hash table and the number of active processing entities. A person or ordinary skill in the art would recognize that this simple bounding ratio could be applied to the bounding the number of removals in the ip_rt_hash_table processing by, for example, computing the ratio of the number of entries in the ip_rt_hash_table to the number of active processes.

3. The Linux version 2.0.1 source code anticipates claims 3 and 7 of the '120 Patent

267. As previously discussed, the Linux v2.0.1 source code discloses a method for storing and retrieving information records using a linked list and/or a hash table with external chaining to store and provide access to records, some of which automatically expire based on whether the record has a reference count of zero and whether the record's lifetime has passed.

a. The Linux version 2.0.1 source code discloses accessing the linked list of records and accessing a linked list of records having the same hash address

268. The Linux v2.0.1 source code discloses accessing a linked list of records having the same hash address in the *rt_cache_add* function, as discussed above in Section VII.C.1.b.

b. The Linux version 2.0.1 source code discloses identifying at least some of the automatically expired ones of the records

269. The Linux v2.0.1 code discloses identifying at least some of the automatically expired ones of the records when the linked list is accessed.

270. As discussed above in Section VII.C.1.c, the *rt_cache_add* function identifies automatically expired records while the linked list is being accessed in search of a duplicate record to the one just inserted.

c. The Linux version 2.0.1 source code discloses removing at least some of the automatically expired ones of the records from the linked list when the linked list is accessed

271. The Linux v2.0.1 code discloses removing at least some of the automatically expired ones of the records during the same access of the linked list as the one in which the automatically expired records are identified.

272. As discussed above in Section Section VII.C.1.c, the *rt_cache_add* function identifies and removes the automatically expired records from the linked list while the linked list is being accessed in search of a duplicate record to the one just inserted.

d. The Linux version 2.0.1 source code discloses inserting, retrieving or deleting one of the records from the system following the step of removing

273. The Linux v2.0.1 code discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

274. As discussed above in Section VII.C.1.d, the *rt_cache_add* function searches for a duplicate record matching the record inserted into the cache. If such a duplicate record is found, it is deleted from the system. This deletion may occur after expired records have been identified and removed.

4. The Linux version 2.0.1 source code anticipates and/or renders obvious claims 4 and 8 of the '120 Patent

275. The Linux v2.0.1 source code discloses the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.

276. As discussed above in Section VII.C.2, under Bedrock's interpretation of this claim, the Linux v2.0.1 code dynamically determines a maximum number of records to remove.

277. As discussed above in Section VII.C.2, it would have been obvious to one of ordinary skill in the art to combine the Linux v2.0.1 code with the dynamic determination of a maximum number that occurs in Dirks.

D. The Nemes '495 Patent Renders Obvious All Claims of the '120 Patent

278. It is my opinion that the Nemes '495 Patent in combination with various references disclosing the alternative collision resolution technique of external chaining (including Knuth, Kruse, or Stubbs), renders obvious claims 1, 3, 5, and 7 of the '120 Patent and, in combination references disclosing external chaining and Dirks, NRL BSD, Linux v2.0.1, or GCache, renders obvious claims 2, 4, 6, and 8.

279. The '120 Patent attempts to claim an application of on-the-fly removal of automatically expiring records to linked lists generally, as well as linked lists used in the context of hashing with collision resolution based on external chaining. However, the '495 Patent previously disclosed the use of on-the-fly removal of automatically expiring records in hashing with collision resolution based on open addressing via linear probing. I believe that this

reference, in combination with the knowledge of a person of ordinary skill in the art as of January 1997 would render the asserted claims of the '120 Patent obvious. Specifically, I believe that it would have been obvious to modify the teachings of on-the-fly deletion of automatically expiring records in hash tables with collision resolution based on open addressing via linear probing, to perform on-the-fly removal of automatically expiring records in hash tables with collision resolution based on external chaining (and thus arrive at on-the-fly removal of records in linked lists).

280. This is because by the 1970s, by January 1997, and today, it was widely known that only two basic types of collision resolution schemes exist: collision resolution by open addressing, and collusion resolution by chaining/external chaining. This fact was taught in the seminal Knuth treatise in 1973 (see, Knuth, pp. 513, 518), taught in the '120 Patent (see, '120 Patent, col. 1, lines 53-64), taught in the 1987 Kruse and 1985 Stubbs data structures texts referenced in the specification of the '120 Patent (see, Kruse, pp. 202, 206, Stubbs, pp. 324-25). Dr. Nemes confirmed this during his August 31, 2010 deposition (see Nemes Depo. Tr. at 194). Because of this fact, and, as explained above, the fact that collision resolution based on external chaining has advantages over open addressing when deleting records from a hash table, it would have been obvious to try to modify references teaching a method of deletion in hash tables with collision resolution based on linear probing to use collision resolution based on external chaining.

1. The Nemes '495 Patent in combination with various references renders claims 1, 3, 5, and 7 of the '120 Patent obvious

281. The '495 Patent discloses a method for information storage and retrieval using hashing with linear probing. For example, the '495 Patent states that "[t]his invention relates to information storage and retrieval systems." Moreover, the '495 Patent discloses a CPU and RAM

programmed with software for on-the-fly removal of expired records from a hash table with linear probing (see ‘495 Patent, col. 3, line 15-col.4, line 15). It would have been obvious to a person of ordinary skill in the art at the time of the filing of the application for the ‘120 Patent to modify the teachings of the ‘495 Patent with respect to on-the-fly removal of automatically expiring records in a hash table with collision resolution based on linear probing, to arrive at a hash table with on-the-fly removal of automatically expiring records where collision resolution was based on external chaining. In making such modifications, the resulting system and methods would also necessarily disclose on-the-fly removal of automatically expiring records from a linked list.

282. First, as discussed in Section III above, there are only two types of collision resolution schemes in existence: collision resolution based on open addressing, and collision resolution based on external chaining. A person of ordinary skill in the art in January 1997 would appreciate that among the two types of methods of collision resolution, external chaining is the simpler of the two methods to implement and is the one more likely to provide higher performance storage and retrieval operations.

283. For example, the ‘495 Patent presents a Pascal pseudo-code implementation of hashing of automatically expiring records where collision resolution is based on linear probing. The ‘120 Patent presents a Pascal pseudo-code implementation of hashing of automatically expiring records where collision resolution is based on external chaining. As both the ‘495 Patent and the ‘120 Patent were awarded to the same sole inventor, it is reasonable to assume the same person, Richard Nemes, developed both codes. Thus, the codes can be compared via a simple complexity metric such as number of lines of code (ignoring comments and blank lines). Such a comparison shows that the pseudo code for linear probing contains more lines of code than the

external chaining version. In particular, with regard to the function for actually removing automatically expired records from the hash table (ignoring the complexity of locating such records), the pseudo code for removing an expired record from the hash table when collision resolution is based on linear probing consists of 30 lines of Pascal pseudo code while the remove procedure for removing an expired record from the hash table when collision resolution is based on external chaining consists of only 12 lines (see, '495 Patent, col. 11, lines 18-65, and '120 Patent, cols. 13-14). However, more importantly, the code for removing an expired record from the hash table when collision resolution is based on external chaining contains no loops. Of the 12 lines of code, during the actual execution of the code, either 2 or 5 Pascal statements will be executed.[14] This will be true independent of the size of the hash table. In marked contrast, the code for removing an expired record from the hash table when collision resolution is based on linear probing contains *nested* loops (contains loop within loops). Thus, of the 30 lines of code in the removal procedure when collision resolution is based on liner probing, during the actual execution of the code, the body of the inner-most loop, containing 14 lines of code, will be executed repeatedly depending on the size of the hash table and the number of non-empty hash buckets. For example, for a hash table with a million records, the removal code for the hash table in the '495 Patent could execute millions of lines of code. In contrast, the removal code for the hash table in the '120 Patent would execute either 2 or 5 lines of code. It is plainly apparent that the pseudo code for deleting records from a hash table with collision resolution based on linear probing is significantly more complex and costly in time to execute than the corresponding code

[14] The pseudo code for removal of an expired record in the '120 Patent contains a call to a "dispose" function to deallocate the node being removed. The '120 Patent does not provide the pseudo code for this function, however, I note that under the Court's construction of "removing … from the linked list," deallocation of the memory associated with the record to be removed is

for deleting records from a hash table with collision resolution based on external chaining. The former pseudo code includes nested loops wherein colliding records have to be rehashed and the table must be traversed, whereas the latter code only requires simple pointer adjustment.

284. This comparison confirms the common knowledge that existed among persons of ordinary skill in the art in 1997 that hashing with collision resolution based on external chaining is simpler to implement than hashing with collision resolution based on open addressing/linear probing and will provide better runtime performance. Thus, the complexity comparison above is not surprising and in fact would be predictable. As discussed in Section III, it was well known that external chaining would provide lower implementation complexity and lower runtime cost (execute faster) than linear probing. For at least these reasons, it would have been immediately obvious for a person of ordinary skill in the art to apply the teachings of the '495 Patent to a hash table with collision resolution based on external chaining. In addition, applying the teachings of the '495 Patent to a hash table with collision resolution based on external chaining would require nothing more than the ordinary skill of a person of ordinary skill in the art as of January 1997.

285. Indeed, in Knuth's seminal work, "The Art of Computer Programming, Volume 3, Searching and Sorting," when discussing the problem of collisions in hash tables, Knuth states:

> We have observed that some hash addresses will probably be burdened with more than their share of keys. Perhaps <u>the most obvious way to solve this problem</u> is to maintain M linked lists, one for each possible has address. (See, Knuth, p. 513, emphasis added.)

Thus, when confronted with a need or desire for a hashing solution, Knuth confirms that it would have been obvious to try a collision resolution scheme based on external chaining. That is, adapting a collision resolution scheme based on open addressing/linear probing, to one based on

---

not required. Nonetheless, I have included the dispose function in the count of the Pascal

external chaining would have been a predictable variation. Moreover, the application of collision resolution based on external chaining to a hashing solution previously using collision resolution based on open addressing would be no more than the predictable use of a prior art element according to its established function and would yield a predictable result (a functionally equivalent hash table and one that likely provided better storage and retrieval performance).

286. This significant difference in complexity and performance was noted in Knuth. As Knuth stated in a section concerning deletion from a hash table where certain complexities were encountered when deleting elements from a hash table with collision resolution based on linear probing:

> Of course when chaining is used with separate lists for each possible hash value, deletion causes no problems since it is simply a deletion from a linked linear list. (See, Knuth, p. 527, emphasis added.)

Thus, Knuth confirms that if deletion of records from a hash table were a concern, as they were in the '495 Patent, it would have been obvious to try a collision resolution scheme based on external chaining because fewer (or no) problems could be expected and, as explained in Section III.C, better performance would likely result. Kruse confirmed that deletion from hash tables using open addressing (such as linear probing) is awkward and should be avoided:

> With the methods we have so far studied for hash tables [open addressing], deletions are indeed awkward and should be avoided as much as possible. (See, Kruse, p. 206, emphasis aded.)

In contrast, with respect to deletions from a hash table with collision resolution based on external chaining, Kruse states:

> Finally, deletion becomes a quick and easy task in a chained hash table [a hash table with collision resolution based on external chaining]. Deletion proceeds in exactly the same way as deletion from a simple linked list. (See, Kruse, p. 206, emphasis added.)

---

statements.

287. Further, as discussed in Section III, a person of ordinary skill in the art in January 1997 would appreciate that often times a collision resolution scheme based on external chaining is preferable to one based in open addressing/linear probing. For example, in applications where it is expected that the storage capacity of a hash table may have to be increased over time, collision resolution based on external chaining may be desirable to collision resolution based on linear probing as the latter requires an expansion of the actual hash table and the rehashing of all the elements, whereas the former approach accommodates growth by simply allowing the chains to grow longer.

288. The fact that collision resolution based on external chaining is often preferable to collision resolution based on linear probing is acknowledged in the ‘120 Patent (see, ‘120 Patent, col. 2, lines 38-51). In particular, the ‘120 Patent acknowledges that these facts were well known and discussed in the Knuth and Kruse texts (see, ‘120 Patent, col. 2, lines 37-40). The Stubbs text also discusses the benefits of external chaining over linear probing (see, Stubbs p. 325). The ‘120 Patent further acknowledges that “hashing techniques for dealing with expiring data that do not use external chaining prove wholly inadequate for certain applications” (see, ‘120 Patent, col. 2, lines 41-44). Thus, the ‘120 Patent confirms that it would have been obvious for a person of ordinary skill in the art dealing with expiring data in January 1997 and aware of the teachings of the ‘495 Patent, to modify the teachings of the ‘495 Patent to use collision resolution based on external chaining rather than linear probing. Related to these disclosures in the ‘120 Patent, the Nemes ‘499 Patent discloses external chaining and teaches that deletion of records from an external chain of a hash table is “easy” (see, ‘499 Patent, col. 8, lines 53-58). These teachings would provide further motivation for a person of ordinary skill in the art to have considered adapting the teachings of the ‘495 Patent to employ external chaining.

289. Importantly, while the '120 Patent states that "[t]he methods of the above-mentioned patent [the '495 Patent] are limited to arrays and cannot be used with linked lists due to the significant difference in the organization of the computer's memory," this statement is not to be understood as a statement that a person of ordinary skill in the art would be unable to adapt the teachings of the '495 Patent to employ collision resolution using external chaining. (See, '120 Patent at col. 2, lines 48-51.) In fact, to the contrary, given the simplicity of external chaining, a person of ordinary skill in the art would have no trouble modifying the teachings of the '495 Patent to use linked lists. While the exact instructions given in the '495 Patent for an embodiment of the '495 Patent (*i.e.*, the PASCAL programming language code appearing columns 9-11 of the '495 Patent), cannot be used with linked lists without modification, a person of ordinary skill in the art in January 1997 could easily adapt the code to transform the solution to one using hashing with collision resolution based on external chaining.

290. Indeed, based on my nearly 30 years of experience teaching programming and data structures, in January 1997, it is my opinion that certain persons of *less* than ordinary skill in the art would have had little trouble converting the hashing solution disclosed in the PASCAL embodiment of the '495 Patent into a corresponding solution using linked lists and external chaining. For example, in January 1997, at a minimum, freshman/sophomore level undergraduate computer science majors at UNC who had successfully completed a standard data structures course would certainly be able to complete this task. During this time frame I had personally observed teams of undergraduates implement a hashing solution from scratch (*i.e.,* without the aid of a reference implementation such as the '495 Patent) within the space of an hour. In addition, the effort required to complete this task would be on the order of hours, not days or weeks. I again note that in 1973, Knuth felt that hashing with collision resolution based

on external chaining was such a straightforward combination of standard techniques that he declined to present a complete solution in his text. As Knuth stated:

> This method [hashing with external chaining] is a straightforward combination of techniques we have discussed before, so we do not need to formulate a detailed algorithm for chained scatter [hash] table. (See, Knuth, p. 513, emphasis added.)

Thus, implementing a collision resolution scheme based on external chaining would have been well within the level of skill of a person of ordinary skill in the art in the 1997 time frame.

291. Thus, when encountering the ‘495 Patent, for the reasons given above and in Section III, a person of ordinary skill in the art in 1997 would have been motivated to adapt its teaching to collision resolution based on external chaining at least because external chaining was well known to provide at least:

- lower implementation complexity,
- lower search times for successful searches (higher search performance),
- lower search times for unsuccessful searches,
- lower removal times,
- better adaptability for dynamic growth in size of the hash table, and
- lower memory requirements when storing records.

In addition, a person of ordinary skill in the art in 1997 would recognize that there was little to no advantage to using hashing with collision resolution based on linear probing in an application requiring on-the-fly removal of records from an in-memory hash table.

292. As further evidence that the claims of the ‘120 Patent are obvious in light of the ‘495 Patent, I note that when Bell Communications Research Inc. (“Bellcore”), the assignee of the ‘495 Patent originally prepared the application for the ‘495 Patent, a version of the application included the paragraph:

> It is to be understood that the present invention will be described in connection with linear probing with open addressing only for convenience and because such a collision-resolution strategy is very commonly used. The techniques of the present invention can just as readily applied [sic] to such other forms of collision-resolution strategies by modifications readily apparent to those skilled in the art. (See TELECORDIA00000253.)

Thus, prior to the application for the ‘495 Patent, Bellcore understood that the teachings of the ‘495 Patent could be readily adapted to other forms of collision resolution. In addition, a person of ordinary skill in the art would understand this statement to mean that the techniques of the ‘495 Patent, on-the-fly deletion of automatically expiring records from a hash table can be readily applied to such other forms of collision resolution strategies such as collision resolution based on external chaining. Moreover, a person of ordinary skill in the art would understand that nothing other than ordinary skill would be required to adapt the teachings of the ‘495 Patent to hashing with collision resolution based on external chaining. Indeed, based on my experience teaching undergraduates, I have no doubt that given the ‘495 Patent, undergraduate computer science majors in a sophomore-level data structures course could conceive of, and implement, on-the-fly deletion of automatically expiring records from a hash table with collision resolution based on external chaining. The ‘495 Patent, like the seminal Knuth reference, gives an implementation of its teachings for collision resolution based on linear probing and leaves the easier, more obvious implementation of collision resolution based on external chaining as an exercise for the reader. A person of ordinary skill in the art in 1997 could easily complete this exercise and most likely do so in a mere matter of hours.

293. I understand that at his first deposition in August 2010, Dr. Nemes, the named inventor of both the ‘120 and ‘495 Patents, was asked about the above passage and did not know why the passage was deleted from the eventual application submitted to the USPTO. However, four months later in December 2010, after reviewing the transcript from his former employer

Telcordia's deposition (see, Nemes Depo. Tr. at 880), Dr. Nemes testified that more than 20 years earlier he deleted the paragraph above from the application for the '495 Patent because,

> So I crossed this out because the techniques of the present invention which center around Knuth's algorithm R are applicable only to linear probing. They are not applicable to other forms of open addressing such as quadratic probing, random probing and certainly not applicable to linked lists. Knuth's algorithm R which is at the heart of the technique in the '495 patent is applicable strictly to open addressing using the linear probing technique. (See, Nemes Dep. Tr., pp. 882, line 20 – 883, line 6.)

294. I note that this testimony is directly at odds with the teaching of the '495 Patent. Knuth's Algorithm R is a sketch of an algorithm to delete items from a hash table with collision resolution based on linear probing. Algorithm R appears on page 527 of Knuth (and no other algorithm appears on page 527). The '495 Patent references Algorithm R, however, the patent explicitly states that Algorithm R (and similar algorithms) can only be used the database is off line and thus not available for use:

> In the prior art, such storage space contamination was avoided by deletion procedures that eliminated deleted records by replacing the deleted record with another record in the collision-resolution chain of records and thus close the chain without leaving any deleted records. One such procedure is shown in the aforementioned text by Knuth at page 527. <u>Unfortunately, such non-contaminating procedures, due to the necessity for successive probes into the storage space, take so much time that they can be used only when the data base is off line and hence not available for accessing</u>. (See, '495 Patent, col. 2, lines 11-22, emphasis added.)

295. Thus, Dr. Nemes's testimony, made more than 20 years after the paragraph in question was written, directly contradicts the '495 Patent. Despite Dr. Nemes 2010 testimony about a detail of his 1988 patent application, Knuth's Algorithm R is not the "heart" of the '495 Patent. First, whereas the '120 Patent, written by Dr. Nemes, does contain an explicit indication as to what "the heart of the technique" is (see, '120 Patent, cols. 11-12), the '495 Patent contains

no such statement. There is nothing in the ‘495 Patent to suggest that Knuth’s Algorithm R is the “heart” of the ‘495 Patent and there is clear evidence in the ‘495 Patent that Knuth’s Algorithm R cannot be used in an on-the-fly algorithm such as that taught by the ‘495 Patent. Second, nowhere in the ‘495 Patent is any indication given that Knuth’s Algorithm R is used in the ‘495 Patent. (Indeed, if Knuth’s Algorithm R really was used in the ‘495 Patent, and was the “heart” of the patent, this would on its face raise serious questions as to the validity of the ‘495 Patent.) The SUMMARY OF THE INVENTION section of the ‘495 Patent explains what the real heart of the ‘495 Patent is: “during normal data insertion or retrieval probes into the data store, the expired, obsolete records are identified and removed in the neighborhood of the probe.” Even if the ‘495 Patent discloses using Algorithm R as one means of removing the expired records, it is the fact that the removal happens on-the-fly that is the heart of the patent, not the particular algorithm used to remove each expired record. In any event, the ‘495 Patent makes clear that Knuth’s algorithm R is not suitable for on-the-fly removal.

296. I believe that the observation Bellcore made in 1988 was correct then and is correct now. The techniques of the ‘495 Patent can just as readily be applied to other forms of collision resolution strategies, including external chaining, by modifications readily apparent to those skilled in the art.

297. I am aware that in granting the application for the ‘120 Patent, the USPTO considered the ‘495 Patent. In particular, I understand that the claims of the ‘120 Patent were originally rejected in a “double patenting” rejection over the ‘495 Patent. However, this rejection was based in large part on the appearance of the words “chain” and “chains” in the ‘495 Patent and the Examiner’s misinterpretation of the “chain” of the ‘495 Patent to connote a linked (or “chained”) list (see, ‘120 Patent File History, April 1998 Office Action). In response, the

Applicant, Richard Nemes, responded that "chains" as used in the '495 Patent were not linked lists. Indeed, in allowing the '120 Patent the Examiner stated that "[a]lthough the prior art of record (Nemes, '495 reference) teaches the use of chains of records and the deletion of records, the Applicant, in the Response dated August 11, 1998, Paper No.5, provided arguments as to why the chain of records as taught in the '495 reference is not the same as the linked list as claimed" (see, '120 File History, Notice of Allowability). However, what Dr. Nemes never disclosed, and what the Examiner never heard, were the facts that while "chains" (as used in the '495 Patent) and external chaining are not literally the same, there are the only two methods of resolving collisions in hash tables (the two methods being those based on open addressing ["chains" of the '495 Patent] and external chaining). Further, Dr. Nemes never disclosed, and the Examiner never heard, that the very references cited in the application for the '120 Patent (and in the '495 Patent), namely Knuth and Kruse, both teach that when deletion of records from hash tables is of concern, use of open addressing and/or linear probing will incur complexity and performance problems not found with external chaining while external chaining provides numerous clear benefits over open addressing. For example, Dr. Nemes never disclosed, and the Examiner never heard, that when considering a hashing solution the Knuth reference cited in the application for the '120 Patent (and in the '495 Patent) taught that "the most obvious way to solve this problem [of collisions]" is to use external chaining (see, Knuth, p. 513), while the Kruse reference cited in the application for the '120 Patent (and in the '495 Patent) taught that when using open addressing, "deletions are indeed awkward and should be avoided as much as possible" (see, Kruse, p. 206). There is no evidence to suggest the Examiner was ever provided with this important evidence demonstrating the obviousness of the use of external chaining in hash tables and linked lists generally.

298. According to Dr. Nemes' deposition testimony, the '495 Patent grew out of work he was doing on the Line Information Data Base (LIDB) at Bellcore (see, Nemes Depo. Tr., p. 457, line 12 to p. 460, line 20). Dr. Nemes testified that LIDB used a hash table with linear probing because of the limitations of the particular medium in which the data base was being stored:

> Q. The sequence of buckets on a disk in an LIDB system, does that sequence – did that sequence restrict the types of collision resolution techniques you could use?
>
> A. Well, the goals of the system which were performance I think dictated the limitation. We wanted to be able to get as much as we could in a single disk read. It's very expensive in time to access a portion of the disk. You want to get as much as you can and you want to avoid having to go to the disk more often than you really have to.
>
> So that being a primary goal of the LIDB database system I think limited the implementation for me at that time.

Nemes Depo. Tr. p. 479, line 21 to p. 480, line 10.

299. For a disk-resident hash table, it may be the case that collision resolution based on linear probing (or open addressing generally) is the most appropriate scheme to use. These would be because unlike memory (RAM) resident data structures where the processor can directly use the data structure, disk based data structures require that the data they contain first be read from disk and copied into memory before the data can be used. Thus, for time sensitive applications, disk-resident data structures must be arranged on disk so that elements that are likely to be used together are close to one another on disk and as a result can be read into memory (ideally) in a single operation. In an environment such as this, an array data structure would typically provide better storage and retrieval performance than a linked list because array elements that are adjacent or close to one another in the array are more likely to be stored close to one another on disk (and thus can be read in a single disk operation). In contrast adjacent elements of a disk-resident linked list are less likely to stored together on disk because of the less structured manner

in which storage is allocated for nodes of a linked list. Because of this, multiple disk operations may be required to read multiple elements of a linked list into memory.

300. For these reasons, in the context of a disk-based system such as that developed at Bellcore, basing collision resolution on open addressing likely made sense. However, nonetheless, the embodiments of the invention of the ‘495 Patent disclosed in the patent are of memory-based hash tables (see, *e.g.*, ‘495 Patent, cols. 8-11). For memory-resident hash tables, such as those disclosed in the ‘495 Patent, a person of ordinary skill in the art would have been motivated to base collision resolution on external chaining rather than open addressing for all the reasons stated in Section III.

301. Finally, given an example that Dr. Nemes himself used to illustrate why an issued patent would be invalid for reasons of obviousness, the ‘120 Patent is surely obvious.

302. I understand that Dr. Nemes developed an example of the obviousness of a patent to use in a class he taught at Pace University on computer security. At his deposition, Dr. Nemes testified that in explaining obviousness to his class he had used the example of a well-known algorithm for sorting a sequence of numbers known as *bubble sort*. Briefly, in bubble sort, an array of numbers is sorted by traversing the array from left to right (from the first array element to the last array element) to first find the largest element in the array and then re-traversing the array to find the second largest element, the third largest element, *etc*. Dr. Nemes obviousness example was that in his opinion, one could not patent a bubble sort method that sorted an array in the “reverse” direction, that is by traversing the array from right to left (from the last element in the array to the first) instead of from left to right. By Dr. Nemes estimation, such a method would be an obvious variation of the well-known bubble sort method because reverse traversal of an array for any purpose was known in the prior art (see, Nemes Dep. Tr., p. 595, line 22 to p.

596 line 1). In particular, “reverse bubble sort” would be obvious because it would have been obvious to take two known things in the art and combine them (see, Nemes Dep. Tr., p. 601, lines 13 to p. 602 line 7), and that a person of ordinary skill in the art’s imagination would be broad enough to convert a bubble sort that sorts in one direction to a bubble sort that sorts in the other direction (see, Nemes Dep. Tr., p. 603, line 23 to p. 604, line 7).

303. If a method to bubble sort a list of numbers in reverse order is obvious because a person of ordinary skill in the art’s imagination is broad enough to conceive of this variation, then given the teachings of the ‘495 Patent, the application of hashing with collision resolution based on external chaining to on-the-fly deletion of automatically expiring records from a hash table is obvious. In fact, a person of ordinary skill in the art’s imagination would not even have to be “broad enough” to conceive of this obvious variation because seminal references such as Knuth, as well as undergraduate textbooks, all teach both the use of collision resolution based on external chaining, as well as the advantages of external chaining over schemes such as open addressing or linear probing. No imagination is necessary to conceive of adapting the teachings of the ‘495 Patent to collision resolution based on linear chaining. It would be an obvious, predictable variation to attempt and nothing more than ordinary skill in the art would be required to make such an adaptation and reduce that adaptation to practice.

a. The ‘495 Patent in combination with various references discloses the “linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring” limitation of claim 1 and the “hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring”

304. The ‘495 Patent discloses a hashing means with linear probing. It claims “an information storage and retrieval system using hashing techniques to provide rapid access to the

records of said system and utilizing a linear probing technique to store records with the same hash address." (See, '495 Patent at col. 11, line 67-col.12, line 3.) It also describes hashing with linear probing in detail (see, '495 Patent at cols. 1-2).

305. The '495 Patent teaches automatically expiring records (see, '495 Patent, Abstract, col. 1, lines 57-60, col. 2, lines 35-38, col. 4, lines 23-28, col. 11, line 68 – col. 12, line 5, col. 12, lines 37-42). For example, the '495 Patent discloses that "[i]n some common types of data storage systems, data records become obsolete merely by the passage of time or by the occurrence of some event. If such expired, lapsed or obsolete records are not removed from the storage table, they will, in time, seriously degrade or contaminate the performance of the retrieval system" (see, '495 Patent col. 4, lines 23-28). Claim 1 of the '495 Patent claims an information storage and retrieval system with "at least some of said records automatically expiring" (see '495 Patent col. 12, lines 3-5). Similarly, Claim 5 of the '495 Patent claims a method for storing and retrieving information records with "at least some of said records automatically expiring" (see '495 Patent col. 12, lines 40-41).

306. It would have been obvious to adapt an embodiment of the '495 Patent to use external chaining instead of linear probing as disclosed in various references, including Knuth, Kruse, and Stubbs, because (1) Knuth said it was obvious to try, and (2) '120 Patent and cited texts claim there are well known shortcomings to the use of linear probing. It's a predictable combination that would yield a predictable result and the combination was well within the level of skill of a person of ordinary skill in the art. Where external chaining is used to resolve collisions rather than linear probing, the techniques of the '495 Patent would be applied to linked lists of records having the same hash address.

b. The Nemes ‘495 Patent in combination with various references discloses the “record search means utilizing a search key to access the linked list” limitation of claim 1 and the “record search means utilizing a search key to access a linked list of records having the same hash address” limitation of claim 5

307. The ‘495 Patent teaches a record search means utilizing a search key to access a chain of records (see, ‘495 Patent, cols. 10-11, col. 12, lines 6-7, FIG. 3). Figure 3, which describes a search table procedure that traverses the chains of a hash table with linear probing in reverse searching for a target record, is almost identical to Figure 3 of the ‘120 Patent. The only differences between the two figures are that (1) the ‘495 Patent relates to a target chain, whereas the ‘120 Patent relates to a list, (2) the ‘495 Patent starts at the end of the chain and proceeds in reverse, whereas the ‘120 Patent starts at the head of a linked list and goes forward, and (3) the ‘495 Patent saves the location of the empty cell that can be used for an insertion, whereas the ‘120 Patent does not. These differences are due to the fact that the ‘495 Patent relates to a hash table with linear probing.

308. Claim 1 of the ‘495 Patent also claims “a record search means utilizing a search key to access a chain of records having the same hash address.” Though this “chain of records” refers to a chain of records in a hash table with linear probing, if external chaining were applied to resolve collisions in the hash table instead of linear probing, the result would be a record search means utilizing a search key to access a linked list of records having the same hash address.

309. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining instead of linear probing. Such an adaptation would have resulted in a search table procedure like the one found in Figure 3 of the ‘120 Patent.

c. The Nemes ‘495 Patent in combination with various references discloses the “record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed” limitation of claims 1 and 5

310. The ‘495 Patent teaches a record search means including means for identifying and removing all expired ones of said records from said chain of records each time said chain is accessed to search for a target record (see, ‘495 Patent, col. 4, line 65-col. 5, line 49, cols. 10-11, FIG. 3, col. 12, lines 8-11). As previously discussed, FIG. 3 of the ‘495 Patent is nearly identical to FIG. 3 of the ‘120 Patent, with the differences due to the fact that the ‘495 Patent relates to linear probing and the ‘120 Patent relates to external chaining.

311. Claim 1 of the ‘495 Patent discloses a “record search means including means for identifying and removing all expired ones of said records from said chain of records each time said chain is accessed.” Removing all expired records encompasses removing at least some of the expired records. Moreover, as discussed above, the chain of records described relates to linear probing, but would be equivalent to a linked list of records if external chaining were used as the method of collision resolution.

312. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining instead of linear probing. Such an adaptation would have resulted in a search table procedure like the one found in Figure 3 of the ‘120 Patent.

d. The Nemes '495 Patent in combination with various references discloses the "means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list" limitation of claim 1 and the "means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records" limitation of claim 5

313. The '495 Patent teaches a record search means including using the record search means for inserting retrieving and deleting records from said system and, at the same time, removing all expired ones of said records in the accessed chains of records (see, '495 Patent, col. 7, line 3-col. 8 line 16, cols. 9-10, FIGS. 5-7, col. 12, lines 12-16). FIG. 5 of the '495 Patent is nearly identical to FIG. 5 of the '120 Patent except that it refers to a "target chain" instead of a "target list," checks the table load before deciding whether to insert a record rather than checking whether memory is available, and it inserts the actual record into the hash table as one normally would in a hash table with linear probing, rather than as one would with a hash table with external chaining. Similarly, FIGS. 6 & 7 of the '495 Patent are nearly identical to FIGS. 6 & 7 of the '120 Patent except for the reference to a target chain instead of a target linked list. If the linear probing technique of the '495 Patent were replaced with an external chaining technique, these figures would be equivalent.

314. Claim 1 of the '495 Patent also claims "means, utilizing the record search means, for inserting, retrieving and deleting records from said system and, at the same time, removing all expired ones of said records in the accessed chains of records." (See, '495 Patent, col. 12, lines 12-16).

315. As already discussed, it would have been obvious to adapt an embodiment of the '495 Patent to use external chaining instead of linear probing. Such an adaptation would have resulted in insert, retrieve, and delete procedures like the ones found in the '120 Patent.

2. The ‘495 Patent in combination with Dirks, NRL BSD, Linux v2.0.1, or GCache renders obvious claims 2 and 6 of the ‘120 Patent

316. The ‘495 Patent in combination with Dirks, NRL BSD, Linux v2.0.1, or GCache renders this claim obvious. The disclosures of NRL BSD, Linux v2.0.1, and GCache are described above. One of ordinary skill in the art would have been motivated to combine the dynamic determination of any of these pieces of art with the ‘495 Patent in order to limit the amount of cleanup done by the record search means.

317. Additionally, it is a fundamental concept in computer programming to make decisions in a program based on factors internal or external to the system being implemented. Thus, arguably, virtually any determination that is made in a computer system is a dynamic determination as that term has been construed. If processing load were a concern in the ‘495 Patent, it would have been obvious to one of ordinary skill in the art to introduce a dynamic determination of the number of expired records to remove from the hash table to limit the amount of clean up work performed during the traversal of the list. It would have been obvious to limit the amount of time spent deleting expired records to, for example, limit the time spent in any one instance of the record search means.

318. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Dr. Nemes concedes that such dynamic determination was obvious when he states in the ‘120 patent that “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one” (see, ‘120 Patent, col. 7, lines 10-15).

319. Furthermore, the ‘495 Patent in combination with the Dirks ‘214 Patent renders these claims obvious. The disclosures of the Dirks ‘214 Patent are described in Sections VI.F and VII.E. The Dirks ‘214 Patent teaches a simple formula for bounding the number hash table entries considered when deleting expired entries as a fraction of the total number of entries in the hash table and the number of active processing entities. A person or ordinary skill in the art would recognize that this simple bounding ratio could be applied to the bounding the number of removals in the hash table processing by, for example, computing the ratio of the number of entries in the hash table to the number of active processes.

3. The ‘495 Patent in combination with various references renders obvious claims 3 and 7 of the ‘120 Patent

320. The ‘495 Patent discloses a method for storing and retrieving information records using hashing techniques to provide rapid access to said records and utilizing a linear probing technique to store records with the same hash address (see, ‘495 Patent, col. 1, lines 10-12, col. 12 lines 37-42, lines 648-1087, 1545-1560).

321. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining as described in Knuth, Kruse, and Stubbs instead of linear probing to store records with the same hash address. In such an instance, the method for storing and retrieving information records would use a linked list to store and provide access to the records.

322. The ‘495 Patent also discloses at least some of the records in the hash table automatically expiring (see, ‘495 Patent, col. 1, lines 57-60, col. 4, lines 23-28, col. 12, lines 40-42). The ‘495 Patent explains that “[i]n some common types of data storage systems, data records become obsolete merely by the passage of time or by the occurrence of some event” (see ‘495 Patent, col. 4, lines 23-25).

a. The ‘495 Patent in combination with various references discloses accessing the linked list of records and accessing a linked list of records having the same hash address

323. Claim 5 of the ‘495 Patent discloses “accessing a chain of records having the same hash address” (see, ‘495 Patent, cols. 10-11, col. 12, lines 43-44, FIG. 3). Moreover, as discussed above in Section VII.D.1.b, the ‘495 Patent discloses utilizing a search key to access a chain of records in a hash table.

324. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining as described in Knuth, Kruse or Stubbs instead of linear probing to store records with the same hash address. In such an instance, the claimed method of the ‘495 Patent would access a linked list of records having the same hash address rather than the chain of records claimed.

b. The ‘495 Patent in combination with various references discloses identifying at least some of the automatically expired ones of the records

325. Claim 5 of the ‘495 Patent discloses “identifying the automatically expired ones of said records” (see, ‘495 Patent, col. 4, line 65-col. 5, line 49, cols. 10-11, FIG. 3, col. 12, lines 45-46). Moreover, as discussed above in Section VII.D.1.c, the ‘495 Patent discloses identifying and removing expired records from a chain of records each time the chain of records is accessed to search for a target record.

326. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining as described in Knuth, Kruse or Stubbs instead of linear probing to store records with the same hash address. In such an instance, the claimed method of the ‘495 Patent would identify automatically expired records in the linked list of records having the same hash address rather than the chain of records claimed.

c. The ‘495 Patent in combination with various references discloses removing at least some of the automatically expired ones of the records from the linked list when the linked list is accessed

327. Claim 5 of the ‘495 Patent discloses “removing all automatically expired records from said chain of records each time said chain is accessed” (see, ‘495 Patent, col. 4, line 65-col. 5, line 49, cols. 10-11, FIG. 3, col. 12, lines 47-48). Moreover, as discussed above in Section VII.D.1.c, the ‘495 Patent discloses identifying and removing expired records from a chain of records each time the chain of records is accessed to search for a target record.

328. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining as described in Knuth, Kruse or Stubbs instead of linear probing to store records with the same hash address. In such an instance, the claimed method of the ‘495 Patent would remove automatically expired records in the linked list of records having the same hash address rather than the chain of records claimed.

d. The ‘495 Patent in combination with various references discloses inserting, retrieving or deleting one of the records from the system following the step of removing

329. Claim 5 of the ‘495 Patent discloses “inserting, retrieving or deleting one of said records from the system following the step of removing” (see ‘495 Patent, col. 7, line 3-col. 8 line 16, cols. 9-10, FIGS. 5-7, col. 12, lines 50-51). Moreover, as discussed above in Section VII.D.1.d, the ‘495 Patent discloses insert, retrieve, and delete procedures that insert, retrieve, or delete a record after identifying and removing expired records found during the access of the chain of records.

330. As already discussed, it would have been obvious to adapt an embodiment of the ‘495 Patent to use external chaining as described in Knuth, Kruse or Stubbs instead of linear probing to store records with the same hash address. In such an instance, the claimed method of

the ‘495 Patent would remove automatically expired records in the linked list of records having the same hash address rather than the chain of records claimed.

4. The ‘495 Patent in combination with Dirks, NRL BSD, Linux v2.0.1, or GCache renders obvious claims 4 and 8 of the ‘120 Patent

331. The ‘495 Patent in combination with Dirks, NRL BSD, Linux v2.0.1, or GCache renders these claims obvious as discussed in Section VII.D.2.

E. The Dirks ‘214 Patent Renders Obvious Claims 2, 4, 6, and 8 of the ‘120 Patent

332. It is my opinion that the Dirks ‘214 Patent in combination with various references previously described renders obvious claims 2, 4, 6, and 8 of the ‘120 Patent.

a. The Dirks ‘214 Patent discloses the “means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records” element of claims 2 and 6 and “the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed” of claims 4 and 8.

333. The Dirks ‘214 Patent describes a means and method of dynamically determining a maximum number of records to remove from a virtual memory page table.

334. The Dirks ‘214 Patent describes a virtual memory manager that removes page table entries on-the-fly. When a process or thread is created or deleted, or in response to some regularly occurring event, a number of entries in the page table are examined to identify and remove expired entries (see, Dirks, col. 7, lines 2-7, col. 8, lines 44-46, col. 9, lines 11-14).

335. The Dirks ‘214 Patent discloses dynamically determines a maximum number of entries in the page table that are examined on any particular sweep of the page table. For example, the Dirks ‘214 Patent states that in one embodiment of the invention, “the total number of entries in the page table and the number of threads and applications that are allowed to be active at any given time.” The factors on which this determination is based are factors internal to

the information storage and retrieval system (see, Dirks, col. 7, lines 14-37). Dirks derives a formula for this calculation (see, Dirks ‘214 Patent, col. 8, lines 29-32):

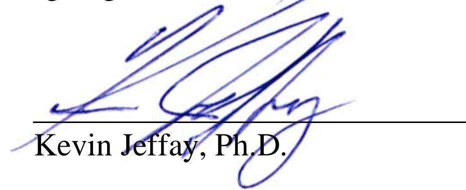
$$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$

336. Moreover, the Dirks ‘214 Patent discloses that “[a]ny other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process,” and that this maximum number “might vary from one step to the next” (see, Dirks, col. 7, lines 38-46).

337. By placing a limit on the number of entries to examine on any particular sweep, the system disclosed in the Dirks ‘214 Patent also determines a maximum number of entries that will be removed, since the system cannot remove more entries than it examines on any particular sweep.

338. It would have been obvious to one skilled in the art that this method and means of dynamically determining a maximum number could be combined with any system doing garbage collection. One skilled in the art would have been motivated to do this for the same reason explained in Dirks. Because the removal of expired records can be an expensive process, placing a hard limit on the number of removals, or garbage collection, occurring at any particular time would have been desirable in any number of the prior art systems and methods described above. Applying this technique described in Dirks to any of these prior art systems was well within the ability of one of ordinary skill in the art in January 1997.

I declare under penalty of perjury that the foregoing is true and correct.



Kevin Jeffay, Ph.D.

January 25, 2011

EXHIBIT A

EXHIBIT A: List of Documents Considered

Documents:

1. Donald E. Knuth, *The Art of Computer Programming*, vol. 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973, produced as the document beginning DEF00006329.

2. Robert L. Kruse, *Data Structures and Program Design,* Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1987, excerpts of which are produced as the documents beginning DEF00005078, DEF00005130.

3. Daniel F. Stubbs and Neil W. Webre*, Data Structures with Abstract Data Types and Pascal*, Brooks/Cole Publishing Company, Monterey, California, 1985, excerpts of which are produced as the documents beginning DEF00005141, DEF00007975.

4. Douglas Comer and Shawn Ostermann, GCache: A Generalized Caching Mechanism, produced as the document beginning RHT-BR00015539.

5. VM-Xinu gcache.c, produced as RHT-BR00015824, DEF00008004-15, KTS0000426-438, and KTS0000439-440.

6. Linux 2.0.1 route.c/route.h, produced as DEF00008567-8605.

7. Linux 1.3.52 route.c/route.h, produced as DEF00001013-1043.

8. Linux 1.3.51 route.c/route.h, produced as DEF00007865-7899.

9. NRL IPv6 key.c/key.h, Alpha release, produced as the documents beginning DEF00007942 and DEF00007971, ftp://ftp.ripe.net/ipv6/nrl.

10. NRL IPv6 key.c/key.h, Alpha-2 release, produced as the document YAHOO507259.

11. U.S. Patent No. 5,893,120, and related prosecution history, produced as BTEX0000174.

12. U.S. Patent No. 5,287,499, produced as the document beginning DEF00004050.

13. U.S. Patent No. 5,121,495, produced as the document beginning GGL-BED00031386.

14. U.S. Patent No. 6,119,214, produced as the document beginning DEF00000797.

15. Transcript of the Deposition of Daniel McDonald and all exhibits used therein.

16. Transcript of the Deposition of Jeffrey Schiller and all exhibits used therein.

17. Transcript of the Depositions of Dr. Richard Nemes and all exhibits used therein.

18. Transcript of the Deposition of Dr. Mark Jones and all exhibits used therein.

19. Transcript of the Deposition of Dr. Shawn Ostermann and all exhibits used therein.

20. Declaration of Alexey Kuznetsov, produced as the document beginning DEF00009284.

21. Document beginning with bates label BTEX124123.

22. Document beginning with bates label TELECORDIA00000152.

23. Bedrock's Second Amended Complaint for Patent Infringement.

24. Plaintiff's P.R. 3-6 Infringement Contentions (Amended 1/12/11) as to Google Inc. and Match.com LLC (various versions).

25. Defendants Joint Amended Invalidity Contentions (12/20/10).

26. The Court's 1/10/11 Claim Construction Order, Case 6:09-cv-00269-LED-JDL, Dkt. 369.

27. "PowerPC Operating Environment Architecture," Book III, Version 2.02, January 2005.

28. http://cs.uttyler.edu/documents/CSCurriculum0809.pdf.

29. http://cs.uttyler.edu/documents/COSCUndergradCourseDesc.pdf.

30. http://cs.uttyler.edu/CourseSyllabi/Undergraduate/COSC%202336.pdf.

31. RFC 1825, Security Architecture for the Internet Protocol, August 1995, http://tools.ietf.org/html/rfc1825.

EXHIBIT B

CURRICULUM VITAE

Kevin Jeffay
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175
(919) 962-1938 Voice, (919) 962-1799 FAX
jeffay@cs.unc.edu
http://www.cs.unc.edu/~jeffay

Education

Ph.D. Computer Science, University of Washington.
Honors: IBM Graduate Fellowship.

M.Sc. Computer Science, University of Toronto.
Honors: University of Toronto Open Fellowship.

B.S. Mathematics with Highest Distinction, University of Illinois at Urbana-Champaign.
Honors: Phi Beta Kappa, James Scholar.

Academic Experience

Gillian T. Cell Distinguished Professor in Computer Science, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 2008–present.

S. S. Jones Distinguished Professor in Computer Science, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 2001–2008.

S. S. Jones Distinguished Term Associate Professor of Computer Science, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 2000.

Associate Professor, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 1996–2000.

Visiting Professor, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1994.

Assistant Professor, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 1989–1995.

Honors and Awards

Favorite Faculty Award, an award given by the Computer Science majors of the 2010 graduating class, May 2010.

Carolina Women's Center Women's Advocacy Award, University of North Carolina at Chapel Hill, Chapel Hill, NC, March 2010.

Gillian T. Cell Distinguished Professor in Computer Science, University of North Carolina at Chapel Hill, College of Arts and Sciences, Chapel Hill, NC, June 2008.

Outstanding Teaching Award, an award given by the Computer Science majors of the 2008 graduating class, May 2008.

Favorite Faculty Award, an award given by the Computer Science majors of the 2004 graduating class, May 2004.

Edward Kidder Graham Outstanding Faculty Award, an award given by the Class of 2004 and the General Alumni Association, April 2004.

Edward Kidder Graham Advisor of the Year Award, an award given by the Class of 2004 and the General Alumni Association, April 2004.

IEEE Service Award, Service to the IEEE Technical Committee on Real-Time Systems, December 2001.

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION.
DISTRIBUTION WITHOUT THE EXPRESS PERMISSION OF KEVIN JEFFAY IS FORBIDDEN.

Outstanding Teaching Award, Computer Science Student Association, University of North Carolina at Chapel Hill, Department of Computer Science, May 2000.

S. Shepard Jones Distinguished Term Professorship, University of North Carolina at Chapel Hill, College of Arts and Sciences, Chapel Hill, NC, March 1999.

Award Papers:

ACM SIGCOMM 2000, 2003.

IEEE International Symposium on High Assurance Systems Engineering 1999, 2000.

IEEE Real-Time Technology and Applications Symposium, 1995.

ACM International Conference on Multimedia, 1994.

Computer Networks and ISDN Systems, 1994.

International Workshop on Network and Operating System Support for Digital Audio and Video, 1991, 1992, 1993, 1997.

Testifying Experience (Within Last Four Years)

nCUBE Corporation (now ARRIS Group, Inc.), v. SeaChange International, Inc., United States District Court for the District of Delaware (C.A. No. 01-011 (LPS)).

Provided deposition on behalf of defendant SeaChange International, Inc.

In the Matter of: Certain Multimedia Display and Navigation Systems and Systems, Components Thereof, and Products Containing the Same, United States International Trade Commission, Investigation No. 337-TA-694.

Provided deposition and trial testimony on behalf of complainant Pioneer Corporation, and Pioneer Electronics (USA), Inc.

In the Matter of: Certain Multimedia Display and Navigation Systems and Systems, Components Thereof, and Products Containing the Same, United States International Trade Commission, Investigation No. 337-TA-694.

Provided deposition and trial testimony on behalf of complainant Pioneer Corporation, and Pioneer Electronics (USA), Inc.

LinkSmart Wireless Technologies, LLC v. T-Mobile USA, Inc., et al., United States District Court for the Eastern District of Texas, Marshall Division (2:08-cv-00264, 00304, 00385, 00026-DF-CE).

Provided deposition testimony on behalf of defendants Cisco Systems Inc., T-Mobile USA, Inc., and related defendants.

Quixtar Inc., v. Signature Management Team, LLC, b/d/a Team, Apollo Works Holdings, Inc. Green Gemini Enterprises, Inc., North Star Solutions, Inc., Northern Lights Services, Inc., Sunset Resources, Inc., and Sky Scope Team, Inc., United States District Court for the District of Nevada (3:07-cv-00505).

Provided deposition testimony on behalf of plaintiff Quixtar Inc.

Netscape Communications Corp., v. ValueClick, Inc., Mediaplex, Inc., FastClick, Inc., Commission Junction, Inc., MeziMedia, Inc., and Web Clients, LLC, United States District Court for the Eastern District of Virginia, Alexandria Division (1:09-cv-225 TSE/TRJ).

Provided deposition testimony on behalf of plaintiff Netscape Communications Corp.

In the Matter of: Certain Automotive Multimedia Display and Navigation Systems, Components Thereof, and Products Containing the Same, United States International Trade Commission, Investigation No. 337-TA-657.

Provided deposition testimony on behalf of respondents Alpine Electronics, Inc., Alpine Electronics of America, Inc., Pioneer Corporation, and Pioneer Electronics (USA), Inc. and testimony at trial on behalf of respondents Pioneer Corporation, and Pioneer Electronics (USA), Inc.

Juniper Networks, Inc., v. Abdullah Ali Bahattab, United States District Court for the District of

Columbia, (1:07-cv-1771-PLF (AK)).

Provided deposition testimony on behalf of defendant Abdullah Ali Bahattab.

Girafa.com, Inc. v. Amazon Web Services LLC, Amazon.com, Inc., Alexa Internet, Inc., IAC Search & Media, Inc., Snap Technologies, Inc., Yahoo! Inc., Smartdevil Inc., Exalead, Inc. and Exalead S.A., United States District Court for the District of Delaware, (07-787 SLR).

Provided deposition testimony on behalf of defendant Exalead, Inc. and Exalead S.A.

Verizon Services Corp., Verizon Communications, Inc., MCI Communications Corporation, and Verizon Global LLC, v. Cox Fibernet Virginia, Inc., Cox Virginia Telcom, Inc., Cox Communications Hampton Roads, LLC, Coxcom, Inc., and Cox Communications, Inc., United States District Court for the Eastern District of Virginia, (1:08CV157 CMH-TRJ).

Provided deposition testimony on behalf of defendant Cox Communications *et al*.

Net2Phone, Inc., v. Ebay, Inc. Skype Technologies SA, Skype, Inc., and John Does 1-10, United States District Court for the District of New Jersey, (06-2469-KSH-PS).

Provided deposition testimony and testimony at an evidentiary hearing on behalf of plaintiff Net2Phone.

Nortel Networks Inc., v. State Board of Equalization of the State of California, Superior Court of California, County of Los Angeles, (BC341568).

Provided both deposition testimony and trial testimony on behalf of Plaintiff Nortel Networks.

Extreme Networks, Inc., v. Enterasys Networks, Inc., United States District Court for the Western District of Wisconsin, (07-C-0229-C).

Provided deposition testimony on behalf of Defendant Enterasys Networks.

Akamai Technologies, Inc. and The Massachusetts Institute of Technology, v. Limelight Networks, Inc., United States District Court for the District of Massachusetts, (06-CV-11109 RWZ).

Provided deposition testimony and trial testimony on behalf of Plaintiffs Akamai Technologies and The Massachusetts Institute of Technology.

Two-Way Media, L.L.C., v. AOL L.L.C., United States District Court for the Southern District of Texas, Corpus Christi Division, (C-04-089).

Provided deposition testimony on behalf of defendant AOL.

Verizon Wireless Personal Communications, L.P., f/k/a Primeco Personal Communications, L.P., v. Gary R. Nikolits, as Property Appraiser of Palm Beach County, Florida, Florida Circuit Court, Fifteenth Judicial Circuit, Palm Beach County, Florida, Civil Division, (05-CA-11462 Division "AE").

Provided deposition testimony and trial testimony on behalf of plaintiff Verizon Wireless.

Express Logic, Inc., v. Green Hills Software, Inc., American Arbitration Association of San Diego, (73 133 Y 00226 06 BRSH).

Provided deposition testimony and arbitration testimony on behalf of defendant Green Hills Software.

Industry Experience

Consultant, Mälardalen University, Vásteras, Sweden, 2001.

Member, Technical Advisory Board, Ganymede Software Inc., Research Triangle Park, NC, 1996–1999.

Consultant and Author, INTEROP Graduate Institute, SOFTBANK Inc., Foster City, CA, 1996–1997.

Consultant, National Science Foundation, Arlington, VA, 1995–present.

Consultant, Monterey Technologies Inc., Cary, NC, 1994–1996.

Consultant, Hewlett-Packard Inc., Ink Jet Components Division, Corvallis, OR, 1993–1994.

Visiting Researcher, IBM T.J. Watson Research Center, Computer Systems Principles Group, Yorktown Heights, NY, 1986.

Consultant, Boeing Aerospace, Kent, WA, 1985–1986.

Software Engineer, R.R. Donnelley & Sons Company Inc., Technical Center, Chicago, IL, 1984.

Computer Programmer, US Army Corps of Engineers Construction Engineering Research Lab, Champaign, IL, 1981–1982.

Community Service

Consultant (unpaid), Children's Museum About the World (now called *Exploris*), Raleigh, NC, 1995–1996.

Professional Activities

Editor

Associate Editor, *Real-Time Systems*, Kluwer Academic Publishers, The Netherlands, 2003-present.

Editor in Chief, *Multimedia Systems*, ACM/Springer-Verlag, Heidelberg, Germany, 2000–2001.

Editorial Board

Journal of Multimedia Tools and Applications, Kluwer Academic Publishers, 1994–1999.

Guest Editor

Computer Communications, special issue on system support for multimedia computing, Volume 18, Number 10, October 1995.

Executive Committees

ACM/SIGCOMM Internet Measurement Conference Steering Committee, 2005-2009.

College of Reviewers, Canada Research Chairs Program, 2004-present.

Statistical and Applied Mathematical Sciences Institute (SAMSI) program on Network Modeling for the Internet, 2002-2004.

IEEE Technical Committee on Real-Time Systems, 2000-present.

ACM Special Interest Group on Multimedia, 2000-2002.

Other Committees

IEEE Distinguished Visitors Program, 2004–2007.

Conference Program Chair

11th International Workshop on Quality-of-Service (Co-Chair), Monterey, CA, June 2003.

Sixth IEEE Symposium on Computers and Communications, Hammamet, Tunisia, July 2001.

21st IEEE Real-Time Systems Symposium, Orlando, FL, November 2000.

Tenth International Workshop on Network and Operating System Support for Digital Audio and Video, Chapel Hill, NC, June 2000.

Seventh ACM International Conference on Multimedia (Co-Chair), Orlando, FL, November 1999.

SPIE/ACM Multimedia Computing and Networking 1999 (Co-Chair), San Jose, CA, January 1999.

Sixth ACM International Conference on Multimedia (Associate Chair), Bristol, UK, September 1998.

IEEE International Conference on Multimedia Computing Systems (Associate Chair), Austin, TX, June 1998.

SPIE/ACM Multimedia Computing and Networking 1998 (Co-Chair), San Jose, CA, January 1998.

SPIE/ACM Multimedia Computing and Networking 1997 (Associate Chair), San Jose, CA, February 1997.

IEEE Workshop on Resource Allocation Problems in Multimedia Systems, Washington D.C., December 1996.

Second IEEE Real-Time Technology and Applications Symposium, Boston, MA, June 1996.

Conference General Chair

ACM SIGCOMM Internet Measurement Conference, San Diego, CA, October 2007.

22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.

Tenth International Workshop on Network and Operating System Support for Digital Audio and Video, Chapel Hill, NC, June 2000.

Third IEEE Real-Time Technology and Applications Symposium, Montreal, Canada, June 1997.

IEEE Workshop on Resource Allocation Problems in Multimedia Systems, Washington D.C., December 1996.

Member of Conference Program Committee

18th IEEE International Conference on Network Protocols, Kyoto, Japan, October 2010.

ACM SIGCOMM 2010, New Delhi, India, August 2010.

17th IEEE International Conference on Network Protocols, Princeton, NJ, October 2009.

9th Passive and Active Measurement Conference 2008, Cleveland, OH, April 2008.

ACM SIGMETRICS 2008, Annapolis, MA, June 2008.

IEEE INFOCOM 2008, Phoenix, AZ, April 2008.

6th ACM SIGCOMM Workshop on Hot Topics in Networks, Atlanta, GA, November 2007.

15th IEEE International Conference on Network Protocols, Beijing, China, October 2007.

5th IEEE Workshop on Embedded Systems for Real-Time Multimedia, Salzburg, Austria, October 2007.

Workshop on Experimental Computer Science, ACM FCRC, San Diego, CA, June 2007.

15th International Workshop on Quality-of-Service, Chicago, IL, June 2007.

17th International Workshop on Network and Operating System Support for Digital Audio and Video, Urbana-Champaign, IL, June 2007.

10th IEEE Global Internet Symposium 2007, Anchorage, AK, May 2007.

ACM SIGMETRICS 2007, San Diego, CA, June 2007.

14th IEEE International Conference on Network Protocols, Santa Barbara, CA, October 2006.

ACM Internet Measurement Conference 2006, Rio de Janeiro, Brazil, October 2006.

Second ACM SIGCOMM Conference on Future Networking Technologies, Lisbon, Portugal, December 2006.

16th International Workshop on Network and Operating System Support for Digital Audio and Video, Newport, RI, June 2006.

IEEE Workshop on Research Directions for Security and Networking in Critical Real-Time and Embedded Systems, San Jose, CA, April 2006.

IEEE INFOCOM 2006, Barcelona, Spain, April 2006.

13th IEEE International Conference on Network Protocols, Boston, MA, November 2005.

15th International Workshop on Network and Operating System Support for Digital Audio and Video, Skamania, WA, June 2005.

IEEE INFOCOM 2005, Miami, FL, March 2005.

SPIE/ACM Multimedia Computing and Networking 2005, San Jose, CA, January 2005.

ACM Multimedia 2004, New York, NY, October 2004.

ACM Internet Measurement Conference 2004, Taormina, Italy, October 2004.

16th EUROMICRO Conference on Real-Time Systems, Catania, Italy, June-July 2004.

14th International Workshop on Network and Operating System Support for Digital Audio and Video, Cork, Ireland, June 2004.

12th International Workshop on Quality-of-Service, Montreal, Canada, June 2004.

10th IEEE Real-time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004.

IEEE INFOCOM 2004, Hong Kong, March 2004.

24nd IEEE Real-Time Systems Symposium, Cancun, Mexico, December 2003.

19th ACM Symposium on Operating Systems Principles, Lake George, New York, October 2003.

15th EUROMICRO Conference on Real-Time Systems, Porto, Portugal, July 2003.

Ninth IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, CA, May 2003.

SPIE/ACM Multimedia Computing and Networking 2003, San Jose, CA, January 2003.

23nd IEEE Real-Time Systems Symposium, Austin, TX, November 2002.

10th International Conference on Network Protocols, Paris, France, November 2002.

Second Workshop on Embedded Software, Grenoble, France, October 2002.

22nd IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation (Performance 2002), Rome, Italy, September 2002.

Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, San Jose, CA, September 2002.

10th International Workshop on Quality-of-Service, Miami, FL, June 2002.

Seventh Global Internet Symposium (held in conjunction with Globecom 2002), Taipei, Taiwan, November 2002.

14th EUROMICRO Conference on Real-Time Systems, Vienna, Austria, June 2002.

ACM SIGMETRICS 2002, Marina del Rey, CA, June 2002.

SPIE/ACM Multimedia Computing and Networking 2002, San Jose, CA, January 2002.

22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.

Sixth Global Internet Symposium (held in conjunction with Globecom 2001), San Antonio, TX, November 2001.

27th EUROMICRO Conference, Warsaw, Poland, September 2001.

Sixth IEEE Symposium on Computers and Communications, Hammamet, Tunisia, July 2001.

Seventh IEEE Real-Time Technology and Applications Symposium, Taipei, Taiwan, June 2001.

Ninth IFIP International Workshop on Quality of Service, Karlsruhe, Germany, June 2001.

SPIE/ACM Multimedia Computing and Networking 2001, San Jose, CA, January 2001.

Fifth Global Internet Mini-Conference (held in conjunction with Globecom 2000), San Francisco, CA, November 2000.

21st IEEE Real-Time Systems Symposium, Orlando, FL, December 2000.

Tenth International Workshop on Network and Operating System Support for Digital Audio and Video, Chapel Hill, NC, June 2000.

Sixth IEEE Real-Time Technology and Applications Symposium, Washington, D.C., June 2000.

SPIE/ACM Multimedia Computing and Networking 2000, San Jose, CA, January 2000.

19th IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1999.

Fourth Global Internet Mini-Conference (held in conjunction with Globecom '99), Rio de Janero, Brazil, November 1999.

Fifth International Workshop on Multimedia Information Systems, Palm Springs, CA, October 1999.

Ninth International Workshop on Network and Operating System Support for Digital Audio and Video,

Basking Ridge, NJ, June 1999.

IEEE Workshop on QoS Support for Real-Time Internet Applications, Vancouver, Canada, June, 1999.

Second ACM Workshop on Internet Server Performance (held in conjunction with SIGMETRICS '99), Atlanta, GA, May 1999.

Seventh IEEE International Workshop on Parallel and Distributed Real-Time Systems, San Juan, Puerto Rico, April 1999.

SPIE/ACM Multimedia Computing and Networking 1998, San Jose, CA, January 1999.

19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

Third Global Internet Mini-Conference (held in conjunction with Globecom '98), Sydney, Australia November 1998.

Sixth ACM International Conference on Multimedia, Bristol, UK, September 1998.

Third IEEE/ACM International Workshop on Multimedia Database Management Systems, Dayton, OH, August 1998.

Eighth International Workshop on Network and Operating System Support for Digital Audio and Video, Cambridge, UK, July 1998.

SPIE Interactive Multimedia Services and Equipment, Zurich, Switzerland, May 1998.

IEEE Workshop on Dependable and Real-Time E-Commerce Systems, Denver, CO, June 1998.

Fourth IEEE Real-Time Technology and Applications Symposium, Denver, CO, June 1998.

IEEE International Conference on Multimedia Computing Systems, Austin, TX, June 1998.

SPIE/ACM Multimedia Computing and Networking 1998, San Jose, CA, January 1998.

16th IEEE Symposium on Reliable Distributed Systems, Durham, NC, October 1997.

Third IEEE Real-Time Technology and Applications Symposium, Montreal, Canada, June 1997.

IEEE Real-Time Education Workshop, Montreal, Canada, June 1997.

Seventh International Workshop on Network and Operating System Support for Digital Audio and Video, St. Louis, MO, May 1997.

17th International Conference on Distributed Computing Systems, Distributed Real-Time Systems Track, Performance of Distributed Systems Track, Distributed Multimedia Track, Baltimore, MD, May 1997.

Fifth IFIP International Workshop on Quality of Service, New York, NY, May 1997.

Third International Conference on Computer Science & Informatics, Raleigh-Durham, NC, March 1997.

SPIE/ACM Multimedia Computing and Networking 1997, San Jose, CA, February 1997.

Fourth ACM International Conference on Multimedia, Boston, MA, November 1996.

Second USENIX Symposium on Operating Systems Design and Implementation, Seattle, WA, October/November 1996.

ACM SIGSOFT '96: Fourth Symposium on the Foundations of Software Engineering, San Francisco, CA, October 1996.

Second IEEE International Workshop on Multimedia Database Management Systems, Blue Mountain Lake, NY, August 1996.

Second IEEE Real-Time Technology and Applications Symposium, Boston, MA, June 1996.

Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, Zushi, Japan, April 1996.

First International Workshop on Real-Time Databases: Issues and Applications, Newport Beach, CA, March 1996.

SPIE Multimedia Computing and Networking 1996, San Jose, CA, February 1996.

16th IEEE Real-Time Systems Symposium, Pisa, Italy, December 1995.

Third ACM International Conference on Multimedia, Technical Paper Program, San Francisco, CA, November 1995.

Third ACM International Conference on Multimedia, Video Program, San Francisco, CA, November 1995.

1995 ACM Conference on Organizational Computing Systems, Technical Track, Milpitas, CA, August 1995.

Fifth IEEE Workshop on Hot Topics in Operating Systems, Orcas Island, WA, May 1995.

15th International Conference on Distributed Computing Systems, Distributed Real-Time Systems Track and CSCW track, Vancouver, British Columbia, Canada, May 1995.

Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, NH, April 1995.

IEEE Workshop on the Role of Real-Time in Multimedia, Research Triangle Park, NC, December 1993.

14th IEEE Real-Time Systems Symposium, Research Triangle Park, NC, December 1993.

13th International Conference on Distributed Computing Systems, Real-Time Issues Track, Pittsburgh, PA, May 1993.

12th IEEE Real-Time Systems Symposium, San Antonio, TX, December 1991.

IEEE Conference on Communication Software: Communications for Distributed Applications and Systems, Chapel Hill, NC, April 1991.

Conference Organizing Committees

Finance Chair, IEEE Real-Time Systems Symposium, Miami, FL, December 2005.

Finance Chair, IEEE Real-Time Systems Symposium, Lisbon, Portugal, December 2004.

Finance Chair, IEEE Real-Time Systems Symposium, Cancun, Mexico, December 2003.

Poster Session Chair, ACM Symposium on Operating System Principles, Bolton Landing, NY, October 2003.

Demonstrations Chair, ACM Conference on Computer-Supported Cooperative Work, Research Triangle Park, NC, November 1994.

Wine Steward, ACM Symposium on Operating System Principles, Asheville, NC, December 1993.

Local Arrangements Chair, 14th IEEE Symposium on Real-Time Systems, Durham, NC, December 1993.

Refereed Publications

Technical Papers

Generating Realistic Synthetic TCP Application Workloads, J. Aikat, K. Jeffay, F.D. Smith, Proceedings, GENI Infrastructure Workshop, Princeton, NJ, June 2010.

Correlations of Size, Rate, and Duration in TCP Connections: The Case Against, C. Park, F. Hernández-Campos, J.S. Marron, K. Jeffay, F.D. Smith, Annals of Applied Statistics, Volume 4, Number 1, May 2010, pages 26-52.

Passive, Streaming Inference of TCP Connection Structure for Network Server Management, J. Terrell, K. Jeffay, F.D. Smith, J. Gogan, J. Keller, IFIP International Workshop on Traffic Monitoring and Analysis, Aachen, Germany, May 2009, in, Lecture Notes in Computer Science, Volume 5537, Springer, Berlin, Germany, pages 42-53.

Exposing Server Performance to Network Managers Through Passive Network Measurements, J. Terrell, K. Jeffay, F.D. Smith, J. Gogan, J. Keller, IEEE Internet Network Management Workshop 2008, Orlando, FL, October 2008, pages 1-6.

Multi-Resolution Anomaly Detection for the Internet, L. Zhang, Z. Zhu, K. Jeffay, J.S. Marron, F.D. Smith, IEEE Workshop on Network Management, Phoenix, AZ, April 2008, 6 pages.

The Effects of Active Queue Management and Explicit Congestion Notification on Web Performance, L. Le, J. Aikat, K. Jeffay, F.D. Smith, IEEE/ACM Transactions on Networking, Volume 15, Number 6, December 2007, pages 1217-1230.

Co-Scheduling Variable Execution Time Requirement Real-time Tasks and Non Real-Time Tasks, A. Singh, K. Jeffay, Proceedings of the 19th Euromicro Conference on Real-Time Systems, Pisa, Italy, July 2007, pages 191-200.

Quantifying the Effects of Recent Protocol Improvements to Standards-Track TCP: Impact on Web Performance, M.C. Weigle, K. Jeffay, F.D. Smith, Computer Communications, Volume 29, Number 15, September 2006, pages 2853-2866.

Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2, M.C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, F.D. Smith, ACM Computer Communications Review, Volume 36, Number 3, July 2006, pages 67-76.

A Loss and Queuing-Delay Controller for Router Buffer Management, L. Le, K. Jeffay, F.D. Smith, Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, Lisbon, Portugal, July 2006, 10 pages.

Understanding Patterns of TCP Connection Usage with Statistical Clustering, F. Hernández-Campos, A.B. Nobel, F.D. Smith, K. Jeffay, 13th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Atlanta, GA, September 2005, pages 35-44.

Delay-Based Early Congestion Detection and Adaptation: Impact on Web Performance, M.C. Weigle, K. Jeffay, F.D. Smith, Computer Communications, Volume 8, Number 8, May 2005, pages 837-850.

Extremal Dependence: Internet Traffic Applications, F. Hernández-Campos, K. Jeffay, C. Park, J.S. Marron, S.I. Resnick, Stochastic Models, Volume 21, Number 1, 2005, pages 1-35.

Generating Realistic TCP Workloads, F. Hernández-Campos, F.D. Smith, K. Jeffay, Proceedings of the Computer Measurement Group's 2004 International Conference, Las Vegas, NV, December 2004, pages 273-284.

Differential Congestion Notification: Taming the Elephants, L. Le, J. Aikat, K. Jeffay, F.D. Smith, Proceedings of the 12th IEEE International Conference on Network Protocols, Berlin, Germany, October 2004, pages 118-128.

Stochastic Models for Generating Synthetic HTTP Source Traffic, J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith M.C. Weigle, Proceedings of IEEE INFOCOM 2004, Hong Kong, March 2004, Volume 3, pages 1546-1557.

Variability in TCP Roundtrip Times, J. Aikat, J. Kaur, D. Smith, K. Jeffay, Proceedings of the 2003 ACM SIGCOMM Internet Measurement Conference, Miami Beach, FL, October 2003, pages 279-284.

Tracking the Evolution of Web Traffic: 1995-2003, F. Hernández-Campos, K. Jeffay, F.D. Smith, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Orlando, FL, October 2003, pages 16-25.

The Effects of Active Queue Management on Web Performance, L. Le, J. Aikat, K. Jeffay, F.D. Smith, Proceedings of ACM SIGCOMM 2003, Karlsruhe, Germany, August 2003, pages 265-276.

Managing Latency and Buffer Requirements in Processing Graph Chains, S.M. Goddard, K. Jeffay, The Computer Journal, Volume 44, Number 6, 2001, (special issue on high-assurance systems), pages 486-

503.

Rate-Based Resource Allocation Methods for Embedded Systems, K. Jeffay, S.M. Goddard, in, *Embedded Software*, Proceedings of the First International Workshop on Embedded Software (*EMSOFT 2001*), Tahoe City, CA, October 2001, Lecture Notes in Computer Science, Volume 2211, T. Henzinger, C. Kirsch, editors, Springer-Verlag, Berlin, Germany, 2001, pages 204-222.

Beyond Audio and Video: Multimedia Networking Support for Distributed, Immersive Virtual Environments, K. Jeffay, T. Hudson, M. Parris, Proceedings of the 27th EUROMICRO Conference, Warsaw, Poland, September 2001, pages 300-307.

Tuning RED for Web Traffic, M. Christiansen, K. Jeffay, D. Ott, F.D. Smith, IEEE/ACM Transactions on Networking, Volume 9, Number 3, (June 2001), pages 249-264.

What TCP/IP Protocol Headers Can Tell Us About the Web, F.D. Smith, F. Hernández Campos, K. Jeffay, D. Ott, Proceedings of ACM SIGMETRICS 2001/Performance 2001, Cambridge, MA, June 2001, pages 245-256.

Experiments in Best-Effort Multimedia Networking for a Distributed Virtual Environment, T. Hudson, M.C. Weigle, K. Jeffay, R.M. Taylor II, in *Multimedia Computing and Networking 2001*, Proceedings, SPIE Proceedings Series, Volume 4312, San Jose, CA, January 2001, pages 88-98.

Analyzing the Real-Time Properties of a U.S. Navy Signal Processing System, S.M. Goddard, K. Jeffay, International Journal of Reliability, Quality and Safety Engineering, Volume 8, Number 4, December 2001, (special issue of HASE '99 best papers) pages 301-322.

A Comparative Study of the Realization of Rate-Based Computing Services in General Purpose Operating Systems, K. Jeffay, G. Lamastra, Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications, Cheju Island, South Korea, December 2000, pages 81-90.

The Synthesis of Real–Time Systems from Processing Graphs, S.M. Goddard, K. Jeffay, Proceedings of the Fifth IEEE International Symposium on High Assurance Systems Engineering, Albuquerque, NM, November 2000, pages 177-186.

Tuning RED for Web Traffic, M. Christiansen, K. Jeffay, D. Ott, F.D. Smith, Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August-September 2000, pages 139-150.

Towards a Better-Than-Best-Effort Forwarding Service for Multimedia Flows, K. Jeffay, IEEE Multimedia, Volume 6, Number 4, October-December 1999, pages 84-88.

A Theory of Rate-Based Execution, K. Jeffay, S.M. Goddard, Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1999, pages 304-314.

Parallel Switching in Connection-Oriented Networks, S. Baruah, K. Jeffay, J. Anderson, Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1999, pages 200-209.

Analyzing the Real-Time Properties of a U.S. Navy Signal Processing System, S.M. Goddard, K. Jeffay, Proceedings of the Fourth IEEE International Symposium on High Assurance Systems Engineering, Washington, DC, November 1999, pages 141-150.

Application-Level Measurements of Performance on the vBNS, M. Clark, K. Jeffay, Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Volume 2, Florence, Italy, June 1999, pages 362-366.

Lightweight Active Router-Queue Management for Multimedia Networking, M. Parris, K. Jeffay, F.D. Smith, in *Multimedia Computing and Networking 1999*, Proceedings, SPIE Proceedings Series, Volume 3654, San Jose, CA, January 1999, pages 162-174.

Proportional Share Scheduling of Operating System Services for Real-Time Applications, K. Jeffay, F.D. Smith, A. Moorthy, J.H. Anderson, Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998, pages 480-491.

Efficient Object Sharing in Quantum-Based Real-Time Systems, J.H. Anderson, R. Jain, K. Jeffay, Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998, pages 346-355.

A Better-Than-Best-Effort Service for Continuous Media UDP Flows, M. Parris, K. Jeffay, F.D. Smith, J. Borgersen, Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video, Cambridge, UK, July 1998, pages 193-197.

Managing Memory Requirements in the Synthesis of Real-Time Systems from Processing Graphs, S. Goddard, K. Jeffay, Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium, Denver, CO, June 1998, pages 59-70.

Fair On-Line Scheduling of a Dynamic Set of Tasks on a Single Resource, S.K. Baruah, J.E. Gehrke, C.G. Plaxton, I. Stoica, H. Abdel-Wahab, K. Jeffay, *Information Processing Letters*, Volume 64, Number 1, October 1997, pages 43-51.

A Two-Dimensional Audio Scaling Enhancement to an Internet Videoconferencing System, P. Nee, K. Jeffay, M. Clark, G. Danneels, Proceedings of the International Workshop on Audio-Visual Services over Packet Networks, Aberdeen, Scotland, UK, September 1997, pages 201-206.

Feasibility Concerns in PGM Graphs With Bounded Buffers, S. Baruah, S. Goddard, K. Jeffay, Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems, Como, Italy, September 1997, pages 130-139.

Analyzing the Real-Time Properties of a Dataflow Execution Paradigm Using a Synthetic Aperture Radar Application, S. Goddard, K. Jeffay, Proceedings of the Third IEEE Real-Time Technology and Applications Symposium, Montreal, Canada, June 1997, pages 60-71.

Real-Time Computing with Lock-Free Shared Objects, J.H. Anderson, S. Ramamurthy, K. Jeffay, ACM Transactions on Computing Systems, Volume 15, Number 2, May 1997, pages 134-165.

The Performance of Two-Dimensional Media Scaling for Internet Videoconferencing, P. Nee, K. Jeffay, G. Danneels, Proceedings of the Seventh International Workshop on Network and Operating System Support for Digital Audio and Video, St. Louis, MO, May 1997, pages 237-248. Republished in "Readings in Multimedia Computing," K. Jeffay, H.J. Zhang, editors, Morgan Kaufmann, 2002.

On the Duality between Resource Reservation and Proportional Share Resource Allocation, I. Stoica, H. Abdel-Wahab, K. Jeffay, in *Multimedia Computing and Networking 1997*, Proceedings, SPIE Proceedings Series, Volume 3020, San Jose, CA, February 1997, pages 207-214.

Strategic Directions in Real-Time and Embedded Systems, J.A. Stankovic, A. Burns, K. Jeffay, M. Jones, G. Koob, I. Lee, J. Lehoczky, J. Liu, A. Mok, K. Ramamritham, J. Ready, L. Sha, A. van Tilborg, *ACM Computing Surveys*, Volume 28, Number 4, December 1996, pages 751-763.

A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems, I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, C.G. Plaxton, Proceedings of the 17th IEEE Real-Time Systems Symposium, Washington, DC, December 1996, pages 288-299.

A General Framework For Continuous Media Transmission Control, T.M. Talley, K. Jeffay, Proceedings of the 21st IEEE Conference on Local Computer Networks, Minneapolis, MN, October 1996, pages 374-383.

A Router-Based Congestion Control Scheme For Real-Time Continuous Media, K. Jeffay, M. Parris, T. Talley, F.D. Smith, Proceedings of the Sixth International Workshop on Network and Operating System

Support for Digital Audio and Video, Zushi, Japan, April 1996, pages 79-86.

Lock-Free Transactions for Real-time Systems, J.H. Anderson, S. Ramamurthy, M. Moir, K. Jeffay, Proceedings of the First Workshop on Real-Time Databases: Issues and Applications, Newport Beach, CA, March 1996, pages 107-114.

Real-Time Computing with Lock-Free Shared Objects, J.H. Anderson, S. Ramamurthy, K. Jeffay, Proceedings of the 16th IEEE Real-Time Systems Symposium, Pisa, Italy, December 1995, pages 28-37.

Early Experience with the Repository for Patterned Injury Data, D. Stotts, J.B. Smith, K. Jeffay, P. Dewan, D.K. Smith, W. Oliver, Proceedings of the SPIE International Symposium on Investigative and Trial Image Processing, San Diego, CA, July 1995, SPIE Volume 2567, 1995, pages 249-260.

Early Prototypes of the Repository for Patterned Injury Data, P. Dewan, K. Jeffay, J. Smith, D. Stotts, W. Oliver, Proceedings of Digital Libraries '95, The Second Annual Conference on the Theory and Practice of Digital Libraries, Austin, TX, June 1995, pages 123-130.

Support For Real-Time Computing Within General Purpose Operating Systems: Supporting co-resident operating systems, G. Bollella, K. Jeffay, Proceedings of the IEEE Real-Time Technology and Applications Symposium, Chicago, IL, May 1995, pages 4-14.

A Rate-Based Execution Abstraction For Multimedia Computing, K. Jeffay, D. Bennett, Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, NH, April 1995, published in *Lecture Notes in Computer Science*, T.D.C. Little, R. Gusella, editors, Volume 1018, pages 64-75, Springer-Verlag, Heidelberg, Germany, 1995.

An Empirical Study of Delay Jitter Management Policies, D.L. Stone, K. Jeffay, *ACM Multimedia Systems*, Volume 2, Number 6, January 1995, pages 267-279. Republished in "Readings in Multimedia Computing," K. Jeffay, H.J. Zhang, editors, Morgan Kaufmann, 2002.

On the Partitioning of Function in Distributed Synchronous Collaboration Systems, J. Menges, K. Jeffay, ACM CSCW '94, Proceedings of the Workshop on Distributed Systems, Multimedia, and Infrastructure Support in CSCW, Research Triangle Park, NC, October 1994, SIGOIS Bulletin, Volume 15, Number 2, December 1994, pages 34-37.

Two-Dimensional Scaling Techniques For Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams, T.M. Talley, K. Jeffay, Proceedings of the Second ACM International Conference on Multimedia, San Francisco, CA, October 1994, pages 247-254.

Transport and Display Mechanisms For Multimedia Conferencing Across Packet-Switched Networks, K. Jeffay, D.L. Stone, F.D. Smith, *Computer Networks and ISDN Systems*, Volume 26, Number 10, July 1994, pages 1281-1304. Republished in "A Guided Tour of Multimedia Systems and Applications," B. Furht, M. Milenkovic, editors, IEEE Computer Society Press, 1995.

A Patterned Injury Digital Library for Collaborative Forensic Medicine, D. Stotts, J.B. Smith, P. Dewan, K. Jeffay, F.D. Smith, D. Smith, S. Weiss, J. Coggins, W. Oliver, Proceedings of Digital Libraries '94, The First Annual Conference on the Theory and Practice of Digital Libraries, College Station, TX, June 1994, pages 25-33.

The Artifact-Based Collaboration System: An infrastructure for supporting and studying collaboration, K. Jeffay, J.B. Smith, F.D. Smith, D.E. Shackelford, J. Menges, Proceedings of the 15th Interdisciplinary Workshop on Informatics and Psychology, Schärding, Austria, May 1994, 27 pages.

On Latency Management in Time-Shared Operating Systems, K. Jeffay, Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software, Seattle, WA, May 1994, pages 86-90.

Inverting X: An Architecture for a Shared Distributed Window System, J. Menges, K. Jeffay, Proceedings of the Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises,

Morgantown, WV, April 1994, IEEE Computer Society Press, pages 53-64.

Issues, Problems, and Solutions in Sharing X Clients on Multiple Displays, H. Abdel-Wahab, K. Jeffay, *Internetworking — Research and Practice*, Volume 5, Number 1, March 1994, pages 1-15.

Dynamic Participation in a Computer-based Conferencing System, G. Chung, K. Jeffay, H. Abdel-Wahab, *Computer Communications*, Volume 17, Number 1, January 1994, pages 7-16.

Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems, K. Jeffay, D.L. Stone, Proceedings of the 14th IEEE Real-Time Systems Symposium, Raleigh-Durham, NC, December 1993, pages 212-221.

Queue Monitoring: A Delay Jitter Management Policy, D.L. Stone, K. Jeffay, Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, UK, November 1993, published in *Lecture Notes in Computer Science*, D. Shepherd, G. Blair, G. Coulson, N. Davies, F. Garcia, editors, Volume 846, pages 149-160, Springer-Verlag, Heidelberg, Germany, 1994.

The Real-Time Producer/Consumer Paradigm: A paradigm for the construction of efficient, predictable real-time systems, K. Jeffay, Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing, Indianapolis, IN, ACM Press, February 1993, pages 796-804.

Accommodating Late-Comers in Shared Window Systems, G. Chung, K. Jeffay, H. Abdel-Wahab, *IEEE Computer*, Volume 26, Number 1, January 1993, pages 72-74.

Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems, K. Jeffay, Proceedings of the 13th IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1992, pages 89-99.

Adaptive, Best-Effort, Delivery of Audio and Video Data Across Packet-Switched Networks, K. Jeffay, D.L. Stone, T. Talley, F.D. Smith, Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, La Jolla, CA, November 1992, published in *Lecture Notes in Computer Science*, V. Rangan, editor, Volume 712, pages 3-14, Springer-Verlag, Heidelberg, Germany, 1993.

Architecture of the Artifact-Based Collaboration System Matrix, K. Jeffay, J.K. Lin, J. Menges, F.D. Smith, J.B. Smith, ACM CSCW '92, Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada, ACM Press, November 1992, pages 195-202.

Client/Server Protocol Filtering: An infrastructure for supporting group collaborations, K. Jeffay, J. Menges, J.-K. Lin, ACM CSCW '92, Proceedings of the Workshop on Tools and Technologies, Toronto, Canada, November 1992, pages 96-99.

Kernel Support for Live Digital Audio and Video, K. Jeffay, D.L. Stone, F.D. Smith, *Computer Communications*, Volume 15, Number 6, July/August 1992, pages 388-395.

On Kernel Support for Real-Time Multimedia Applications, K. Jeffay, Proceedings of the Third IEEE Workshop on Workstation Operating Systems, Key Biscayne, FL, April 1992, pages 39-46.

On Non-Preemptive Scheduling of Periodic and Sporadic Tasks, K. Jeffay, D.F. Stanat, C.U. Martel, Proceedings of the Twelfth IEEE Real-Time Systems Symposium, San Antonio, TX, December 1991, pages 129-139.

Kernel Support for Live Digital Audio and Video, K. Jeffay, D.L. Stone, F.D. Smith, Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991, published in *Lecture Notes in Computer Science*, R.G. Herrtwich, editor, Volume 614, pages 10-21, Springer-Verlag, Heidelberg, Germany, 1992.

UNC Collaboratory Project Overview, J.B. Smith, F.D. Smith, P. Calingaert, J.R. Hayes, D. Holland, K. Jeffay, M. Lansman, Proceedings of the 1991 Symposium on Command and Control Research, National

Defense University, Washington, D.C., June 1991, pages 341-391.

YARTOS: Kernel support for efficient, predictable real-time systems, K. Jeffay, D. Stone, D. Poirier, Proceedings of the Joint Eighth IEEE Workshop on Real-Time Operating Systems and Software and IFAC/IFIP Workshop on Real-Time Programming, Atlanta, GA, May 1991, in Real-Time Systems Newsletter, Volume 7, Number 4, Fall 1991, pages 8-13. Republished in “Real-Time Programming,” W. Halang, K. Ramamritham, editors, 1992.

System Design for Workstation-Based Conferencing With Digital Audio and Video, K. Jeffay, F.D. Smith, Proceedings of the IEEE Conference on Communication Software: Communications for Distributed Applications and Systems, Chapel Hill, NC, April 1991, pages 169-180.

Designing a Workstation-Based Conferencing System Using the Real-Time Producer/Consumer Paradigm, K. Jeffay, F.D. Smith, Proceedings of the First International Workshop on Network and Operating System Support for Digital Audio and Video, International Computer Science Institute, Berkeley, CA, November 1990, pages 40-55.

Analysis of a Synchronization and Scheduling Discipline for Real-Time Tasks with Preemption Constraints, K. Jeffay, Proceedings of the Tenth IEEE Real-Time Systems Symposium, Santa Monica, CA, December 1989, pages 295-305.

Corset & Lace: Adapting Ada Runtime Support to Real-Time Systems, T.P. Baker, K. Jeffay, Proceedings of the Eighth IEEE Real-Time Systems Symposium, San Jose, CA, December 1987, pages 158-167.

Research in Real-Time Systems, A.C. Shaw, C. Binding, W.L. Hu, K. Jeffay, Proceedings of the Third IEEE Workshop on Real-Time Operating Systems, Boston, MA, February 1986, pages 121-131.

Book Chapters

Rate-Based Resource Allocation Methods, K. Jeffay, in, “Handbook of Real-Time and Embedded Systems,” I. Lee, J. Y-T. Leung, S.H. Son, editors, Chapman & Hall/CRC Press, Boca Raton, FL, 2008, pages 4-1 – 4-15. .

Visualization and Natural Control Systems for Microscopy, R.M. Taylor II, D. Borland, F.P. Brooks Jr., M. Falvo, M. Guthold, T. Hudson, K. Jeffay, G. Jones, D. Marshburn, S.J. Papadakis, L.-C. Qin, A. Seeger, F.D. Smith, D.H. Sonnenwald, R. Superfine, S. Washburn, C. Weigle, M.C. Whitton, P. Williams, L. Vicci, W. Robinett, in “Visualization Handbook,” C. Johnson, C. Hansen, editors, Harcourt Academic Press, 2005, pages 875-900.

An Empirical Study of Delay Jitter Management Policies, D.L. Stone, K. Jeffay, in “Readings in Multimedia Computing and Networking,” K. Jeffay, H.J. Zhang, editors, Morgan Kaufmann, San Francisco, CA, 2002, pages 525-537.

The Performance of Two-Dimensional Media Scaling for Internet Videoconferencing, P. Nee, K. Jeffay, G. Danneels, in “Readings in Multimedia Computing and Networking,” K. Jeffay, H.J. Zhang, editors, Morgan Kaufmann, San Francisco, CA, 2002, pages 581-592.

Lock-Free Transactions for Real-time Systems, J.H. Anderson, S. Ramamurthy, M. Moir, K. Jeffay, in “Real-Time Database Systems: Issues and Applications,” A. Bestavros, K.J. Lin, S.H. Son, editors, Kluwer Academic Publishers, Norwell, MA, 1997, pages 215-234.

Transport and Display Mechanisms For Multimedia Conferencing Across Packet-Switched Networks, K. Jeffay, D.L. Stone, F.D. Smith, in “A Guided Tour of Multimedia Systems and Applications,” B. Furht, M. Milenkovic, editors, IEEE Computer Society Press, Los Alamitos, CA, 1995, pages 439-461.

Contributor to: *R&D for the NII: Technical Challenges*, M.K. Vernon, E.D. Lazowska, S.D. Personick, editors, Interuniveristy Communications Council (EDUCOM), 1994.

YARTOS: Kernel support for efficient, predictable real-time systems, K. Jeffay, D. Stone, D. Poirier, in “Real-Time Programming,” W. Halang, K. Ramamritham, editors, Pergamon Press, Oxford, UK, 1992,

pages 7-12.

Videotapes

Adaptive, Best-Effort Delivery of Live Audio and Video Across Packet-Switched Networks, K. Jeffay, D.L. Stone, ACM Multimedia '94 Video Proceedings, San Francisco, CA, October 1994, 6 minutes. Excerpts also appear on the CD-ROM version of the conference proceedings.

Abstracts & Short Papers

Quantifying the Effects of Recent Protocol Improvements to Standards-Track TCP, M.C. Weigle, K. Jeffay, and F.D. Smith, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Orlando, FL, October 2003, pages 226-229.

Engineering of Rate Based Services for Real-Time Computing, G. Lamastra, K. Jeffay, 20th IEEE Real-Time Systems Symposium Work in Progress Proceedings, Phoenix, AZ, December 1999, pages 51-55.

Lightweight Active Router-Queue Management for Multimedia Networking, M. Parris, K. Jeffay, F.D. Smith, IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, San Jose, CA, January 1999, pages 280-281.

Support For Real-Time Computing in Windows NT, K. Jeffay, Usage Abstracts, USENIX Windows NT Workshop, Seattle, WA, August 1997, page 84.

On the Duality between Resource Reservation and Proportional Share Resource Allocation, I. Stoica, H. Abdel-Wahab, K. Jeffay, IS&T/SPIE Ninth Symposium on Electronic Imaging: Science and Technology 1997, San Jose, CA, February 1997, pages 135-136.

Technical and Educational Challenges For Real-Time Computing, K. Jeffay, *ACM Computing Surveys*, Volume 28A, Number 4(es), December 1996, URL *http://www.acm.org/pubs/contents/journals/surveys/1996-28/#4es*, 3 pages.

Distributed Real-Time Dataflow: An Execution Paradigm for Image Processing and Anti-Submarine Warfare Applications, S. Goddard, K. Jeffay, 19th IEEE Real-Time Systems Symposium Work in Progress Proceedings, Washington, DC, December 1996, pages 55-58.

Why Using the Request Abstraction in Proportional Share Allocation Systems is Useful, I. Stoica, H. Zhang, K. Jeffay, Proceedings of the IEEE Real-Time Systems Symposium Workshop on Resource Allocation Problems in Multimedia Systems, Washington, DC, December 1996, 4 pages.

Future distributed embedded and real-time applications will be adaptive: Meanings, challenges, and research paradigms, A.K. Mok, C.L. Heitmeyer, K. Jeffay, M.B. Jones, C.D. Locke, R. Rajkumar, 15th Proceedings of the 15th IEEE International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada, May 1995, pages 182-184.

Adaptive, Best-Effort Delivery of Live Audio and Video Across Packet-Switched Networks, K. Jeffay, D.L. Stone, Proceedings of the Second ACM International Conference on Multimedia, San Francisco, CA, October 1994, pages 487-488.

Adaptive Rate-Based Flow and Latency Management of Audio and Video Streams, T.-M. Chen, T. Talley, K. Jeffay, Abstracts of the Fourth Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, UK, November 1993, pages 17-20.

Client/Server Protocol Filtering: An infrastructure for supporting group collaborations, K. Jeffay, J. Menges, J.-K. Lin, ACM CSCW '92, Proceedings of the Workshop on Tools and Technologies, Toronto, Canada, November 1992, pages 96-99.

Network and Operating Systems Support for Digital Audio and Video, K. Jeffay, in 13th ACM Symposium on Operating System Principles "Work in Progress" Abstracts, E.D. Lazowska, editor, Operating Systems Review, Volume 26, Number 2, April 1992, pages 7-31.

Posters

Multivariate SVD Analyses For Network Anomaly Detection, J. Terrell, L. Zhang, K. Jeffay, A. Nobel, H.

Shen, F.D. Smith, Z, Zhu, ACM SIGCOMM 2005 Poster Session, Philadelphia, PA, August 2005.

How Real Can Synthetic Traffic Be?, F. Hernández-Campos, K. Jeffay, F.D. Smith, ACM SIGCOMM 2004 Poster Session, Portland, OR, August 2004.

A Non-Parametric Approach to Generation and Validation of Synthetic Network Traffic, F. Hernández-Campos, A. Nobel, F.D. Smith, K. Jeffay, IMA Workshop on Measurement, Modeling, and Analysis of the Internet, Minneapolis, MN, January 2004.

Sync-TCP: Using GPS Synchronized Clocks for Early Congestion Detection in TCP, M.C. Weigle, K. Jeffay, F.D. Smith, ACM SIGCOMM 2000 Poster Session, Stockholm, Sweden, August 2000, page 6.

Reviews

Advance Reservation Systems, K. Jeffay, Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, NH, April 1995, published in *Lecture Notes in Computer Science*, T.D.C. Little, R. Gusella, editors, Volume 1018, pages 1-2, Springer-Verlag, Heidelberg, Germany, 1995.

Books and Proceedings

The Effects of Active Queue Management on TCP Application Performance, An experimental performance evaluation, L. Le, K. Jeffay, F.D. Smith, Lambert Academic Publishing, Berlin, Saarbrücken, Germany, 2009, 222 pages.

Quality-of-Service — IWQoS 2003, K. Jeffay, I. Stoica, K. Wehrle, editors, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, Germany, Volume 2707, 2003, ISBN 3-540-40281-0, 517 pages.

Readings in Multimedia Computing and Networking, K. Jeffay, H.J. Zhang, editors, Morgan Kaufman, San Francisco, CA, 2002, ISBN 1-55860-651-3, 863 pages.

Proceedings, The 22nd IEEE Real-Time Systems Symposium, K. Jeffay, G. Buttazzo, editors, IEEE Computer Society Press, Los Alamitos, CA, 2001, ISBN 0-7695-1420-0, 321 pages.

Proceedings, The Sixth IEEE International Symposium on Computers and Communications, K. Jeffay, R. Steinmetz, editors, IEEE Computer Society Press, Los Alamitos, CA, 2001, ISBN 0-7695-1177-5, 754 pages.

Proceedings, The 21st IEEE Real-Time Systems Symposium, K. Jeffay, W. Zhao, editors, IEEE Computer Society Press, Los Alamitos, CA, 2000, ISBN 0-7695-0900-2, 311 pages.

Proceedings, The 10th International Workshop on Network and Operating System Support for Digital Audio and Video, K. Jeffay and H. Vin, editors, Technical Report, Department of Computer Science, University of North Carolina at Chapel Hill, 2000, 320 pages. Proceedings also published on-line at *http: //www.cs.unc.edu/nossdav2000*.

ACM Multimedia '99, D. Bulterman, K. Jeffay, H.J. Zhang, editors, ACM Press, Los Angeles, CA, 1999, ISBN 0-8194-3125-7, 500 pages.

Multimedia Computing and Networking 1999, D.D. Kandlur, K. Jeffay, T. Roscoe, editors, Proceedings of SPIE, Volume 3654, SPIE, Bellingham, WA, 1998, ISBN 0-8194-3125-7, 328 pages.

Multimedia Computing and Networking 1998, K. Jeffay, D.D. Kandlur, T. Roscoe, editors, Proceedings of SPIE, Volume 3310, SPIE, Bellingham, WA, 1997, ISBN 0-8194-250-0, 266 pages.

Proceedings, 1997 IEEE Real-Time Technology and Applications Symposium, R. Rajkumar, K. Jeffay, editors, IEEE Computer Society Press, Los Alamitos, CA, 1997, ISBN 0-8186-8016-4, 269 pages.

Proceedings, IEEE Real-Time Systems Symposium Workshop on Resource Allocation Problems in Multimedia Systems, K. Jeffay, editor, *http://www.cs.unc.edu/~jeffay/meetings/mm-wrkshp96/prog. html,*

1996, 250 pages.

Proceedings, 1996 IEEE Real-Time Technology and Applications Symposium, K. Jeffay, W. Zhao, editors, IEEE Computer Society Press, Los Alamitos, CA, 1996, ISBN 0-8186-7448-2, 264 pages.

Unrefereed Publications

Invited Papers

Modeling and Generation of TCP Application Workloads, F. Hernández-Campos, K. Jeffay, F.D. Smith, Proceedings of the 4th IEEE International Conference on Broadband Communications, Networks, and Systems, Raleigh, NC, September 2007, 10 pages.

Statistical Clustering of Internet Communication Patterns, F. Hernández-Campos, A.B. Nobel, F.D. Smith, K. Jeffay, Proceedings of the 35th Symposium on the Interface of Computing Science and Statistics, Salt Lake City, UT, July 2003, Computing Science and Statistics, Volume 35, 2004.

Rate-Based Resource Allocation Methods for Embedded Systems, K. Jeffay, S.M. Goddard, in, *Embedded Software*, Proceedings of the First International Workshop on Embedded Software (*EMSOFT 2001*), Tahoe City, CA, October 2001, Lecture Notes in Computer Science, Volume 2211, T. Henzinger, C. Kirsch, editors, Springer Verlag, Berlin, Germany, 2001, pages 204-222.

Beyond Audio and Video: Multimedia Networking Support for Distributed, Immersive Virtual Environments, K. Jeffay, T. Hudson, M. Parris, Proceedings of the 27th EUROMICRO Conference, Workshop on Multimedia and Telecommunication, Warsaw, Poland, September 2001, pages 300-307.

Experiments in Best-Effort Multimedia Networking for a Distributed Virtual Environment, T. Hudson, M.C. Weigle, K. Jeffay, R.M. Taylor II, in *Multimedia Computing and Networking 2001*, Proceedings, SPIE Proceedings Series, Volume 4312, San Jose, CA, January 2001, pages 88-98.

Towards a Better-Than-Best-Effort Forwarding Service for Multimedia Flows, K. Jeffay, IEEE Multimedia, Volume 6, Number 4, October-December 1999, pages 84-88.

Network Support For Distributed, Immersive, Virtual Laboratories K. Jeffay, Proceedings of the NSF Workshop on Automated Control of Distributed Instrumentation, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL, April 1999, pages 73-78.

Storage Requirements For Distributed Virtual Laboratories, K. Jeffay, Proceedings of the Internet2 Distributed Storage Infrastructure Application Workshop, University of North Carolina at Chapel Hill, Chapel Hill, NC, March 1999, pages 57-59.

Efficient Kernel Support for Continuous Time Media Systems, K. Jeffay, Proceedings of the ITC Workshop on Continuous Time Media, Carnegie Mellon University, Pittsburgh, PA, June 1991, 4 pages.

White Papers

Contributor to: *NSF Report on Network Research Testbeds*, B. Braden, M. Gerla, J. Kurose, J. Lepreau, R. Rao, J. Turner, editors, December 2002. A white paper commission by NSF that led to the creation of a $10 million funding program for network research testbeds.

Contributor to: *R&D for the National Information Infrastructure: Technical Challenges*, M.K. Vernon, E.D. Lazowska, S.D. Personick, editors, Interuniveristy Communications Council (EDUCOM), 1994.

Technical Reports

Beyond Window Sharing Hacks: Support for First-Class Window Sharing, J. Menges, K. Jeffay, Report TR97-021, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, April 1997.

A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems, I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, C.G. Plaxton, Report TR96-038, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, September 1996.

Predicting Worst Case Execution Times on a Pipelined RISC Processor, S.J. Bharrat, K. Jeffay, Report TR94-072, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, April 1994.

The Design, Implementation, and Use of a Sporadic Tasking Model, K. Jeffay, D. Becker, D. Bennett, S. Bharrat, T. Gramling, M. Housel, Report TR94-073, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, April, 1994.

UNC Collaboratory Project Overview, J.B. Smith, F.D. Smith, P. Calingaert, J.R. Hayes, D. Holland, K. Jeffay, M. Lansman, Report 90-042, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, November 1990.

An Implementation and Application of the Real-Time Producer/Consumer Paradigm, D. Poirier, K. Jeffay, Report 90-038, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, October 1990.

The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations, K. Jeffay, Report 89-09-15, University of Washington, Department of Computer Science, Seattle, WA, September 1989, (Ph.D. Dissertation).

On Optimal, Non-Preemptive Scheduling of Periodic and Sporadic Tasks, K. Jeffay, R. Anderson, Report 88-11-06, University of Washington, Department of Computer Science, Seattle, WA, November 1988.

On Optimal, Non-Preemptive Scheduling of Periodic Tasks, K. Jeffay, Report 88-10-03, University of Washington Department of Computer Science, Seattle, WA, October 1988.

Software Engineering of Real-Time Operating Systems, A.C. Shaw, K. Jeffay, Report 88-01-01, University of Washington Department of Computer Science, Seattle, WA, January 1988.

Concurrent Programming with Time, A thesis proposal, K. Jeffay, Report 87-10-03, University of Washington Department of Computer Science, Seattle, WA, October 1987.

A Lace for Ada's Corset, T.P. Baker, K. Jeffay, Report 86-09-06, University of Washington Department of Computer Science, Seattle, WA, September 1986.

Software Distributions

tmix — A synthetic TCP workload generator, F. Hernández Campos, K. Jeffay, F.D. Smith, June 2005.

Distributed to other academic and industry networking research groups via the Web.

UNC VNC — A version of the public domain workspace sharing system VNC to support secure sharing of workspace regions and windows of individual applications. J. Branscomb, L. Fowler, K. Jeffay, August 2004.

Distributed to other research groups and the public via the Web and SourceForge.

thttp 2003 — An HTTP/v1.0 and v1.1 traffic generation program used to simulate the traffic generated by a collection of geographically distributed web browsers and servers. L. Le, A. Van Osdol, F.D. Smith, K. Jeffay, January 2004.

Distributed to other academic and industry networking research groups via the Web.

thttp 2001 — An HTTP/v1.0 and v1.1 traffic generation program used to simulate the traffic generated by a collection of geographically distributed web browsers and servers. F. Hernández Campos, F.D. Smith, K. Jeffay, August 2002.

Distributed to other academic and industry networking research groups via the Web.

Packmime Implementation in ns — An implementation of the Lucent Bell Labs (Bill Cleaveland) HTTP traffic model in ns. M.C. Weigle, K. Jeffay, F.D. Smith, September 2001.

Distributed as part of the Network Simulator (**ns-2**) distribution.

UNC Campus Network Trace Data and Empirical Distributions — A collection of network traces and empirical distributions illustrating the nature of web browsing and the structure of web pages. F. Hernández Campos, K. Jeffay, F.D. Smith, June 2001.

Distributed to other academic and industry networking research groups via the Web.

HTTP Analysis Tools— A set of tools for analyzing http connections. F. Hernández Campos, K. Jeffay, F.D. Smith, June 2001.

Distributed as part of the Network Simulator (**ns-2**) distribution.

thttp — An HTTP traffic generation program used to simulate the traffic generated by a collection of geographically distributed web browsers and servers. M. Christiansen, F.D. Smith, K. Jeffay, January 2000.

Distributed to other academic and industry networking research groups via the Web.

FlowGen — A general purpose, programmable network flow generator. M. Clark, K. Jeffay, F.D. Smith, December 1999.

Distributed to Advanced Networking Systems Inc. for use in performance evaluation studies of Internet 2 and Abilene. Also available via anonymous *ftp* from *http://www.cs.unc.edu/Research/Dirt*.

TruTime Device Drivers for FreeBSD — A set of FreeBSD device drivers for the TruTime GPS boards, A. Moorthy, K. Jeffay, and F.D. Smith, December 1998.

Distributed to Advanced Networking Systems Inc. for use in the IETF Internet Performance Measurement Initiative (IPMI) Surveyor network.

YARTOS (*Yet Another Real-Time Operating System*) — A real-time operating system kernel for Intel x86 platforms that uses a novel task implementation based on a single, shared run-time stack, an earliest-deadline-first processor scheduling algorithm developed at UNC, and integrates processor scheduling and inter-process communication. K. Jeffay and D. Stone, January 1996.

Distributed to several universities for use in operating system research and teaching. Also adopted by researchers at IBM's Networking Software Division, RTP, NC, for use in constructing a network protocol evaluation testbed.

XPEL (*The X Protocol Engine Library*) — A C++ library for constructing modular X Window System pseudo-servers, J. Menges and K. Jeffay, November 1993.

Distributed on the Internet via anonymous *ftp* from *ftp.cs.unc.edu*.

XTV (*X Terminal View*) — An X Window System pseudo-server that supports window sharing and conferencing across the Internet, H. Abdel-Wahab (Old Dominion University) and K. Jeffay, August 1991.

Distributed by MIT as part of the contributed software portion of the X Window System distribution (release 11, version 5). Last revised January 1993.

Also adopted by the NEC Corporation of Japan as the basis for their *Xprotodist* (X protocol distributor) product.

Patents

Network Server Performance Anomaly Detection, U.S. Provisional Patent Application No. 61/167,819, J.S. Terrell. K. Jeffay, F.D. Smith, E. Broadhurst, filed April, 2009.

Method for Understanding the Use of TCP/IP Networks by Users, and Non-Parametric Generation of Synthetic Internet Traffic, U.S. Patent Number 7,447,209, F. Hernández-Campos, K. Jeffay, F.D. Smith, A. Nobel, November 2008.

User Controlled Adaptive Flow Control for Packet Networks, U.S. Patent Number 5,892,754, V. Kompella, F.D. Smith, J.P. Gray, and K. Jeffay, April 1999.

Public Demonstrations

The nanoManipulator “reverse field trip” to Orange County High-School, G. Jones, R. Superfine, R.M. Taylor, M. Whitton, T. Lovelace, R. Parameswaran, K. Jeffay, F.D. Smith, Orange County High School, Hillsboro, NC, November 1999.

The Distributed NanoManipulator Project, R.M. Taylor, K. Jeffay, T. Hudson, M. Clark, an invited demonstration for the Internet 2 Spring 1999 Members Meeting, Washington, DC, April 1999.

A Demonstration of Internet Access to UNC-CH Advanced Microscopy Facilities for Science Education Outreach, K. Jeffay, F.D. Smith, R. Superfine, G. Jones, R.M. Taylor, T. Hudson, M. Clark, Orange County High School, Hillsboro, NC, June 1998.

The Artifact-Based Collaboration (ABC) System, J.B. Smith, K. Jeffay, D. Shackelford, J. Menges, J. Hilgedic, B. Ladd, M. Parris, E. Kupstas, T. Ellis, and T. Hudson, CSCW ‘94, ACM Conference on Computer-Supported Cooperative Work, RTP, NC, October 1994.

CONCUR: A Window System Supporting Window Sharing, J. Menges and K. Jeffay, CSCW ‘94, ACM Conference on Computer-Supported Cooperative Work, RTP, NC, October 1994.

Adaptive, Best-Effort Delivery of Live Audio and Video Across Packet-Switched Networks, K. Jeffay, Second MCNC/NSF Packet Video Workshop, Research Triangle Park, NC, December 1992.

Grants and Awards

Active

Exploring Privacy Breaches in Encrypted VoIP Communications, F. Monrose, K, Jeffay, National Science Foundation, grant number CNS 10-17318, August 2010, 3 years, $496,492.

Synthetic Traffic Generation Tools and Resource: A Community Resource for Experimental Networking Research, K, Jeffay, F.D. Smith, UNC, M. Weigle, Old Dominion University, A. Vahdat, University of California San Diego, P. Barford, University of Wisconsin, National Science Foundation, grant number CRI 07-09081, August 2007, 3 years, $799,745.

Completed

Modeling and Testing of Application Workloads on Corporate Enterprise Networks, K, Jeffay, F.D. Smith, IBM Global Services Faculty Award, September 2006, 2 years, $70,000 (exempt from indirect costs).

Tera-pixels: Using High-resolution Pervasive Displays to Transform Collaboration and Teaching, K, Jeffay, A. Lastra, K. Mayer-Patel, L. McMillan, F.D. Smith, National Science Foundation (CISE RI Program), grant number EIA 03-03590, August 2003, 5 years, $962,902.

Extracting and Using Semantic Information in Network Workloads, K, Jeffay, F.D. Smith, IBM Global Services Faculty Award, September 2006, 1 year, $40,000 (exempt from indirect costs).

Modeling and Testing of Application Workloads on Corporate Enterprise Networks, K, Jeffay, F.D. Smith, IBM Global Services Faculty Award, September 2006, 1 year, $40,000 (exempt from indirect costs).

Generation and Validation of Synthetic Internet Traffic, K, Jeffay, F.D. Smith, A.B. Noble, National Science Foundation, grant number ANI 03-23648, September 2003, 3 years, $470,000.

Rate-Based Resource Allocation Methods for Real-Time Embedded Systems, K, Jeffay, F.D. Smith, National Science Foundation, grant number CCR 02-08924, August 2002, 3 years, $179,990.

Empirical Workload Characterizations for Advanced Networks, K. Jeffay, F.D. Smith, Cisco Systems, November 2000, 3 years, $212,500.

Support for Active Queue Management, Distributed XP Programming, and a Teaching Laboratory for

Web Services, F.D. Smith, K. Jeffay, J.B. Smith, P.D. Stotts, IBM Shared University Research Grant, August 2002, 1 year, $83,000.

Internet Traffic Measurement and Analysis and A Teaching Laboratory for Enterprise Java Computing, F.D. Smith, K. Jeffay, J.B. Smith, P.D. Stotts, IBM Shared University Research Grant, April 2001, 1 year, $83,000.

An NS Implementation of the HTTP Connection Model, F.D. Smith, K. Jeffay, M. Clark Weigle, Lucent Technologies, April 2001, $10,000.

Multimedia Networking Research in Support of Virtual Field Trips, K. Jeffay, F.D. Smith, R.M. Taylor II, Dell Computer Corporation Strategic Technology and Research (STAR) program, December 2000, 1 year, $25,000.

Rate-based Scheduling Technology for Latency-Sensitive Graphics Applications, J. Anderson, S. Baruah, K. Jeffay, R. Taylor, National Science Foundation, grant number ITR 00-82866, September 2000, 3.5 years, $350,000.

Active Router Queue Management for Congestion Control and Quality-of-Service, K. Jeffay, F.D. Smith, National Science Foundation, grant number ITR 00-82870, September 2000, 3 years, $451,903.

Internet Measurements and Analysis to Support the nanoManipulator and Tele-Immersion, F.D. Smith, K. Jeffay, P. Jones, IBM Shared University Research Grant, September 2000, 1 year, $250,000.

The Performance of Differentiated Services Implementations for Supporting Distributed Virtual Environment Applications, K. Jeffay, North Carolina Network Initiative, September 2000, 1 year, $25,000.

Empirical Application-Workload Characterizations for Advanced Networks, K. Jeffay, F.D. Smith, Sun Microsystems, September 2000, $79,000.

Empirical Application-Workload Characterizations for Advanced Networks, F.D. Smith, K. Jeffay, MCNC, September 2000, 1 year, $50,000.

A Quality-of-Service Network for the nanoManipulator, K. Jeffay, Cabletron Inc. and the North Carolina Network Initiative, September 1999, $25,000.

Interactive Graphics for Molecular Studies and Microscopy — Supplement for Collaborative Science, F.P. Brooks, D. Erie (Chemistry), K. Jeffay, J. Samulski (Gene Therapy), F.D. Smith, D. Sonnenwald (Information and Library Science), R. Superfine (Physics and Astronomy), R. Taylor, National Institute of Health, October 1998, 4 years, $1,902,544.

Computing Power for Collaborative Science, S.R. Aylward, G. Bishop, D.W. Brenner, E. Bullitt, E.L. Chaney, V.L. Chi, B.J. Dempsey, N. England, A.G. Gash, D. Fritsch, H. Fuchs, B. Hemminger, J. Hermans, K. Jeffay, K. Keller, A. Lastra, M. Lin, D. Manocha, L.S. Nyland, S.M. Pizer, J.W. Poulton, J.F. Prins, J. Rosenman, F.D. Smith, R. Superfine, R.M. Taylor, S. Tell, G. Tracton, A. Tropsha, S. Washburn, G. Welch, Intel Corporation, August 1998, 3 years, $2,858,747. PI on *Multimedia Networking* and *A Distributed Teaching Laboratory for Networking and Internet Technologies* sections (F.D. Smith, B.J. Dempsey, Co-PIs), $557,669.

Empirical Application-Workload Characterizations for Advanced Networks, F.D. Smith, K. Jeffay, North Carolina Network Initiative, September 1998, $40,000.

Congestion control for high-speed networks, F.D. Smith, K. Jeffay, IBM Corporation, 1998, $20,000.

A Communications Middleware For Immersive Distributed Virtual Environments, K. Jeffay, North Carolina Network Initiative, 1998, $15,000.

Technology for Real-Time Services in Protocol Stack Implementations, K. Jeffay, IBM Corporation,

1997, $80,000.

Internet Access to UNC-CH Advanced Microscopy Facilities for Science Education Outreach, K. Jeffay, R. Superfine (Physics and Astronomy), G. Jones (Education), University of North Carolina at Chapel Hill Instructional Technology Award, 1997, $29,750.

Support for the Real-Time Technology and Applications Symposium, K. Jeffay, Office of Naval Research, 1997, $7,000.

Research Experience For Undergraduates supplement to *Object Sharing Technology For Real-Time Systems*, J. Anderson, K. Jeffay, National Science Foundation, 1996, $10,000.

Infrastructure for Research in Collaborative Systems, S.F. Weiss, J.B. Smith, K. Jeffay, P. Dewan, P. D. Stotts, D.K. Smith, F.D. Smith, W. Oliver (U.S. Armed Forces Institute of Pathology), National Science Foundation, grant number CDA-9624662, August 1996, 5 years, $1,260,830.

Collaboration Bus: An Infrastructure for Supporting Interoperating Collaborative Systems, P. Dewan, K. Jeffay, H. Abdel-Wahab (Old Dominion University), P. D. Stotts, L. Nyland, J.B. Smith, J. Mchugh (Portland State University), J. Menges (Hewlett Packard), Advanced Research Projects Agency, grant number 96-06580 (High Performance Distributed Services Technology), 1996, $973,334.

Object Sharing Technology For Real-Time Systems, J. Anderson, K. Jeffay, National Science Foundation, grant number CCR 95-10156, 1996, $209,442.

Flexible Shared Windows, P. Dewan, K. Jeffay, National Science Foundation, grant number IRIS 95-08514, 1995, $343,811.

An ATM Testbed For Multimedia Networking and Computer-Supported Cooperative Work, K. Jeffay, IBM Corporation, 1995, $400,000.

An Examination of Flow and Congestion Control Mechanisms for Media Transmission in Collaborative Systems, K. Jeffay, IBM Corporation, 1995, $125,242.

Software Infrastructure for the Rapid Development of Interactive and Collaborative Educational Simulations, J.F. Prins, K. Jeffay, P.D. Stotts, L.S. Nyland, Advanced Research Projects Agency, grant number 95-36871 (Computer Aided Education and Training Initiative), 1995, $200,000.

A Proposal For Network Routers, K. Jeffay, IBM Corporation, Research Triangle Park, NC, 1994, $40,000.

System Support For Video Teleconferencing Across Local Area Networks, K. Jeffay, Intel Corporation, 1993, $253,385.

The Integration and Use of Digital Audio and Video in Desktop Computing Environments, K. Jeffay, IBM Corporation, 1993, $100,000.

Construction of a UNC-Tektronix MBONE Link, K. Jeffay, Tektronix Corporation, 1993, $10,000.

An Empirical Determination of the Limits of Human Perception of Properties of Digital Audio and Video Streams, K. Jeffay, University of North Carolina Research Council, 1993, $2,000.

Processor and Resource Allocation Problems in Hard-Real-Time Systems: Theory and Practice, K. Jeffay, National Science Foundation, RIA Award, 1991, $59,400.

Accommodating Continuous Media in a Local Area Network: An exercise in distributed real-time computing, K. Jeffay, University of North Carolina Junior Faculty Development Award, 1990, $3,000.

Building and Using a Collaboratory: A Foundation for Supporting and Studying Group Collaborations, J.B. Smith, F.D. Smith, P. Calingaert, K. Jeffay, J.R. Hayes, D. Holland, M. Lansman, National Science

Foundation, grant number ICI-9015443, 1990, $946,000.

Enhanced program of research and teaching in communications software and distributed systems, K. Jeffay, IBM Corporation, 1989, $429,000.

The Design and Construction of Predictable Real-Time Systems, K. Jeffay, Digital Faculty Program Award, Digital Equipment Corporation, 1989, $180,000.

Invited Presentations

Keynote Addresses

Network Neutrality Considered Harmful, Revisiting the Quality-of-Service Morass

ACM International Workshop on Network and Operating System Support for Digital Audio and Video, Newport, RI, May 2006.

Rate-Based Resource Allocation Methods for Multimedia Computing

SPIE Multimedia Computing and Networking 2003, Santa Clara, CA, January 2003.

Network Support For Distributed Virtual Environments: The Tele-nanoManipulator

NCNI Advanced Networking Symposium, Research Triangle Park, NC, May 1999.

The Internet 2 Spring Members Meeting (presented jointly with R.M. Taylor), Washington, DC, April 1999.

Network Support For Distributed, Immersive, Virtual Environments

Workshop on Automated Control of Distributed Instrumentation, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL, April 1999.

The Future of Networking: The Networking Revolution Has Yet to Begin

North Carolina Governor's Board of Science and Technology Retreat, Banner Elk, NC, September 1998.

Distinguished Lectures

The Synthetic Traffic Generation Problem: The least sexy problem in computer networking

University of Minnesota, Minneapolis, MN, January 2008.

The Effect of Active Queue Management on Web Performance: The Good, the Bad, and the Ugly

University of Nebraska, Lincoln, NE, November 2003.

University of Pennsylvania, April 2004.

The Evolution of Quality-of-Service on the Internet

Mälardalen University, Sweden, February 2001.

Colloquia

The Evolution of Quality-of-Service on the Internet

IEEE Computer Society Seattle Chapter, Seattle, WA, September 2005.

Modeling and Generating TCP Application Workloads

Georgia Institute of Technology, Atlanta, GA, September 2005.

Microsoft Research, Redmond, WA, September 2005.

Worcester Polytechnic Institute, Worcester, MA, July 2005.

Cisco Systems, RTP, NC, May 2005 (presently jointly with F. Hernandez-Campos and F.D. Smith).

A Rate-Based Execution Abstraction For Embedded Real-Time Systems

University of Pennsylvania, April 2004.

How "Real" Can Synthetic Network Traffic Be?

University of Virginia, March 2004.

Non-Parametric Approach to Generation and Validation of Synthetic Network Traffic

Columbia University, New York, NY, January 2004.

Is Explicit Congestion Notification (ECN) Worthwhile?

Cisco Systems, San Jose, CA, October 2003.
Intel Architecture Labs, Hillsboro, OR, October 2003.

Tuning RED for Web Traffic: RED Considered Harmful?

Sprint Advanced Technology Research Laboratories, San Francisco, CA, March 2000.
University of Illinois at Urbana-Champaign, Urbana, IL, March 2000.

Internet Traffic: Measurement & Generation

Ganymede Software, Research Triangle Park, NC, June 1999. (Presented jointly with F.D. Smith.)

Lightweight Active Router-Queue Management for Multimedia Networking

University of Toronto, October 2000.
HP Laboratories, Palo Alto, CA, January 1999.
Sprint Advanced Technology Research Laboratories, San Francisco, CA, January 1999.

A Better-Than-Best-Effort Service For UDP: Lightweight Active Router-Queue Management for Multimedia Networking

University of Virginia, January 1999.

Congestion Control Mechanisms for Real-Time Communications on the Internet

IBM Networking Systems, Research Triangle Park, NC, November 1998.

A Better Than Best-Effort Forwarding Service For UDP

Microsoft Research, Redmond, WA, March 1998.
IBM Networking Systems, Research Triangle Park, NC, March 1998.

Multimedia Networking Research at UNC Chapel Hill

IBM Networking Systems, Research Triangle Park, NC, February 1998.
Lucent Technologies, Bell Laboratories, Holmdel, NJ, June 1997.

Design Principles for Distributed, Interactive, Virtual Laboratories

Mitsubishi Electric Research Laboratory, Cambridge, MA, January 1998.
Lucent Technologies, Bell Laboratories, Holmdel, NJ, June 1997.

The Performance of Two-Dimensional Media Scaling for Internet Videoconferencing

Mitsubishi Electric Research Laboratory, Cambridge, MA, January 1998.
Carnegie Mellon University, Pittsburgh, PA, March 1997.

Some Approaches to Enabling Real-Time Computation on Desktop Operating Systems

Honeywell Technology Center, Minneapolis, MN, October 1996.

Two-Dimensional Scaling Techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams

Oregon State University, Corvallis, OR, May 1996.

A Router-Based Congestion Control Scheme For Real-Time Continuous Media

Intel Architecture Development Laboratory, Hillsboro, OR, May 1996.
NetEdge Inc., Research Triangle Park, NC, March 1996.
North Carolina State University, Raleigh, NC, March 1996.

A Hybrid Reservation-Based/Best-Effort Transmission Scheme For Real-Time Transmission of Multimedia Data

Intel Research Council, Hillsboro, OR, November 1995.

Operating System Support For Multimedia Computing

IBM Networking Systems, Research Triangle Park, NC, July 1995.

Network Support For Multimedia Computing

IBM Networking Systems, Research Triangle Park, NC, July 1995.

A Rate-Based Execution Abstraction For Multimedia Computing

Tektronix Computer Research Laboratory, Beaverton, OR, May 1996.
INRIA, Rocquencourt, France, December 1995.
AT&T Bell Laboratories, Murray Hill, NJ, July 1995.

A Theory of Rate-Based Scheduling

Carnegie Mellon University, Pittsburgh, PA, August 1994.

Demonstration of Videoconferencing Over Packet-Switched Internetworks

Global Lecture Hall of the U.S.-Russian Electronic Distance Education System/TELE-TEACHING ‘93, Trondheim, Norway (via satellite from Chapel Hill), August 1993.

Transport and Display Mechanisms For Multimedia Conferencing Across Packet-Switched Networks

York University, York, England, November 1993.
Purdue University, West Lafayette, IN, February 1993.
Intel Architecture Development Laboratory, Hillsboro, OR, February 1993.
Oregon Graduate Institute, Beaverton, OR, February 1993.
Second MCNC/NSF Packet Video Workshop, Research Triangle Park, NC, December 1992.
Carnegie Mellon University, Pittsburgh, PA, November 1992.

Multimedia and Conferencing Research at UNC-CH

IBM Multimedia Networking, Research Triangle Park, NC, September 1992.

System Support for Synchronous Collaboration

North Carolina Artificial Intelligence and Advanced Computing Symposium, Raleigh, NC, March 1992.

Software Architectures for Predictable Real-Time Computer Systems

Carnegie Mellon University, Pittsburgh, PA, March 1992.

Operating System Requirements for Digital Audio and Video

MCNC/NSF Packet Video Videoconferencing Workshop, Research Triangle Park, NC, December 1991.

Some Experiments With Live Digital Audio and Video on a Local-Area Network

IBM European Networking Center, Heidelberg, Germany, November 1991.

Network and Operating System Support for Digital Audio and Video

Duke University, Durham, NC, November 1994.
Carnegie Mellon University, Pittsburgh, PA, April 1994.
Intel Corporation, Hillsboro, OR, December 1993.
IBM Networking Systems, Research Triangle Park, NC, June 1992.
IBM TJ Watson Research Center, Yorktown Heights, NY, March 1992.
University of Washington, Seattle, WA, February 1992.
Old Dominion University, Norfolk, VA, October 1991.

Systems Research in the UNC Collaboration Project

Intel Architecture Development Laboratory, Hillsboro, OR, May 1992.
University of Colorado, Boulder, CO, March 1991.

Some Deterministic Resource Allocation Problems in Real-Time Computer Systems

Research Triangle Institute, Research Triangle Park, NC, June 1990.
University of North Carolina at Chapel Hill, Department of Operations Research, Chapel Hill, NC, October 1989.

The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations

Carnegie-Mellon University, Pittsburgh, March 1991.
Duke University, Durham, NC, June 1990.
Technical University of Denmark, Lyngby, Denmark, September 1989.
University of California at Davis, Davis, CA, April 1989.
University of Minnesota, Minneapolis, MN, April 1989.
Purdue University, West Lafayette, IN, April 1989.
University of North Carolina at Chapel Hill, Chapel Hill, NC, April 1989.
University of British Columbia, Vancouver, BC, Canada, April 1989.
Rice University, Houston, TX, March 1989.
University of Arizona, Tucson, AZ, March 1989.
Oregon Graduate Center, Beaverton, OR, March 1989.
Tektronix Computer Research Laboratory, Beaverton, OR, March 1989.
University of Colorado, Boulder, CO, March 1989.
IBM TJ Watson Research Center, Yorktown Heights, NY, March 1989.
IBM Systems Integration Division, Owego, NY, March 1989.
New York University, New York, NY, March 1989.
Bell Communications Research, Morristown, NJ, March 1989.

Concurrent Programming With Time

Olivetti Research Center, Menlo Park, CA, December 1987.

IBM TJ Watson Research Center, Yorktown Heights, NY, September 1987.
Columbia University, New York, NY, September 1987.

Short Courses

Trends in Congestion Control and Quality-of-Service: Active queue management on the Internet of the future

Mälardalen University, Sweden, February 2001.

Rate-Based Execution Models For Real-Time Multimedia Computing

Scuola Superiore Santa Anna, Pisa, Italy, September 1997.

Tutorials

Issues in Multimedia Delivery Over Today's Internet

IEEE International Conference on Multimedia Computing Systems, Austin, TX, June 1998.

Systems Issues in the Design and Realization of Desktop Videoconferencing Systems,

Second ACM International Conference on Multimedia, San Francisco, CA, October 1994.