Flow Identification for Supporting Resource Reservation

Zhiruo Cao *Zheng Wang*

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
zhiruo@cc.gatech.edu

Bell Labs
Lucent Technologies
Holmdel, NJ 07733
zhwang@dnrc.bell-labs.com

GIT–CC–99/15

May, 1999

Abstract

This paper considers the problem of flow identification for supporting resource reservation. We propose several hashing-based schemes for flow identification and present a quantitative analysis of their performance and scalability limits. Of the hash functions we studied using simulation with real traffic traces, 32-bit CRC and XOR-folding of the five-tuple demonstrate excellent performance, both on the memory requirement for a collision rate target, and on the number of collided flows on average and in the worst-case. Our findings show that, with hashing-based schemes, it is feasible to implement flow identification at high speeds to support hundreds of thousands of reserved flows.

College of Computing

Georgia Institute of Technology

Atlanta, Georgia 30332–0280

1 Introduction

Over the last several years there has been considerable interest in multimedia communications over the Internet. Unlike the traditional data applications such as TELNET, FTP and email, real-time applications such as video conferencing and Internet telephony usually have stringent delay and jitter requirements; they do not work well over the best-effort Internet where delay and losses can not be easily controlled.

In response to the new challenges of supporting real-time applications in the Internet, the Internet Engineering Task Force (IETF) has standardized the Integrated Services model [2], which allows explicit end-to-end resource reservation. The RSVP protocol [3] has been developed to signal resource requirements and install reservation state inside the network.

To support the Integrated Services in the network, an IP router has to implement two key functions: flow identification and packet scheduling [2]. When the RSVP protocol installs a reservation, it adds an entry to the reservation table. In the packet forwarding path, routers must examine every incoming packet and decide if the packet belongs to one of the reserved RSVP flows. An IP flow is identified by the five-tuple in the packet header (source IP address, destination IP address, protocol ID, source port and destination port). To determine if a packet is from one of the reserved flows, the flow identification module has to compare the five-tuple of the incoming packet with the five-tuples of all flows in the reservation table. If there is a match, the corresponding reservation state is retrieved from the reservation table, and the packet is dispatched to a queue specifically set up for that flow. Some variation of weighted fair queuing algorithms can be used to guarantee delay and bandwidth committed to the flow [4, 6, 1].

Most Internet backbones currently operate at the OC12 (622 Mbps) speed and are expected to upgrade to OC48 (2.5 Gbps) soon. A single backbone trunk can easily have tens of thousands of concurrent flows [7]. Performing flow identification at wire-speed with a large number of RSVP flows is a non-trivial task. Because of this, there have been debates about the scalability of the Integrated Services and RSVP; some people believe

that routers are simply not able to support end-to-end reservation in the backbone. However, we have not seen any quantitative studies in the literature on the scalability issues of implementing end-to-end reservation.

In this paper we address one of the issues for end-to-end reservation, specifically, the problem of flow identification, and present a quantitative analysis on its scalability limits. We propose a hashing-based approach for flow identification and evaluate the performance and scalability limit of several hashing-based schemes, both analytically and using simulation based on real traffic traces. Of the hash functions we studied, 32-bit CRC and XOR-folding of the five-tuple demonstrate excellent performance, both on the memory requirement for a collision rate target, and on the number of collided flows on average and in the worst-case. Our findings show that, with hashing-based schemes, it is feasible to implement flow identification at high speeds to support hundreds of thousands of reserved flows.

The rest of the paper is organized as follows. The next section discusses the key issues in implementing flow identification. In Section 3, we describe several hashing-based schemes. We present the detailed simulation results in Section 4. Section 5 concludes the paper.

2 Flow Identification in Routers

The basic problem of flow identification can be summarized as the following: Given the five-tuple of an incoming packet and a reservation table with five-tuples of reserved flows, we want to determine if an incoming packet matches one of the flows in the reservation table, and if so, retrieve the reservation information of the matched flow (Figure 1).

Flow identification has to be performed on every packet, thus it is essential that the operation be very fast and complete within the amount of time available for processing a packet; at high speed the per-packet processing time is extremely small (e.g., about 1 microsecond for a 64 bytes IP packet at OC12 speed).

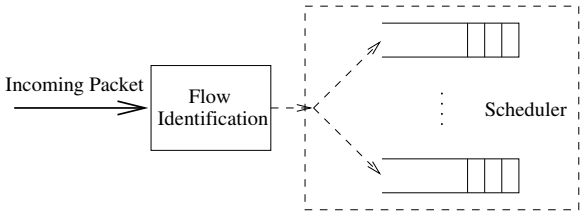


Figure 1: Flow Identification

There are a number of possible approaches for implementing flow identification, all involving speed versus space tradeoffs. One extreme is a direct memory lookup, requiring only a single memory access. But this approach is not practical as the five-tuple is 104-bit long.

The other extreme is binary search. Binary search has the most efficient memory usage but is relatively slow. For example, to support 64K reserved flows, 17 memory accesses and comparisons may be needed to identify a flow.

In this paper, we examine hashing-based schemes for flow identification. Hashing offers a good tradeoff between speed and memory requirements. Hashing-based flow identification is simple to implement. It involves the calculation of a hash function and a small number of comparisons if there is a collision.

The performance of a hashing-based scheme depends on the collision rate. Generally, increasing the size of hash table reduces the collision rate. This gives us a knob to fine tune the tradeoff between speed and space based on the actual memory and speed requirements.

The main objective of this paper is to study the performance of different hash functions, the tradeoff between speed and space, and the scalability of the hashing-based approaches. Specifically, we try to address the following questions:

- Which hash functions produce good performance
- How much memory is needed for a reasonable collision rate target
- Is it possible to support hundreds of thousands of reserved flows at high speeds

In the next two sections, we first present several hashing-based schemes, and then present the details of our findings.

3 Hashing-Based Approach

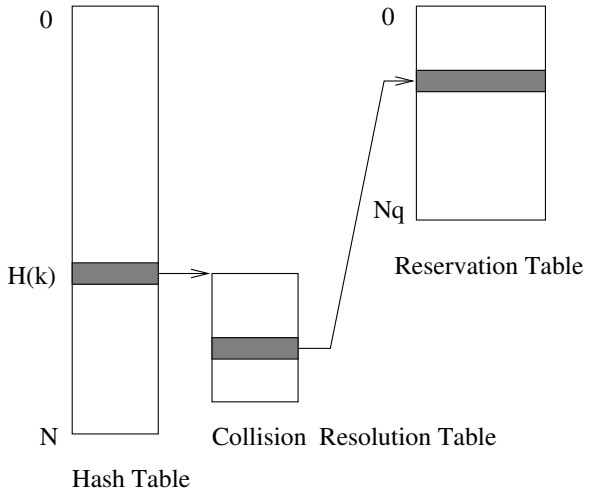


Figure 2: Hashing-based Flow Identification

Figure 2 illustrates a hashing-based system for flow identification. When a RSVP reservation is made, a router applies a hash function to the five-tuple of a packet that identifies the reserved flow. If the output hash value has not been used by other reserved flows, the router can simply associate all reservation state (e.g., bandwidth etc.) with that hash value. If other flows have been hashed into the same position, we need to resolve the collision. A simple approach is to set up a collision resolution table to hold the five-tuples of all collided flows with the same hash value.

In the packet forwarding path, the router applies the same hash function to the five-tuple of each incoming packet. If the hash value has no collision, it can be used to retrieve the reservation state for the flow. Otherwise, the router needs to compare the five-tuple of the incoming packet with all flows in the collision resolution table in order to find the exact match. Once the reservation state is retrieved, the packet is dispatched to the scheduler.

The average performance of the system depends on the average collision rate, and

the worst case performance is determined by the maximum number of flows that may have the same hash value. If the collision rate is very low, it may be acceptable to skip the collision resolution step; this is a possible engineering tradeoff between precision and speed.

Figure 3: 104-bit 5-Tuple as Flow ID

We now describe the four hash functions and one double hashing that we consider in our study. To reduce the implementation complexity, the first two hash functions only use two fields from the five-tuple for hash computation. The other two use the five-tuple as a 104-bit string (Figure 3).

3.1 XOR-Folding of Source and Destination IP Addresses

This hashing scheme concatenates source and destination IP addresses into a 64-bit string and performs XOR-folding according to the hash table size. The computation is very simple and does not need access to the layer four information in the packet. Mathematically, it can be expressed as:

$$H(\cdot) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$$

where A_i's are l-bit long for a hash table size 2^l and the concatenation of A_i's forms the 64-bit source and destination IP addresses plus zero paddings at the end for l-bit-wise alignment. For example, if the hash table has $1M$ entries, the hash function can be implemented in the following code:

```
hash_index = srcIP >> 12;
temp = (srcIP << 8) | (destIP >> 24);
hash_index ^= temp;
temp = destIP >> 4;
hash_index ^= temp ^ (destIP << 16);
```

```
return hash_index;
```

3.2 XOR-Folding of Destination IP Addresses and Destination Port

For many streaming applications, data is delivered from the server towards the client. In such cases, the destination address and port number tend to be more effective in distinguishing a flow from others. Thus, we also look at a simple variation of the previous hash function, by replacing the source address with destination port number. The XOR-folding of destination IP address and destination port number can be expressed as:

$$H(\cdot) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$$

where A_i's are l-bit long for a hash table size 2^l and the concatenation of A_i's forms the 48-bit (destination IP addresses, destination port number) plus zero paddings at the end for l-bit-wise alignment.

3.3 XOR-Folding of Five-Tuple

This hash function concatenates all bits of the five-tuple into a 104-bit string, and performs XOR-folding according to the hash table size. For example, when the hash table address is 20-bit long, the hash function can be implemented in the following code:

```
hash_index = srcIP >> 12;
temp = (srcIP << 8) | (destIP >> 24);
hash_index ^= temp;
temp = destIP >> 4;
hash_index ^= temp;
temp = (destIP << 16) | srcPort;
hash_index ^= temp;
temp = (destPort << 4 ) | (proto >> 4);
hash_index ^= temp;
temp = protocol & 0xF;
hash_index ^= temp;
```

```
return hash_index;
```

The five-tuple contains all information for identifying a flow, thus we expect this scheme to perform better than the previous two. The interesting question is how much the improvement will be.

3.4 32-bit CRC

The 32-bit CRC algorithm, as defined by ISO 3309, is known to be able to exploit the randomness in traffic well, thus it could be a good hash function for flow identification. However, the 32-bit CRC (also called CRC32) is much more complex compared to the previous hash functions we have described, and thus it should be considered only when it can add significant improvement.

The CRC polynomial employed is[1]:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

The hash table index is computed by applying the CRC32 algorithm to the five-tuple of the incoming packet modulo the hash table size N.

3.5 Double Hashing

We also consider the technique of double hashing [5], where a second hash value is computed with a different hash function if the first hashing causes a collision. In double hashing, a collision occurs when a collision is observed on both hash functions. The probability of a collision with two different hash functions is likely to be substantially smaller than that with one hash function. However, the extra complexity has to be justified with substantial increase in performance.

In the simulation, we use hash function XOR-folding of full five-tuple as the first hash function H_1, and use XOR-folding of source and destination IP addresses as the second one, i.e., H_2.

[1] A sample CRC implementation can be found at `http://www.w3.org/TR/REC-png#CRC-algorithm`.

3.6 Performance of Perfect Hashing

Before we present our simulation results in the next section, let us first look at the theoretic performance limits of hashing-based schemes. For the purpose of evaluating the performance of hashing-based flow identification, we define the *collision rate* as the proportion of total active flows that are hashed into places already occupied.

Assume that we have a perfect hash function that can uniformly map random flows into a hash table. Let N be the size of hash table and m be the number of distinguished flows that are hashed into the table. We show in the Appendix that collision rate can be expressed as

$$Cr = 1 - \frac{N(1 - (\frac{N-1}{N})^m)}{m}.$$

Figure 4(a) shows the relationship between the collision rate (Cr) and the hash table size N for a given number of active flows m. The collision rate decreases quickly when the hash table size increases. The drop in the collision rate starts to level off after the hash table size is about 10 times larger than the number of flows.

Figure 4(b) shows the relationship between the collision rate (Cr) and the number of active flows (m) for a given hash table size. The collision rate increases almost linearly with the number of flows.

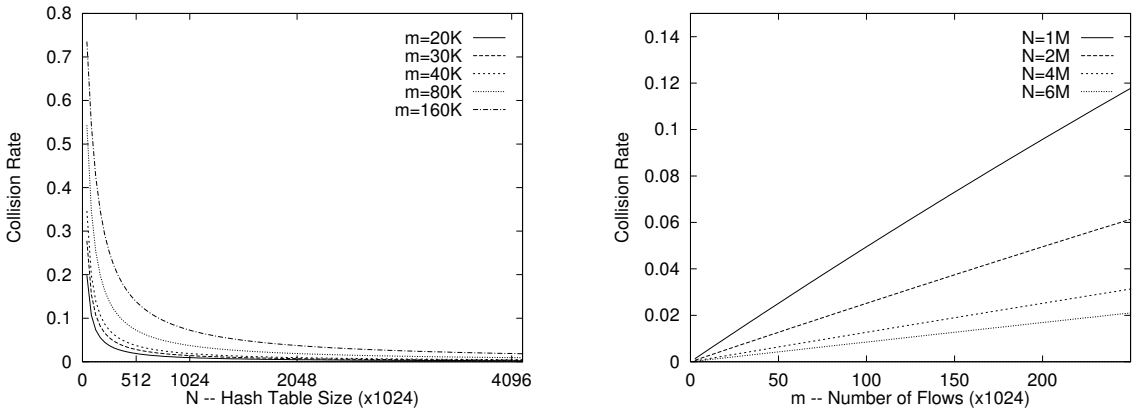


Figure 4: Collision rate versus flow number and hash table size

In practice, hash functions typically will not be able to match the performance of the perfect hash function we have described. The distribution of the bits in the five-tuple is not completely random, and the port numbers in particular tend to concentrate on a small number of values (e.g., 80 for web servers). However, a good hash function should be able to exploit all inherent randomness in flow identifiers, i.e., the five-tuples.

4 Simulation Results

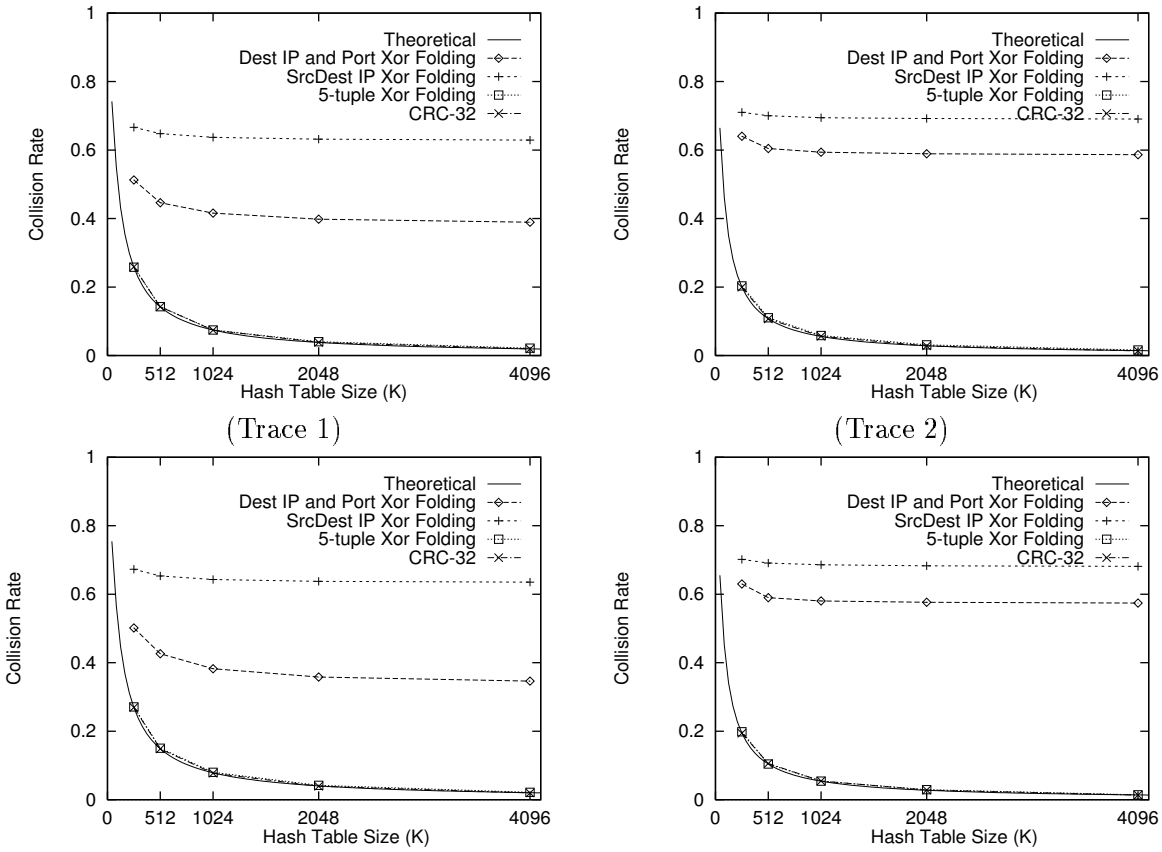
We now present the simulation results on the performance of hashing-based flow identification. Our simulation is based on real packet traces collected on a major Internet backbone (over one OC12 trunk and one OC3 trunk) during the summer of 1998. The four sets of traces contain complete packet headers and timing information of about 8 million packets. Traces 1 and 3 came from an OC12 trunk, while traces 2 and 4 came from an OC3 trunk.

There are 164K, 112K, 173K, and 117K different flows in the four sets of traces. It is unlikely that all flows would require reservation. However, in order to examine the scalability limits of the schemes, we assume in our study that all flows are reserved flows. The four sets of traces were collected over four intervals of approximately 14 seconds each. Given the relatively short duration, we also assume that all flows in the traces are active during the simulation; no garbage collection on the hash table is performed.

We used a trace-based simulation tool to generate the collision rates and the maximum number of collided flows for hash table sizes ranging from 256K and 4M.

4.1 Collision Rate

Figure 5 shows the collision rate with the first four hash functions we described in Section 3, together with that of the perfect hashing. Both the CRC32 and the XOR-folding of the five-tuple perform extremely well, close to the curve of perfect hashing; the performances of the XOR-folding of source and destination addresses and the XOR-folding of destination address and destination port number are rather poor. The results



(Trace 1) (Trace 2)

(Trace 3) (Trace 4)

Figure 5: Collision rate with four single hashing

are very consistent across all four sets of traces. For CRC32 and XOR-folding of five-tuple, the amount of memory needed to reach a small collision target is very reasonable.

We then compared the CRC32 and double hashing in a close-up examination. While the CRC32 and double hashing both show excellent performance, double hashing based on less complex functions outperform CRC32, although the difference is not significant (see Figure6).

4.2 Worst-Case Number of Collided Flows

For collided flows, it is important to know how many flows that have been hashed to the same hash table address. In particular, the worst-case number of collided flows determines the maximum number of comparisons needed to perform to find the exact

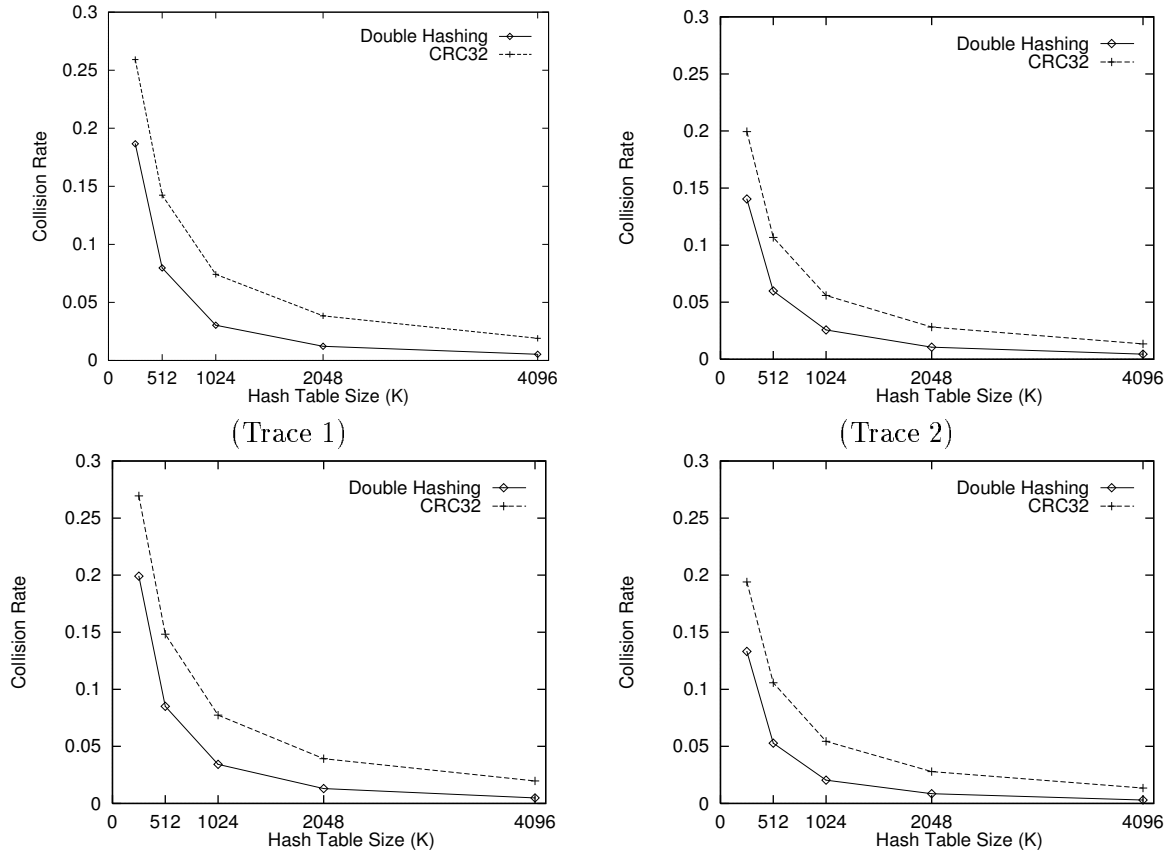


Figure 6: Collision rate with 32-bit CRC and double hashing

match. To achieve wire-speed forwarding, routers have to be designed to deal with the worst-case. Therefore, the worst-case number is a good indication of the schemes' scalability.

Table 1 shows the average and worst-case number of collided flows. For the average numbers, all four hash functions perform well, although CRC32 and the XOR-folding of five-tuple are slightly better than the other two. For the worst-case numbers, however, the difference between the worst and the best is significant; CRC32 and XOR-folding of five-tuple have the worst-case numbers consistently below 7, while XOR-folding of source and destination addresses, and XOR-folding of destination address and destination port have the worst-case numbers over 1000.

One may notice that the worst-case numbers for XOR-folding of source and desti-

	Table Size(k)	Average				Worst Case			
		SrcDestIP	Dest	5-tuple	CRC32	SrcDestIP	Dest	5-tuple	CRC32
Trace 1	256	3.00	2.05	1.348	1.350	1284	2353	7	6
	512	2,84	1.81	1.166	1.166	1284	2350	5	6
	1024	2.76	1.71	1.081	1.080	1284	2350	4	5
	2048	2.72	1.66	1.042	1.040	1284	2350	4	5
	4096	2.70	1.64	1.022	1.020	1284	2351	4	4
Trace 2	256	3.45	2.78	1.255	1.249	4601	3277	6	6
	512	3.34	2.53	1.124	1.120	4601	3277	5	5
	1024	3.27	2.46	1.062	1.060	4601	3277	4	4
	2048	3.25	2.43	1.032	1.030	4601	3277	4	3
	4096	3.23	2.42	1.016	1.014	4601	3277	3	3
Trace 3	256	3.06	2.01	1.372	1.369	1076	1390	7	7
	512	2.89	1.74	1.177	1.174	1076	1390	5	6
	1024	2.80	1.62	1.087	1.084	1076	1390	5	4
	2048	2.76	1.56	1.044	1.041	1076	1390	4	4
	4096	2.74	1.53	1.022	1.020	1076	1390	4	4
Trace 4	256	3.35	2.70	1.247	1.241	1242	3898	6	6
	512	3.24	2.44	1.117	1.118	1242	3898	5	5
	1024	3.18	2.38	1.057	1.058	1242	3898	4	4
	2048	3.15	2.36	1.030	1.029	1242	3898	4	3
	4096	3.14	2.35	1.015	1.014	1242	3898	3	3

Table 1: Average and worst-case numbers of collided flows

nation addresses, and XOR-folding of destination address and destination port remain unchanged when the hash table size is increased. A close examination of the simulation logs reveals that the worst-case numbers are approximately the numbers of flows that have the same source/destination addresses and destination address/destination port numbers. Thus, hashing with the two fields as input cannot distinguish them.

We also compared double hashing and CRC32 on worst-case numbers of collided

Table Size (k)	Trace 1		Trace 2		Trace 3		Trace 4	
	CRC32	DHash	CRC32	DHash	CRC32	DHash	CRC32	DHash
256	6	6	6	6	7	6	6	5
512	6	4	5	4	6	4	5	5
1024	5	4	4	3	4	4	4	4
2048	5	4	3	3	4	3	3	3
4096	4	3	3	3	4	2	3	2

Table 2: Worst-case numbers of collided flows for CRC32 and double hashing

flows. The result is shown in Table 2. It indicates that double hashing improves the worst-case number slightly.

5 Conclusion

In this paper, we proposed and evaluated several hashing-based schemes for flow identification. Our simulation results, based on real traffic traces from OC12/OC3 backbone trunks, show that CRC32 and XOR-folding of the five-tuple perform, consistently, extremely well. Moreover, double hashing demonstrates it can offer even better performance at minimum cost.

Based on our findings, we conclude that, with hashing-based schemes, it is feasible to implement flow identification at gigabit speeds with several hundreds of thousands of reserved flows.

Acknowledgment

We would like to thank Kevin Thompson and Greg Miller of MCI for their help with the traffic traces.

References

[1] J. C. R. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. In *In Proceedings of SIGCOMM'96*, August 1996.

[2] R. Braden, D. Clark, and S. Shender. Integrated Services in the Internet Architecture: an Overview, RFC1633, June 1994.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification, RFC2205, September 1997.

[4] A. Demers, S. Keshav, and S. Shenkar. Analysis and Simulation of a Fair Queueing Algorithm. *SIGCOMM Symposium on Communications Architectures and Protocols*, September 1989.

[5] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, MA, 1973.

[6] A. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Feburary 1992.

[7] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, November/December 1997.

A The Proof of Hashing Collision Rate

Hashing random flows into a hash table is equivalent to randomly dropping balls into buckets. In the following conjecture and proposition, we assume $N > k$.

Conjecture A.1 *Consider dropping k balls randomly into N buckets. Let M_k be the average number of buckets that the k balls occupy. Then, $E(M_{k+1}) = E(M_k) \times \frac{E(M_K)}{N} + E(M_K) \times \frac{N-E(M_k)}{N}$.*

Proof: Simplify the above assertion,

$$\begin{aligned} E(M_{k+1}) &= E(M_k)\frac{E(M_K)}{N} + E(M_K)\frac{N - E(M_k)}{N} \\ &= E(M_k)(1 - \frac{1}{N}) + 1. \end{aligned}$$

Let P_k^i be the probability that k balls are dropped into i buckets of the total N buckets. When the $(k+1)$th ball is to be dropped, it is either dropped into one of the m buckets occupied by the first k balls, or it is dropped into one of the other unoccupied $N - m$ buckets. Therefore,

$$\begin{aligned} E(M_{k+1}) &= P_k^1\frac{1}{N} + P_k^2\frac{2}{N}2 + \ldots P_k^k\frac{k}{N}k + P_k^1\frac{N-1}{N}2 + P_k^2\frac{N-2}{N}3 + \ldots P_k^k\frac{N-k}{N}(k+1) \\ &= \frac{1}{N}(P_k^1(2N-1) + P_k^2(3N-2) + \ldots + P_k^k((k+1)N - k)) \\ &= \frac{1}{N}((N-1)(P_k^1 2 + P_k^2 3 + \ldots + P_k^k(k+1)) + P_k^1 + P_k^2 + \ldots + P_k^k). \end{aligned}$$

Note that $\sum_{i=1}^{k} P_k^i = 1$ and $\sum_{i=1}^{k} P_k^i i = E(M_k)$,

$$\begin{aligned} E(M_{k+1}) &= \frac{1}{N}((N-1)((P_k^1 1 + P_k^2 2 + \ldots + P_k^k k) + 1) + 1) \\ &= E(M_k)(1 - \frac{1}{N}) + 1 \end{aligned}$$

which proves the conjecture. □

We define the average collision rate C_r as the proportion, on average, of balls that are dropped in buckets already occupied. The following proposition reveals the relations among C_r, N and k.

Proposition A.1 $C_r = 1 - \frac{N(1-(\frac{N-1}{N})^k))}{k}$.

Proof: From Conjecture *A*.1, $E(M_k) = E(M_{k-1})(1 - \frac{1}{N}) + 1$. Note that we have boundary condition $E(M_1) = 1$. Solving the series, we have

$$E(M_k) = N(1 - (\frac{N-1}{N})^k).$$

The average collision rate C_r then can be calculated as

$$\begin{aligned} C_r &= \frac{k - E(M_k)}{k} \\ &= 1 - \frac{N(1 - (\frac{N-1}{N})^k)}{k}. \end{aligned}$$

□