Maintaining Statistics Counters in Router Line Cards

A network device stores and updates statistics counters. Using an optimal counter management algorithm minimizes required SRAM size and ensures correct line-rate operation for many counters.

Devavrat Shah
Sundar Iyer
Balaji Prabhakar
Nick McKeown
Stanford University

• • • • • • Packet switches (that is, IP routers and ATM and Ethernet switches) maintain statistics for performance monitoring, network management, security, network tracing, and traffic engineering. Counters usually collect such statistics as the number of arrivals of a specific packet type or they count a particular event, such as when the network drops a packet. A packet's arrival can lead to the updating of several different statistics counters.

The number of statistics counters in a network device and their rate of update are often limited by memory technology. On-chip registers or SRAM (on- or off-chip) can hold a few counters. Often, a network device has to maintain many counters and therefore must store them in off-chip DRAM. But the large random access times of DRAMs make their use difficult when supporting high-bandwidth links. The time it takes to read, update, and write a single counter would be too long, and worse still, each arriving packet can trigger the update of multiple counters.

To alleviate these problems, we use a well-known architecture for storing and updating statistics counters. This approach maintains smaller-size counters in fast (potentially on-chip) SRAM, while maintaining full-size counters in a large, slower DRAM. Our goal is to ensure that the system always correctly maintains counter values at line rate. An optimal counter management algorithm (CMA) minimizes the required SRAM size while ensuring correct line-rate operation for a large number of counters.

Role of packet switches

Packet switches perform many processing tasks on arriving packets. Jobs include address lookup, classification, buffering, quality-of-service scheduling, header editing, and statistics maintenance. Packet switches typically perform these tasks on the line cards of switches and routers, and therefore need to occur at line rate. When optical carrier line rates increase beyond Sonet specification OC-192 (10 Gbps) to OC-768 (40 Gbps), packet processing tasks will becomes more difficult. Although several proposed techniques deal with address lookup,[1] packet classification,[2] packet buffering,[3-5] and quality-of-service scheduling,[6] we are not aware of research besides ours that addresses the maintenance of a large number of statistics counters.

Packet switches maintain statistics for many reasons. These include firewall support (especially stateful inspection), intrusion detection, performance monitoring (for example, remote monitoring), network tracing, load balancing, and traffic engineering (for example, policing and shaping of traffic patterns). In addition, most packet switches maintain statistics counters to facilitate network management. We can characterize the general problem of statistics maintenance as follows: When a packet arrives, the router classifies an arriving packet to deter-

0272-1732/02/$17.00 © 2002 IEEE

mine what actions to perform on it—should it be accepted or dropped, receive expedited service or not, and so on. Depending on the chosen action, the router updates statistics counters.

We are interested in statistics that count events. For example, the number of fragmented, dropped, or arriving packets or the number of bytes forwarded, and so on. We refer to these types of statistics as counters. Here, we describe and quantitatively analyze the problem of maintaining these counters.

We are particularly interested in applications that maintain many counters, such as a routing table that counts how many times a packet uses each prefix, or a router that counts the packets belonging to each TCP connection. Both examples would require simultaneously maintaining several hundreds of thousands, or even millions, of counters, making it infeasible (or at least very costly) to store them in SRAM and hence requiring DRAM storage. Furthermore, we are interested in applications that have frequent updates, such as an OC-192c link in which the router updates multiple counters upon each packet arrival. These read-modify-write operations must occur at the same rate as packet arrival.

If each counter is M bits wide, then a counter update operation

- reads the M bit value stored in the counter,
- increments the M bit value, and
- writes back the updated M bit value.

If packets arrive at rate R (in gigabits per second), the minimum packet size is P bits, and if the router updates C counters each time a packet arrives, the memory may need to be accessed (either read or written) every $P/(2CR)$ ns. Let's consider the example of 40-byte TCP packets arriving on a 10-Gbps link; each arrival leads to the update of two counters. The memory needs to be accessed every 8 ns, about eight times faster than the random-access speed of today's commercial DRAMs.

It is a strict requirement that routers correctly update a counter or counters every time a packet arrives. Counters must account for every packet. If the scheme that updates counters performs an update operation every time a packet arrives and update C counters per packet, then minimum bandwidth R_D required on the memory interface where the counters are stored would be at least $2RMC/P$. Again, this bandwidth requirement can become unmanageable as the size of the counters and the line rates increase.

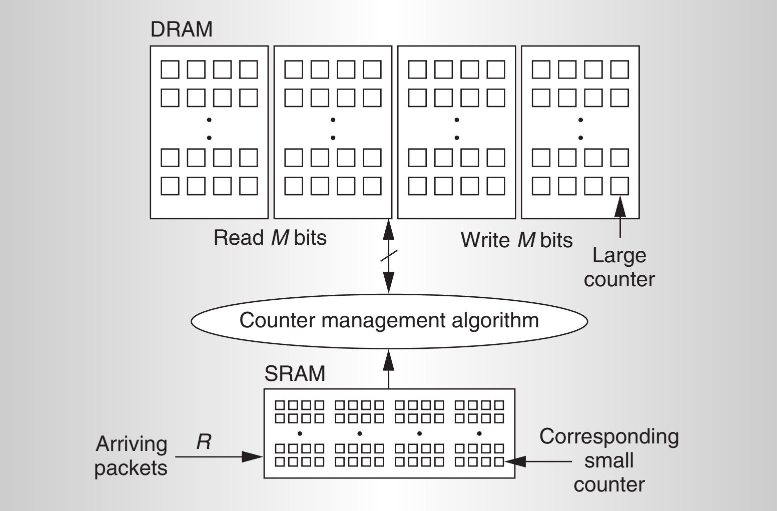

Figure 1. Memory hierarchy for the statistics counters. A fixed-size ingress SRAM (with N counters of width $m < M$ bits) stores the small counters, which are periodically transferred to the large counters in DRAM (with N counters of width M bits). The DRAM memory bandwidth decreases by factor b, the number of timeslots between updates of the large DRAM counters.

We propose an approach that uses DRAM to maintain statistics counters and a small fixed amount of (possibly on-chip) SRAM to support these operations. We assume that DRAM stores N counters of width M bits and that SRAM stores N counters of width $m < M$. The SRAM counters track the number of updates not yet reflected in the DRAM counters. Periodically, under the control of a CMA, our approach updates the DRAM counters by adding the values in the SRAM counters to the DRAM counters, as shown in Figure 1. Updating the DRAM counters relatively infrequently reduces memory bandwidth requirements.

Our approach derives strict bounds on the size of the SRAM so that—irrespective of the arriving traffic pattern—none of the SRAM counters overflow. DRAM access rate and bandwidth requirements decrease but still ensure correct counter operation.

SRAM size and DRAM access rate both depend on the CMA used. The largest-counter-first (LCF) CMA minimizes SRAM size. We derive necessary and sufficient conditions on counter sizes (and hence the SRAM that stores these counters), and prove that the LCF CMA is optimal.

As an example of how our technique can work, consider an OC-192c line card on a router that maintains a million counters. Assume that the maximum size of a counter is 64 bits and that each arriving packet updates a maximum of 10 counters. Our results indicate that such a system can use a statistics counter with a 51.2-ns DRAM access time, 1.25-Gbps DRAM memory bandwidth, and 9-Mbyte SRAM.

Memory hierarchy

Packets arriving at a switch have variable lengths; we denote *minimum packet size P* as the minimum length of a packet. The *time slot* is the time taken to receive a minimum-size packet at link rate R. We organize the SRAM as a statically allocated memory, consisting of separate storage spaces for each of the N counters. In this article, we assume that an arriving packet increments only one counter. If we instead considered the case where each packet arrival updates C counters, the line rate on the interface would be CR.

A large counter of size M bits in DRAM, and a small counter of size $m < M$ bits in SRAM represent each counter. The small counter counts the most recent events, and the large counter counts events occurring since the large counter was last updated. At any time, the correct counter value is the sum of the small and large counters.

Updating a DRAM counter consists of a read-modify-write operation: Read an M bit value from the large counter. Add the m bit value of the corresponding small counter to the large counter. Write the new M bit value of the large counter to DRAM. Reset the small-counter value.

Our goal is to decrease DRAM bandwidth by factor b, that is, $R_D = 2RM/(Pb)$, and increase DRAM access time accordingly, that is, access time $A_t = Pb/(2R)$. Thus, the CMA will update a large counter only once every b time slots. The minimum SRAM size is function g, which depends on N, M, and b. Therefore, the system designer can trade off SRAM size $g(N, M, b)$ with DRAM bandwidth R_D and access time A_t. (The system designer chooses variable b ($b \geq 1$). If $b = 1$, no SRAM is required, but the DRAM must be fast enough for all counters to reside in DRAM.

Count $C(i, t)$ is, at time t, the number of times that the ith small counter has been incremented since the ith large-counter update. An *empty counter* is when counter i is empty at time t; that is, $C(i, t) = 0$.

The correct large-counter value could be lost if the small counter overflows before it is added to the large counter. Our approach is to find the smallest possible size of counters in the SRAM and a suitable CMA such that the small counter cannot overflow before its corresponding large-counter update.

Necessity conditions on any CMA

For this hierarchy of counters to work, under any CMA the SRAM must meet certain conditions, which we define in the following theorem.

Theorem 1 (necessity): Under any CMA, a counter can reach a count $C(i, t)$ of

$$\frac{\ln\{(N-1)[b/(b-1)]^{(b-1)}\}}{\ln[b/(b-1)]}$$

Proof: We will argue that we can create an arrival pattern for which, after some time, there exists k such that there will be $(N - 1)/[(b - 1)/b]^k$ counters with count $k + 1$ irrespective of the CMA.

Consider the following arrival pattern. In time slot $t = 1, 2, 3, \ldots, N$, small counter t is incremented. Every bth time slot, one of the large counters is updated and the corresponding small counter reset to 0. Therefore, at the end of time slot N, there are $N(b - 1)/b$ counters with count 1 and N/b empty counters. During the next N time slots, the N/b empty counters are again incremented, and N/b^2 of these counters are used to update the large counter and then reset. Therefore, after $2N$ time slots there are $[N(b - 1)/b] + (N(b - 1/b^2)$ counters that have count 1.

In a similar way, we can make $N - 1$ counters have a count of 1 at time slot $N - 1$. During the next $N - 1$ time slots, all $N - 1$ counters are incremented once and $1/b$ of them are served—their corresponding large counter is updated and these small counters are reset to zero. Now assume that all of the remaining approximately N/b empty counters are incremented twice in the next $2N/b$ time slots, while $2N/b^2$ counters become empty due to service. Note that the number of empty counters decreased to $2N/b^2$ from N/b (if $b = 2$, there is no change). In this way, after some time, we see $N - 1$ counters of count two.

By continuing this argument, we can arrange for all $N - 1$ counters to have a count $b - 1$. We denote by T the time slot at which this first happens.

During the interval from time slot $2(N - 1)$ to $3(N - 1)$, all of the counters are again incremented, and $1/b$ of them are served. while the rest have a count of two. Service means that the counters are reset to zero. In the next $N - 1$ time slots each of the counters with size two are incremented, and again $1/b$ are served, while the rest have a count of three. Thus there are $(N - 1)/[(b - 1)/b]^2$ counters with a count of three. In a similar fashion, if only nonempty counters keep being incremented, after a while there will be $(N - 1)/[(b - 1)/b]^k$ counters with count $k + 1$. Hence there will be one counter with count

$$\frac{\ln(N-1)}{\ln[(b/(b-1)]} = \frac{\ln(N-1)+(b-1)\ln[b/(b-1)]}{\ln[b/(b-1)]}$$
$$= \frac{\ln[b/(b-1)^{b-1}(N-1)]}{\ln[b/(b-1)]}$$

Thus, there exists an arrival pattern for which a counter can reach a count $C(i, t)$ of

$$\frac{\ln\{(N-1)[b/(b-1)]^{(b-1)}\}}{\ln[b/(b-1)]}$$

A CMA that minimizes SRAM size

Every b time slots, the LCF CMA algorithm selects the counter i with the largest count. If multiple counters have the same count, LCF CMA arbitrarily picks one. LCF CMA updates the value of corresponding counter i in the DRAM and sets $C(i, t) = 0$ in SRAM.

Optimality

Key to establishing the LCF CMA's ability to minimize SRAM size in the concept of optimality, which we explain using the theorem that follows.

Theorem 2 (optimality of LCF CMA): Under all arriving traffic patterns, LCF CMA is optimal in the sense that it minimizes the count of the required counter.

Proof: We give a brief intuition of this proof here. Consider a traffic pattern from time t, which causes some counter C_i (which is smaller than the largest counter at time t) to reach maximum threshold M^*. A similar traffic pattern can cause the largest counter at time t to exceed M^*. This implies that not serving the largest counter is suboptimal. We provide a detailed proof elsewhere.[7]

Sufficiency conditions on LCF service policy

We must show that under the LCF service policy what size of SRAM is sufficient.

Theorem 3 (sufficiency): Under the LCF policy, count $C(i, t)$ of every counter is no more than $\ln(bN) / \ln[b/(b - 1)]$.

Proof (by induction): Let $d = b/(b - 1)$. Let $N_i(t)$ denote the number of counters with count i at time t. We define

$$F(t) = \sum_{i \geq 1} d^i N_i(t)$$

We claim under the LCF policy that $F(t) \leq bN$ for every time t. We shall prove this by induction. At time $t = 0$, $F(t) = 0 \leq bN$. Assume that at time $t = bk$ for some k that $F(t) \leq bN$. For the next b time slots, some b counters with count $i \geq i_2 \geq \ldots \geq i_b$ are incremented. For simplicity, we assume that the counter values are distinct (even though the proof does not require this assumption). After the counters are incremented, they have counts $i_1 + 1$, $i_2 + 1$, …, $i_b + 1$, and the largest counter among all the N counters is serviced. The largest counter has at least value $C(i,t) \geq i_1 + 1$.

Case 1. If all the counter values at time t were nonzero, then the contribution of these b counters in $F(t)$ is $\alpha=d^{i_1}+d^{i_2}+\ldots+d^{i_b}$. Consider the values of these counters' values after they are incremented, their contribution to $F(t + b)$ becomes $d\alpha$. But a counter with count $C(i,t) \geq i_1 + 1$ is served at time $t + b$ and its count becomes zero. Hence, the decrease to $F(t + b)$ is at least $d\alpha/b$. Thus, the net increase is at most $d\alpha[1 - (1/b)] - \alpha$. But $d [1 - (1/b)] = 1$. Hence, the net increase is at most zero, that is, if arrivals occur to nonzero queues, $F(t)$ can't increase.

Case 2. Now we address when one or more counters at time t are zero. For simplicity, assume that all incremented b counters are initially empty. For these empty counters, their contribution to $F(t)$ was zero, and their contribution to $F(t + b)$ is db. Again, the counter with the largest count among all N counters is served at time $t + b$. If $F(t) \leq (bN - db)$, then the inductive claim holds trivially. If not, that is, $F(t) > (bN - dB)$, then at least one of the $N - b$ counters, which did not get incremented, has count $i^* + 1$ such that $d^{i*} = b$. If this were not so, it contradicts the assumption $F(t) > bN - db$. Hence, a counter with count at least $i^* + 1$ is served, which decreases $F(t + b)$ by $d^{i*+1} = db$. Hence the net increase is zero. You can similarly argue this case when arrivals occur at fewer than b empty counters.

Thus, we have shown that, for all times t when the counters are served, $F(t) \leq bN$. This means that the counter value can not be larger than i_m, where $d^{im}= Nb$, that is, $C(i,t) \leq \ln(bn)/\ln(d)$. Substituting for d, we get the counter value bound $\ln(bN) / \ln[b / (b - 1)]$.

Theorem 4 (sufficiency): A counter of size $\log_2\{\ln(bN) / \ln[b / (b - 1)]\}$ bits is sufficient.

Proof: We know that to store value x we need at most $\log_2 x$ bits. Hence, the proof of this theorem follows for the Theorem 3.

Choosing the correct value of b

There are three constraints to consider when choosing b.

- *Lower bound derived from DRAM access time.* The DRAM access time is $A_t = Pb/2R$. Therefore, if the DRAM supports a random access time T_R, we require $Pb/2R \geq T_R$. Hence, $b \geq 2RT_R/P$, which gives a lower bound on b.
- *Lower bound derived from memory I/O bandwidth.* Let DRAM I/O bandwidth be D. Every counter update operation is a read-modify-write, which takes $2M$ bits of bandwidth per update. Hence, $2RM/Pb \leq D$ or $b \geq 2RM/PD$. This gives a second lower bound on b.
- *Upper bound derived from counter size for LCF policy.* From Theorem 4, the size of the counters in SRAM is bounded by $\log_2\{\ln(bN) / \ln[b / (b - 1)]\}$. However, since our goal is to keep only a small-size counter in SRAM, we require that $\log_2\{\ln(bN) / \ln[b / (b - 1)]\} < M$. This gives us an upper bound on b.

The system designer can choose any value of b that satisfies these three bounds. Very large N and small M can have no suitable value of b. Such a case forces the system designer to store all the counters in SRAM.

OC-192 line card counter design

Consider an OC-192c line card that maintains a million counters. Assume that the maximum size of a counter is $P = 64$ bytes and that each arriving packet updates a maximum of $C = 10$ counters, hence, $R = 100$ Gbps. Suppose that the fastest available DRAM has access time $T_R = 51.2$ ns. Since our approach requires $Pb/2R \geq T_R$, this means that $b \geq 20$. Given present DRAM technology, this is sufficient to meet the lower bound obtained on b using the memory I/O bandwidth constraint. Hence the lower bound on b is simply $b \geq 20$.

We consider the upper bound on b, using two different values for counter size M, required in the system. If $M = 64$, then $\log_2\{\ln(bN) / \ln[b / (b - 1)]\} < M$ and we design the counter architecture with $b = 20$. We find that 9 bits is the minimum size for the SRAM counters, as required for the LCF policy. This counter size results in a 9-Mbyte SRAM. Keeping the SRAM memory on-chip supports the required access rate. If $M = 8$, then $\forall b, b \geq 20, \log_2\{\ln(bN) / \ln[b / (b - 1)]\} > M$. Thus there is no optimal value of b, and this design must store all the counters in SRAM without using DRAM.

Packet switches need to maintain counters for gathering statistics on various events. Our method can help build a high-bandwidth statistics counter for any pattern of arrival traffic. We discussed the necessary condition on the size of SRAM required to keep exact statistics under any policy. LCF CMA policy is optimal in the sense of smallest SRAM size, and we obtained the bounds on the size of SRAM. But LCF CMA is a complex algorithm to implement at a very high speed. It will be interesting to obtain a similar performance as LCF CMA with a less complex algorithm. MICRO

Acknowledgments

This research was supported by the National Science Foundation under Next Generation Internet (NGI) contract ANI-9872761, the NSF Career award, the Alfred P. Sloan Foundation, and the Terman Fellowship.

References

1. M. Sanchez, E. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, vol. 15, no. 2, 2001, pp. 8-23.
2. P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network*, vol. 15, no. 2, 2001, pp. 24-32.
3. M. Arpaci and J. Copeland, "Buffer Management for Shared-Memory ATM Switches," *IEEE Comm. Surveys and Tutorials*; http://www.comsoc.org/livepubs/surveys/public/1q00issue/copeland.html.
4. S. Iyer, R.R. Kompella, and N. McKeown, "Analysis of a Memory Architecture for Fast Packet Buffers," *Proc. IEEE Workshop High Performance Switching and Routing* (HPSR), IEEE, Piscataway, N.J., 2001.
5. M.L. Irland, "Buffer Management in a Packet Switch," *IEEE Trans. Comm.*, vol. COM-26, no. 3, Mar. 1978, pp. 328-337.
6. R. Geurin and V. Peris, "Quality-of-Service in Packet Networks: Basic Mechanisms and Directions," *Computer Networks*, vol. 31, no. 3, Feb. 1999, pp. 169-189.
7. D. Shah et al., "Analysis of a Statistics Counter Architecture," *Proc. IEEE Hot Interconnects 9*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 107-111.

Devavrat Shah is a PhD candidate in the Department of Computer Science at Stanford University. His research interests include the design, analysis, and implementation of network algorithms. Shah has a bachelor's degree in computer science and engineering from the Indian Institute of Technology, Bombay.

Sundar Iyer is a doctoral candidate in the Computer Science Department at Stanford University, where he works in the high-performance networking group. His research interests include load-balancing algorithms for network design with an emphasis on packet switching, packet buffer design, packet classification, and routing. Iyer has a BTech in computer science from the Indian Institute of Technology, Bombay and an MS in computer science from Stanford University. He received the Christofer Stephenson award for the best Master's thesis in computer science at Stanford University.

Balaji Prabhakar is an assistant professor of electrical engineering and computer science and a Terman Fellow at Stanford University. He is also a fellow of the Alfred P. Sloan Foundation. His research interests include network algorithms (especially for switching, routing and quality of service), wireless networks, Web caching, network pricing, information theory, and stochastic network theory. Prabhakar has a PhD from the University of California at Los Angeles. He has received a Career award from the National Science Foundation and the Erlang Prize from the Informs Applied Probability Society.

Nick McKeown is a professor of electrical engineering and computer science at Stanford University. His research interests include the architecture, analysis, and design of high-performance switches and Internet routers, IP lookup and classification algorithms, scheduling algorithms, Internet traffic analysis, traffic modeling, and network processors. McKeown has a PhD from the University of California at Berkeley in electrical engineering and computer sciences. He is the Robert Noyce Faculty Fellow at Stanford, a Fellow of the Alfred P. Sloan Foundation, and recipient of a Career award from the National Science Foundation. He received the 2000 IEEE Rice Award for the best paper in communications theory. He is a senior member of the IEEE and serves as an editor for the *IEEE Transactions on Communications* and *ACM/IEEE Transactions on Networking*.

Direct questions or comments about this article to Sundar Iyer, Computer Systems Laboratory, Stanford University, Stanford, Calif., 94305-9030; sundaes@stanford.edu.

For further information on this or any other computing topic, visit our Digital Library at http://computer.org/publications/dlib.