Analysis of a Statistics Counter Architecture*

Devavrat Shah, Sundar Iyer, Balaji Prabhakar, Nick McKeown
Computer Systems Laboratory, Stanford University
Stanford, CA 94305-9030
{devavrat, sundaes, balaji, nickm}@stanford.edu

Abstract -- **Packet switches (e.g., IP routers, ATM switches and Ethernet switches) maintain statistics for a variety of reasons: performance monitoring, network management, security, network tracing, and traffic engineering. The statistics are usually collected by counters which might, for example, count the number of arrivals of a specific type of packet, or count particular events, such as when a packet is dropped. The arrival of a packet may lead to several different statistics counters being updated. The number of statistics counters and the rate at which they are updated is often limited by memory technology. A small number of counters may be held in on-chip registers or in (on- or off-chip) SRAM. But often, the number of counters is very large, and hence they need to be stored in off-chip DRAM. However, the large random access times of DRAMs make it difficult to support high bandwidth links. The time taken to read, update and write a single counter would be too large, and worse still multiple counters may need to be updated for each arriving packet. In this paper we consider a specific architecture for storing and updating statistics counters. Smaller sized counters are maintained in fast (potentially on-chip) SRAM, while a large, slower DRAM maintains the full-sized counters. The problem is to ensure that the counter values are always correctly maintained at line-rate. We describe and analyze an optimal counter management algorithm (*LCF-CMA*), which minimizes the size of the SRAM required while ensuring correct line-rate operation of a large number of counters.**

Keywords--**packet-switch, statistics, counters, largest counter first (*LCF*).**

I. INTRODUCTION

Packet switches perform many processing tasks on each arriving packet. These include address-lookup, classification, buffering, QoS scheduling, header editing and statistics maintenance. Each of these tasks are typically performed on the line cards of switches and routers, and therefore need to be done at line rate. With line rates expected to increase beyond OC192 (10Gb/s) to OC768 (40Gb/s), each of the above packet processing tasks becomes more difficult. There have been several different techniques proposed for address-lookup [1], packet-classification [2], packet buffering [3][4][5], and QoS scheduling [6]. We are not aware of papers that describe the problem of maintaining a large number of statistics counters.

Packet switches maintain statistics for many reasons. These include firewalling (especially stateful firewalling), intrusion detection, performance monitoring (e.g. RMON), network tracing, load balancing and traffic engineering (e.g. policing and shaping). In addition, most packet switches maintain statistics counters to facilitate network management.

The general problem of statistics maintenance can be characterized as follows: When a packet arrives, it is first classified to determine what actions will be performed on the packet. For example, whether the packet should be accepted or dropped, whether it should receive expedited service or not, and so on. Depending on the chosen action, some statistics counters are updated.

The statistics we are interested in here are those which count events. For example, the number of fragmented packets, the number of dropped packets, the total number of packets arrived, the total number of bytes forwarded etc. In the rest of this paper we shall refer to these as *counters*. It is our goal to study and quantitatively analyze the problem of maintaining these counters.

We are particularly interested in applications that maintain a large number of counters. For example, a routing table that keeps a count of how many times each prefix is used, or a router that keeps a count of packets belonging to each TCP connection. Both examples would require several hundreds of thousands, or even millions of counters to be maintained simultaneously, making it infeasible (or at least very costly) to store them in SRAM. Instead, it becomes necessary to store the counters in off-chip, relatively slow DRAM.

Furthermore, we are interested in applications in which updates are frequent. For example, an OC192c link in which multiple counters are updated upon each packet arrival. These read-modify-write operations must be conducted at the same rate as packets arrive.

If each counter is M bits wide, then a counter update operation is as follows: 1) Read the M bit value stored in the counter, 2) increment the M bit value, and 3) write the updated M bit value back. If packets arrive at a rate R Gb/s, the minimum packet size is P bits, and if we update C counters each time a packet arrives, the memory may need to be accessed (either read or written) every $P/2CR$ nanoseconds. Let's consider the example of 40byte TCP packets arriving on a 10Gb/s link, each leading to the updating of two counters. The memory needs to be accessed every 8ns, about eight times faster than the random-access speed of commercial DRAMs today.

It is a strict requirement that the counter(s) be correctly updated every time a packet arrives. No packet must be left unaccounted for. If we do an update operation every time a packet arrives and update C counters per packet, then the minimum bandwidth R_D required on the memory interface where the counters are stored would be at least $2RMC/P$. Again this can become unmanageable as the size of the counters and the line

*This research was supported by the National Science Foundation, under NGI contract ANI-9872761, the NSF CAREER award, the Alfred P. Sloan Foundation and the Terman Fellowship.

0-7695-1357-3/01 $10.00 © 2001 IEEE

rates increase.

In this paper, we propose an approach which uses DRAMs to maintain statistics counters and a small fixed amount of (possibly on-chip) SRAM. We assume that N counters of width M bits are to be stored in the DRAM, and that N counters of width $m < M$ bits are stored in SRAM. The counters in SRAM keep track of the number of updates not yet reflected in the DRAM counters. Periodically, under the control of a counter management algorithm (CMA), the DRAM counters are updated by adding to them the values in the SRAM counters, as shown in Figure 1. The basic idea is that by updating the DRAM counters relatively infrequently, the memory bandwidth requirements are reduced.

We are interested in deriving strict bounds on the size of the SRAM such that — irrespective of the arriving traffic pattern — none of the counters in the SRAM overflow and the access rate and the bandwidth requirements on the DRAM are decreased, while still ensuring correct operation of the counters.

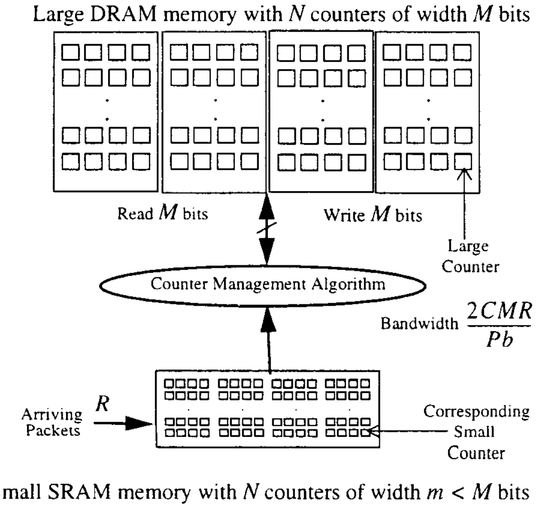
Large DRAM memory with N counters of width M bits
Read M bits
Write M bits
Large Counter
Counter Management Algorithm
Bandwidth $\frac{2CMR}{Pb}$
Arriving Packets
R
Corresponding Small Counter
Small SRAM memory with N counters of width $m < M$ bits

Figure 1: Memory hierarchy for the statistics counters. A fixed-sized ingress SRAM stores the small counters, which are periodically transferred to the large counters in DRAM. The memory bandwidth on the DRAM is decreased by a factor b

We will see that the size of the SRAM, and the access rate of the DRAM both depend on the CMA used. The main result of this paper is that there exists a CMA (which we call largest counter first, *LCF*) that minimizes the size of the SRAM. We derive necessary and sufficient conditions on the sizes of the counters (and hence of the SRAM which stores all these counters), and we prove that *LCF* is optimal. It is interesting to note that this problem has some similarities with a related problem in packet buffering [7].

As an example of how this technique can be used, consider an OC192c linecard 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 a statistics counter can be built with a DRAM of access time 51.2 ns, a DRAM memory bandwidth of 1.25 Gb/s and 9 Mb of SRAM.

II. Definitions

We will now describe the memory hierarchy used to hold the statistics counters, and in the following sections describe the *LCF-CMA* (Largest Counter First Counter Management Algorithm).

A. Memory Hierarchy

Definition 1: ***Minimum Packet Size,*** *P : Packets arriving at a switch have variable lengths. We shall denote by P the minimum length that a packet can have.*

Definition 2: ***Time Slot:*** *— The time taken to receive a minimum-sized packet at a link rate R.*

The SRAM is organized as a statically allocated memory, consisting of separate storage space for each of the N counters. We will assume from here-on that an arriving packet increments only one counter. If instead we wish to consider the case where C counters are updated per packet, we can consider the line rate on the interface to be CR.

Each counter is represented by a large counter of size M bits in the DRAM, and a small counter of size $m < M$ bits in SRAM. The small counter counts the most recent events, while the large counter counts events since the large counter was last updated. At any time instant the correct counter value is the sum of the small and large counters.

Updating a DRAM counter consists of a read-modify-write operation: 1) Read an M bit value from the large counter. 2) Add the m bit value of the corresponding small counter to the large counter, 3) Write the new M bit value of the large counter to DRAM, and 4) Reset the small counter value.

In this paper, we take it as a requirement to decrease the DRAM bandwidth by a factor b, i.e. $R_D = 2RM/Pb$ and increase the access time of the DRAMs accordingly, i.e. $A_t = Pb/2R$. Thus the CMA will update a large counter only once every b time slots. We will derive the minimum size of the SRAM as a function $g(.)$ and show that it is dependent on N, M and b. Thus the system designer is given a choice of trading off the SRAM size $g(N, M, b)$ with the DRAM bandwidth R_D and access time A_t.[1]

Definition 3: ***Count*** *$C(i, t)$: At time t, the number of times that the i^{th} small counter has been incremented since the i^{th} large counter was last updated.*

[1] The variable b $(b > 1)$ is chosen by the system designer. If $b = 1$, no SRAM is required, but the DRAM must be fast enough for all counters to be maintained in DRAM.

Definition 4: ***Empty Counter:*** *A counter i is said to be empty at time t if $C(i, t) = 0$.*

We note that the correct value of a large counter may be lost if the small counter is not added to the large counter in time, i.e. before an overflow of the small counter. Our goal is to find the smallest sized counters in the SRAM, and a suitable CMA, such that a small counter cannot overflow before its corresponding large counter is updated.

III. Necessity conditions on any CMA

Theorem 1:(Necessity) Under any CMA, a counter can reach a count $C(i, t)$ of $\dfrac{\ln[(N-1)(b/(b-1))^{(b-1)}]}{\ln(b/(b-1))}$.

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 b^{th} time slot one of the large counters is updated, and the corresponding small counter reset to 0. So 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 incremented once more, and N/b^2 of these counters are now used to update the large counter and reset. So after $2N$ time slots there are $[N(b-1)/b] + [N(b-1)/b^2]$ counters which 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 and 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 empty counters decreased to $2N/b^2$ from N/b (if $b = 2$, there is no change). In this way, after some time, we can have $N-1$ counters of count 2.

By continuing this argument, we can arrange for all $N-1$ counters to have a count $b-1$. Let us 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 and reset to 0, while the rest have a count of two. In the next $N-1$ time slots each of the counters with size 2 is incremented and again $1/b$ are served and reset to 0, 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 non-empty 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 $\dfrac{\ln[(N-1)(b/(b-1))^{(b-1)}]}{\ln(b/(b-1))}$. ❒

IV. A CMA which minimizes the size of the SRAM.

A. LCF-CMA

Algorithm Description: Every b time slots, *LCF-CMA* selects the counter i which has the largest count. If multiple counters have the same count, *LCF-CMA* picks one arbitrarily. *LCF-CMA* updates the value of the corresponding counter i in the DRAM and sets $(i, t) = 0$ in the SRAM.

B. Optimality

Definition 5: ***Domination****: Let $v = (v_1, v_2, \ldots v_N)$, and $= (u_1, u_2, \ldots u_N)$ denote the values of (i, t) for two different systems of N counters at any time t. Let π, σ be an ordering of the counters $(1, 2, 3, \ldots N)$ such that they are in descending order i.e. for v we have, $v_{\pi(1)} \geq v_{\pi(2)} \geq v_{\pi(3)} \geq \ldots \geq v_{\pi(N)}$ and for u we have $u_{\sigma(1)} \geq u_{\sigma(2)} \geq u_{\sigma(3)} \geq \ldots \geq u_{\sigma(N)}$.*

We say that, v dominates u denoted $v \gg u$, if $v_{\pi(i)} \geq u_{\sigma(i)}, \forall i$. Every arrival can possibly increment any of N different counters. The set of all possible arrivals patterns at time t can be defined as: $\Omega_t = \{(w_1, w_2, w_3, \ldots, w_t), 1 \leq w_i \leq N, \forall i\}$.

Theorem 2:(Optimality of LCF-CMA) Under arrival sequence $a(t) = (a_1, a_2, a_3, \ldots, a_t)$, let $q(a(t), P_C) = (q_1, q_2, q_3, \ldots, q_N)$ denote the count $C(i, t)$ of N counters at time t under service policy P_C. For any service policy P, there exists a $1-1$ function $f^t_{P.LCF}: (\Omega_t \rightarrow \Omega_t)$, for any t such that, $q\left(f^t_{P,LCF}(w), P\right) \gg q(w, LCF), \forall(w \in \Omega_t), \forall t$

Proof: See Appendix I. ❒

C. Sufficiency conditions on LCF service policy

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

Proof: *(By Induction)* Let $= b/(b-1)$. Let $_i(t)$ denote the number of counters with count i at time at time t. We define:

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

We claim that under *LCF* policy, $F(t) \leq bN$ 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, $F(t) \leq bN$. For the next b time slots, some b counters with count $i_1 \geq i_2 \geq \ldots \geq i_b$ are incremented. Even though not required for the proof, we assume that the counter values are distinct for simplicity. After the counters are incremented they have counts $i_1 + 1, i_2 + 1, \ldots, i_b + 1$ respectively, and the largest counter amongst all the N counters is serviced. The largest counter has at least a value $(.) \geq i_1 + 1$.

- ***Case 1***: If all the counter values at time t were nonzero, then the contribution of these b counters in $F(t)$ was:

  $$\alpha = d^{i_1} + d^{i_2} + \ldots d^{i_b}$$

  After considering the values of these counters after they are incremented, their contribution to $F(t+b)$ becomes $d\alpha$. But a counter with a count $C(.) \geq i_1 + 1$ 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; i.e. if arrivals occur to non-zero queues, $F(t)$ can not increase.
- ***Case 2***: Now we deal with the case when one or more counters at time t were zero. For simplicity assume all b counters which are incremented 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 amongst 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, $d^{i^*} = b$; otherwise 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. One can argue the case when arrivals occur to fewer than b empty counters similarly.

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

Theorem 4:(Sufficiency) A counter of size $\log_2\left(\frac{\ln bN}{\ln(b/(b-1))}\right)$ *bits is sufficient.*

Proof: We know that in order to store a value x we need at most $\log_2 x$ bits. Hence the proof follows from Theorem 3. □

D. Choosing the correct value of b.

We can see from Theorem 4 that the size of the counters in SRAM is bounded by $\log_2\left(\frac{\ln bN}{\ln(b/(b-1))}\right)$. However, since our goal is to keep only a small sized counter in the SRAM we need that $\log_2\left(\frac{\ln bN}{\ln(b/(b-1))}\right) < M$.

This gives us an upper bound on b. Also, note that the access time on the DRAM is now $A_t = Pb/2R$. Hence, if the DRAM technology being used supports a random access time T_R, we need $Pb/2R \geq T_R$. This gives us a lower bound on b. Similarly, if the DRAM bandwidth is also a constraint, another lower bound on b is obtained from the fact that $2RM/Pb$ must be less than the maximum bandwidth available from a DRAM subsystem. The system designer can choose any value of b between the lower and upper bounds, and derive a corresponding value for the SRAM size.

We note that for very large values of N and small values of M, it is possible that there is no suitable value of b; i.e., one cannot optimize the system using the above technique. In such a case, the system designer is forced to store all the counters in SRAM.

As an example, consider an OC192c linecard. Say that it 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 the actual rate arriving rate can be considered to be $R = 100$ Gb/s. Suppose that the fastest available DRAM has an access time of $T_R = 51.2$ ns. Since we require $Pb/2R \geq T_R$, this means that $b \geq 20$.

We will now consider two variants on the requirement of the size of the counter needed in the system.

1. If $M = 64$, then $\log_2\left(\frac{\ln bN}{\ln(b/(b-1))}\right) < M$ and we design the counter architecture with $b = 20$. We get that the minimum size of the counters in SRAM required for the *LCF* policy is 9 bits and this results in an SRAM of size 9Mb. The required access rate can be supported by keeping the SRAM memory on-chip.

2. If we require $M = 8$ then we can see that $\forall b, b \geq 20, \log_2\left(\frac{\ln bN}{\ln(b/(b-1))}\right) > M$. Thus there

is no optimal value of b and all the counters are always stored in SRAM without any DRAM.

V. Conclusions

Packet switches need to maintain counters for gathering statistics on various events. The general method presented in this paper can be used to build a high bandwidth statistics counter for any arrival traffic pattern. An algorithm, called largest counter first (*LCF*), was introduced for doing this and was shown to be optimal in the sense that it only requires a small optimally-sized SRAM, running at line rate, which temporarily stores the counters, and a DRAM running at slower than the line rate to store complete counters. For example, a statistics update arrival rate of 100 Gb/s on 1 million counters can be supported with currently available DRAMs (having a random access time of 51.2 ns) and 9 Mb of SRAM.

References

[1] M. Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, pp. 8-23, vol: 15:2, 2001.

[2] P. Gupta, and N. McKeown, "Algorithms for packet classification," *IEEE Network*, pp. 24-32, vol: 15:2, 2001.

[3] M. Arpaci, and J. Copeland, "Buffer Management for shared-memory ATM switches," *IEEE Comm. Surveys and Tutorials*, available at http://www.comsoc.org/pubs/surveys/1q00issue/copeland.html

[4] S. Iyer, R. R. Kompella, and N. McKeown, "Analysis of a memory architecture for fast packet buffers," *In Proc. IEEE HPSR*, Dallas, Texas, 2001.

[5] M. L. Irland, "Buffer Management in a Packet Switch," *IEEE Trans. Communication*, vol. COM-26:3, pp. 328-337, Mar. 1978.

[6] R. Geurin, and V. Peris, "Quality-of-Service in packet networks: Basic mechanisms and directions," *Computer Networks*, 31:3, pp. 169-189, Feb. 1999.

[7] H. Gail, G. Grover, R. Guerin, S. Hantler, Z. Rosberg, and M. Sidi, "Buffer size requirements under longest queue first," *Proceedings IFIP'92*, vol. C-5, pp. 413-24, 1992.

APPENDIX A

Theorem 2:(Optimality of LCF-CMA). Under arrival sequence $a(t) = (a_1, a_2, a_3, \ldots, a_t)$, *let* $(a(t), P_C) = (q_1, q_2, q_3, \ldots, q_N)$ *denote the count* $C(i, t)$ *of* N *counters at time* t *under service policy* P_C. *For any service policy* P, *there exists a* $1-1$ *function* $f^t_{P.LCF}: (\Omega_t \rightarrow \Omega_t)$, *for any* t *such that,* $q\left(f^t_{P.LCF}(w), P\right) \gg q(w, LCF), \forall (w \in \Omega_t), \forall t$.

Proof: We prove the existence of such a function $f^t_{P.LCF}$ inductively over time t. Let us denote the counters of the *LCF* system by $(l_1, l_2, \ldots, l_N)$ and the counters of the P system by $(p_1, p_2, \ldots, p_N)$. It is trivial to check that there exists such a function for $t = 1$. Inductively assume that $f^t_{P,LCF}$ exists with the desired property till time t, and we want to extend it to time $t+1$. This means that there exists ordering π^t, σ^t such that, $l_{\pi^t(i)} \le p_{\sigma^t(i)}, \forall i$. Now, at the time $t+1$, a counter may be incremented and a counter may be completely served. We consider both these parts separately below:

- ***Part 1*: (Arrival)** Let a counter be incremented at time $t+1$ in both systems. Suppose that counter $\pi^t(k)$ is incremented in the *LCF* system. Then extend $f^t_{P.LCF}$ for $t+1$ by letting an arrival occur in counter $\sigma^t(k)$ for the P system. By induction, we have $l_{\pi^t(i)} \le p_{\sigma^t(i)}, \forall i$. Let π^{t+1}, σ^{t+1} be the new ordering of the counters of the *LCF* and P systems respectively. Since one arrival occurred to both the systems in a queue with the same relative order, the domination relation does not change.
- ***Part 2*: (Service)** Let one of the counters be served at time $t+1$. Under the *LCF* policy, the counter $\pi^t(1)$ with count $l_{\pi^t(1)}$ will be served and its count is set to zero i.e. $C(\pi^t(1), t+1) = 0$, while under P any queue can be served out, depending on the CMA prescribed by P. Let P serve the counter with rank k, i.e. counter $\sigma^t(k)$. Then we can create a new ordering π^{t+1}, σ^{t+1} as follows:

a. $\pi^{t+1}(i) = \pi^t(i+1), 1 \le i \le N-1$,
$\pi^{t+1}(N) = \pi^t(1)$

b. $\sigma^{t+1}(i) = \sigma^t(i), 1 \le i \le k-1$,
$\sigma^{t+1}(i) = \sigma^t(i+1), k \le i \le N-1$,
$\sigma^{t+1}(N) = \sigma^t(k)$

Under this definition, it is easy to check that, $l_{\pi^{t+1}(i)} \le p_{\sigma^{t+1}(i)}, \forall i$ given $l_{\pi^t(i)} \le p_{\sigma^t(i)}, \forall i$. Thus we have shown explicitly how we can extend $f^t_{P.LCF}$ to $f^{t+1}_{P.LCF}$ with the desired property. Hence it follows inductively that *LCF* is dominated by any other policy P. ❐