



$$R_{ij}[k_0; k_1] = \sum_{k=k_0}^{k_1-1} r_{ij}[k], \quad k_1 > k_0. \quad (5)$$

The round-trip time equals  $\tau$  time slots, where a time slot equals the duration of one packet, and we assume symmetric forward and reverse paths and that  $\tau$  is even. Let  $\delta = \tau/2$ .

The pending request counters  $C_{ij}$  are updated as follows:

$$C_{ij}[k+1] = C_{ij}[k] + r_{ij}[k-\delta] - g_{ij}[k]. \quad (6)$$

By recursively applying (6) combined with (3) and (5), we obtain

$$C_{ij}[k_1] = C_{ij}[k_0] + R_{ij}[k_0 - \delta; k_1 - \delta] - G_{ij}[k_0; k_1]. \quad (7)$$

In general,  $L_{ij}[k]$  and  $C_{ij}[k]$  will not be equal even in the absence of errors, because requests and grants may be in flight. To take this into account, we define consistency<sup>1</sup> as follows:

*Definition 1 (Consistency):* We define the state of  $L_{ij}$  and  $C_{ij}$  to be consistent at time slot  $k$  if and only if (8) holds:

$$L_{ij}[k] = C_{ij}[k] + R_{ij}[k - \delta; k] + G_{ij}[k - \delta; k]. \quad (8)$$

We can only determine whether this equality holds by counting the requests  $R_{ij}[k - \delta; k]$  and the grants  $G_{ij}[k - \delta; k]$ . In Sec. III we describe a protocol that achieves exactly this.

### III. CENSUS PROTOCOL

In a more general context, this problem is also known as obtaining a *snapshot* [5] of a distributed system to determine whether its global state is consistent. Our census protocol is related to the reliability mechanisms used in hop-by-hop credit flow control, as proposed in [6]. The *marker* used there is akin to our census in that it travels the RT. The main difference is that their scheme accounts for outstanding credits, whereas our scheme accounts for outstanding requests, which leads to different marker/census updating rules, as described below. For notational convenience, we define  $k_n = k_{n-1} + \delta$ ,  $n \geq 1$ .

#### A. Protocol description

Line card  $i$  injects a census message for  $Q_{ij}$  on the control channel to the arbiter at time  $k_1$ , comprising a census count  $\Delta$ , which is initialized to the current length of  $Q_{ij}$ :

$$\Delta[k_1] = L_{ij}[k_1]. \quad (9)$$

Simultaneously, we reset a counter  $H_{ij}[k]$ ,  $H_{ij}[k_1] = 0$ , to count the number of grants for  $Q_{ij}$  received since the census was injected, i.e.,  $H_{ij}$  is incremented for every grant received for  $Q_{ij}$ . It follows that for  $k \geq k_1$  (see also Fig. 2),

$$\begin{aligned} H_{ij}[k] &= \sum_{k'=k_1}^{k-1} g_{ij}[k' - \delta] = G_{ij}[k_1 - \delta; k - \delta] \\ &= G_{ij}[k_0; k - \delta]. \end{aligned} \quad (10)$$

When the census arrives at the arbiter at time slot  $k_2$ , the arbiter modifies the census count by subtracting  $C_{ij}[k_2]$ :

$$\Delta[k_2] = \Delta[k_1] - C_{ij}[k_2]. \quad (11)$$

By substituting (6) and (9) into (11), we obtain

$$\Delta[k_2] = L_{ij}[k_1] - C_{ij}[k_1] - R_{ij}[k_0; k_1] + G_{ij}[k_1; k_2]. \quad (12)$$

The arbiter then returns the census to line card  $i$ . When it arrives there at time slot  $k_3$ , the line card updates the census count by subtracting  $H_{ij}[k_3]$ .

$$\Delta[k_3] = \Delta[k_2] - H_{ij}[k_3]. \quad (13)$$

Combining (12), (10), and (4) we obtain

$$\Delta[k_3] = L_{ij}[k_1] - C_{ij}[k_1] - R_{ij}[k_0; k_1] - G_{ij}[k_0; k_1]. \quad (14)$$

We are now ready to prove the main theorem.

#### B. Census check

*Theorem 1 (Census):* Using the above protocol, the states of  $L_{ij}$  and  $C_{ij}$  are consistent according to Definition 1 at time slot  $k_1$  if and only if  $\Delta[k_3] = 0$ , with  $k_3 = k_1 + \tau$ .

*Proof:* The proof is straightforward. Assume that  $L_{ij}$  and  $C_{ij}$  are consistent at time slot  $k_1$ . Substituting (8) with  $k = k_1$  in (14) and noting that  $k_0 = k_1 - \delta$ , we obtain

$$\begin{aligned} \Delta[k_3] &= C_{ij}[k_1] + R_{ij}[k_1 - \delta; k_1] + G_{ij}[k_1 - \delta; k_1] \\ &\quad - C_{ij}[k_1] - R_{ij}[k_0; k_1] - G_{ij}[k_0; k_1] \\ &= 0. \end{aligned}$$

Conversely, assume that  $\Delta[k_3] = 0$ . Then by (14) we immediately obtain

$$0 = L_{ij}[k_1] - C_{ij}[k_1] - R_{ij}[k_0; k_1] - G_{ij}[k_0; k_1]$$

and equivalently

$$L_{ij}[k_1] = C_{ij}[k_1] - R_{ij}[k_0; k_1] - G_{ij}[k_0; k_1],$$

which by Definition 1 implies that  $L_{ij}$  and  $C_{ij}$  are consistent at time slot  $k_1$ . ■

### IV. ERROR DETECTION

We now introduce errors in the system. The most likely source of errors are bit errors introduced on the transmission links, which translate into corruption and therefore loss of requests and grants. Less likely, but still possible, are errors in the hardware that maintains and updates  $L_{ij}$  and  $C_{ij}$ .

In principle, every term in (14) can be in error. However, because we need a point of reference, we assume that  $L_{ij}$  is always correct.<sup>2</sup> We denote potentially erroneous terms with tildes:

$$\begin{aligned} \Delta[k_3] &= L_{ij}[k_1] - \tilde{C}_{ij}[k_1] - \tilde{R}_{ij}[k_0; k_1] \\ &\quad + \tilde{G}_{ij}[k_1; k_2] - \tilde{G}_{ij}[k_0; k_2]. \end{aligned} \quad (15)$$

We rewrite (15) by lumping all errors into a single, separate error term  $\varepsilon$ :

$$\Delta[k_3] = L_{ij}[k_1] - C_{ij}[k_1] - R_{ij}[k_0; k_1] - G_{ij}[k_0; k_1] + \varepsilon. \quad (16)$$

<sup>1</sup>In distributed computing, the appropriate term is *coherence*, as consistency concerns temporal instruction ordering [4, pp. 549–551].

<sup>2</sup>If  $L_{ij}$  is in error, the system state may be consistent, but still incorrect. Such errors can only be dealt with at higher protocol levels.

This also implies that the values of the other terms in (16) are error-free and thus consistent. Substituting (8) into (16) yields

$$\Delta[k_3] = \varepsilon. \quad (17)$$

From Theorem 1 it follows immediately that we have consistency if and only if  $\varepsilon = 0$ . However, note that  $\varepsilon = 0$  does not imply that no errors have occurred—there may have been multiple errors with a net effect of zero, but as the result is still consistent, no action needs to be taken.

If  $\varepsilon \neq 0$  the magnitude of the inconsistency equals  $|\varepsilon|$ . If  $\varepsilon > 0$ ,  $L_{ij}[k_1]$  is larger than the sum of outstanding requests and grants, i.e.,  $|\varepsilon|$  requests are missing. This situation can be rectified by issuing  $|\varepsilon|$  extra requests. On the other hand, if  $\varepsilon < 0$ , then  $L_{ij}[k_1]$  is smaller than the sum of outstanding requests and grants, i.e., there are  $|\varepsilon|$  requests too many. Note that this situation ( $\varepsilon < 0$ ) will correct itself, because the spurious requests will eventually result in spurious grants that are dropped by the line card because it has no further packets for  $Q_{ij}$ . However, this may take a long time and introduces a loss of efficiency. To correct the inconsistency faster, the line card can either cancel  $|\varepsilon|$  future requests or issue a special control message to the arbiter, instructing it to cancel  $|\varepsilon|$  requests.

## V. PRACTICAL CONSIDERATIONS

### A. Counter size

As the arbiter comprises  $N \times M$  counters, their size has a significant impact on the total arbiter chip area. In principle, these counters only need to store one RT's worth of requests, although the VOQs at the line cards may be able to store many more packets. This can be solved by maintaining a pending request counter  $P_{ij}$  that keeps track of the number of unissued requests for  $Q_{ij}$ , i.e., the difference between  $L_{ij}$  and the number of requests issued. Let  $\hat{C}$  be the maximum value that the VOQ counters  $C_{ij}$  can represent, e.g., when these counters are implemented using  $n$ -bit registers, then  $\hat{C} = 2^n - 1$ .

If a new packet arrives at  $Q_{ij}$  and  $L_{ij} - P_{ij} \geq \hat{C}$ , the corresponding request is withheld and  $P_{ij}$  is incremented. The request function  $r_{ij}$  is redefined as follows:

$$r_{ij}[k] = \begin{cases} 1 & \text{if } (a_{ij}[k] + P_{ij}[k] > 0) \\ & \wedge (L_{ij}[k] - P_{ij}[k] < \hat{C}), \\ 0 & \text{otherwise,} \end{cases}$$

where  $a_{ij}[k]$  equals the number of packets arriving at  $Q_{ij}$  at time  $k$ . After computing  $r_{ij}[k]$ ,  $P_{ij}[k]$  is updated:

$$P_{ij}[k+1] = P_{ij}[k] + a_{ij}[k] - r_{ij}[k].$$

The only difference in the census protocol is in the definition of  $\Delta[k_1]$ :  $\Delta[k_1] = L_{ij}[k_1] - P_{ij}[k_1]$ .

### B. Overhead

A census for a specific VOQ introduces some overhead on the forward and the reverse control channel. A census message consists of two fields: a VOQ identifier of  $\lceil \log_2(N) \rceil$  bits and a census count. Note that the range of census-count values equals  $[-\hat{C}, B]$ , which can be encoded with  $1 + \lceil \log_2(\max(B, \hat{C})) \rceil$  bits, where  $B$  equals the maximum allowed VOQ length,  $\forall i, j, k: 0 \leq L_{ij}[k] \leq B$ .

### C. Census activation

We propose to use a timer-activated census per VOQ to ensure that every counter is checked regularly. The timer period should be sufficiently short to minimize the likelihood of an error within this period, but be sufficiently long so as not to introduce too much overhead. The overall system reliability and bit-error rate of the transmission channels play a key role in selecting this value. The timer triggers a census for one VOQ at a time, visiting all  $M$  VOQs in a round-robin fashion. A census may also be triggered by the explicit detection of an error, i.e., a failed parity or CRC check, or by an extended period of inactivity.

Note that, as pointed out in [6], two consecutive census messages for the same VOQ must be spaced by at least  $\tau$  time slots to avoid ambiguity.

### D. Robustness

Our protocol is extremely robust. The timer-based operation ensures that loss or corruption of a census message will be corrected by the next successful census. Requests or grants lost or duplicated during a census will either be caught by the current census (if they occur ahead of the census message) or by the next census (if they occur after the census message). The mechanism can even easily cope with the loss of a census message, as the next census will reset the (still running)  $H_{ij}$  counter.

## VI. CONCLUSION

We introduced a request-grant protocol for crossbar-based input-queued packet switches. The protocol exchanges incremental request and grant information between line card and arbiter, which reduces overhead and handles long round-trip times well in terms of performance. We also introduced a reliability mechanism on top of this protocol to ensure that the queue state information of the line cards and the arbiter remains consistent. We provided a formal proof that the proposed protocol detects any inconsistency, including its magnitude. It does not interfere with normal data or control traffic, is very robust, and entails acceptable overhead. Finally, the addition of a pending request counter on the line card reduces the complexity of the counters in the arbiter.

## REFERENCES

- [1] C. Minkenberg, R. Luijten, F. Abel, W. Denzel, and M. Gusat, "Current issues in packet switch design," *ACM Computer Commun. Rev.*, vol. 33, no. 1, pp. 119–124, Jan. 2003.
- [2] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, and I. Iliadis, "A four-terabit packet switch supporting long round-trip times," *IEEE Micro*, vol. 23, no. 1, pp. 10–24, Jan./Feb. 2003.
- [3] C. Minkenberg, "Performance of i-SLIP scheduling with large round-trip latency," in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2003*, Torino, Italy, June 24–27, 2003, pp. 49–54.
- [4] J. Hennessy and D. Patterson, *Computer Architecture - A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, Elsevier, May 2002.
- [5] K. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comp. Syst.*, vol. 3, no. 1, pp. 63–75, Feb. 1985.
- [6] C. Özveren, R. Simcoe, and G. Varghese, "Reliable and efficient hop-by-hop flow control," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 4, pp. 642–650, May 1993.