US006954829B2

(12) **United States Patent**
Beers et al.

(10) **Patent No.: US 6,954,829 B2**
(45) **Date of Patent: Oct. 11, 2005**

(54) **NON-SPECULATIVE DISTRIBUTED CONFLICT RESOLUTION FOR A CACHE COHERENCY PROTOCOL**

(75) Inventors: **Robert H. Beers**, Hillsboro, OR (US); **Herbert H. J. Hum**, Portland, OR (US); **James R. Goodman**, Madison, WI (US)

(73) Assignee: **Intel Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 463 days.

(21) Appl. No.: **10/326,232**

(22) Filed: **Dec. 19, 2002**

(65) **Prior Publication Data**

US 2004/0123052 A1 Jun. 24, 2004

(51) **Int. Cl.**[7] **G06F 12/00**
(52) **U.S. Cl.** **711/141**; 711/141; 709/216; 709/230; 709/238
(58) **Field of Search** 709/216, 230, 709/238; 711/141, 130

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,275,905 B1 8/2001 Keller et al.
6,640,287 B2 * 10/2003 Gharachorloo et al. 711/141

OTHER PUBLICATIONS

Jim Handy, "The Cache Memory Book," Second Edition, 1998, pp. vii–229, Academic Press Limited, London, United Kingdom.

"IEEE Standard for Scalable Coherent Interface (SCI)," IEEE Std 1596–1992, 1992, pp. i–243, Institute of Electrical and Electronics Engineers, Inc., USA.

"UltraSPARC User's Manual, UltraSPARC–I, UltraSPARC–II," Jul. 1997, pp. iii–394, Sun Microsystems, USA.

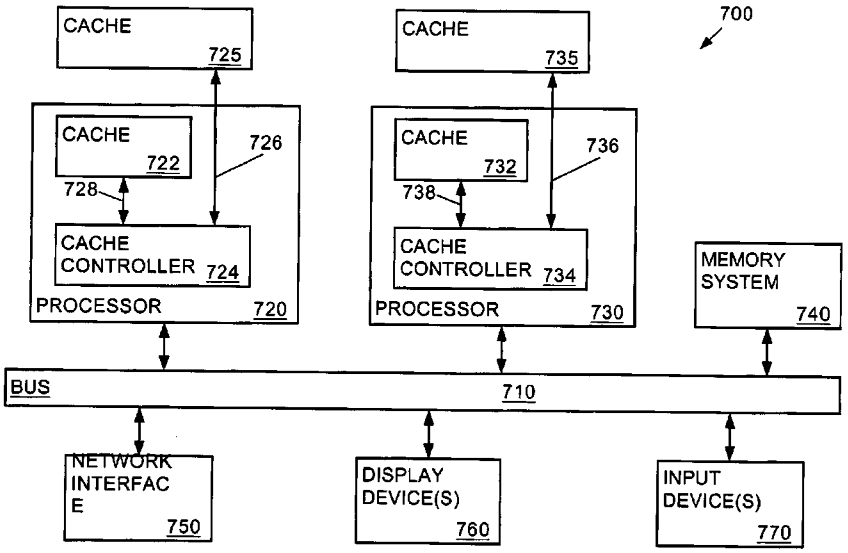
* cited by examiner

Primary Examiner—Jack A. Lane
(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor and Zafman

(57) **ABSTRACT**

A conflict resolution technique provides consistency such that all conflicts can be detected by at least one of the conflicting requestors if each node monitors all requests after that node has made its own request. If a line is in the Exclusive, Modified or Forward state, conflicts are resolved at the node holding the unique copy. The winner of the conflict resolution, and possibly the losers, report the conflict to the home node, which pairs conflict reports and issues forwarding instructions to assure that all requesting nodes eventually receive the requested data. If a requested cache line is either uncached or present only in the Shared state, the home node provides a copy of the cache node and resolves conflicts. In one embodiment, a blackout period after all responses until an acknowledgement message has been received allows all conflicting nodes to be aware of conflicts in which they are involved.

14 Claims, 24 Drawing Sheets



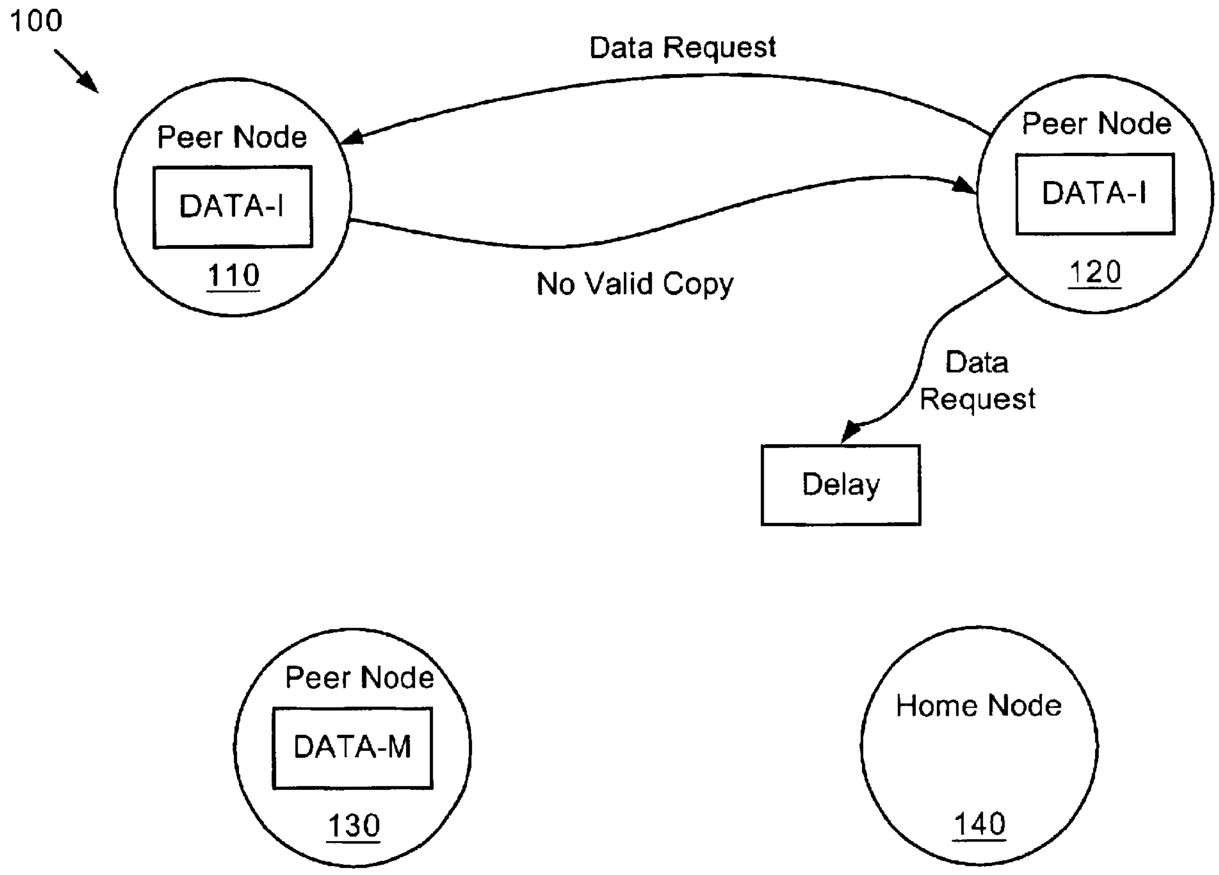


FIG. 1a
(Prior Art)

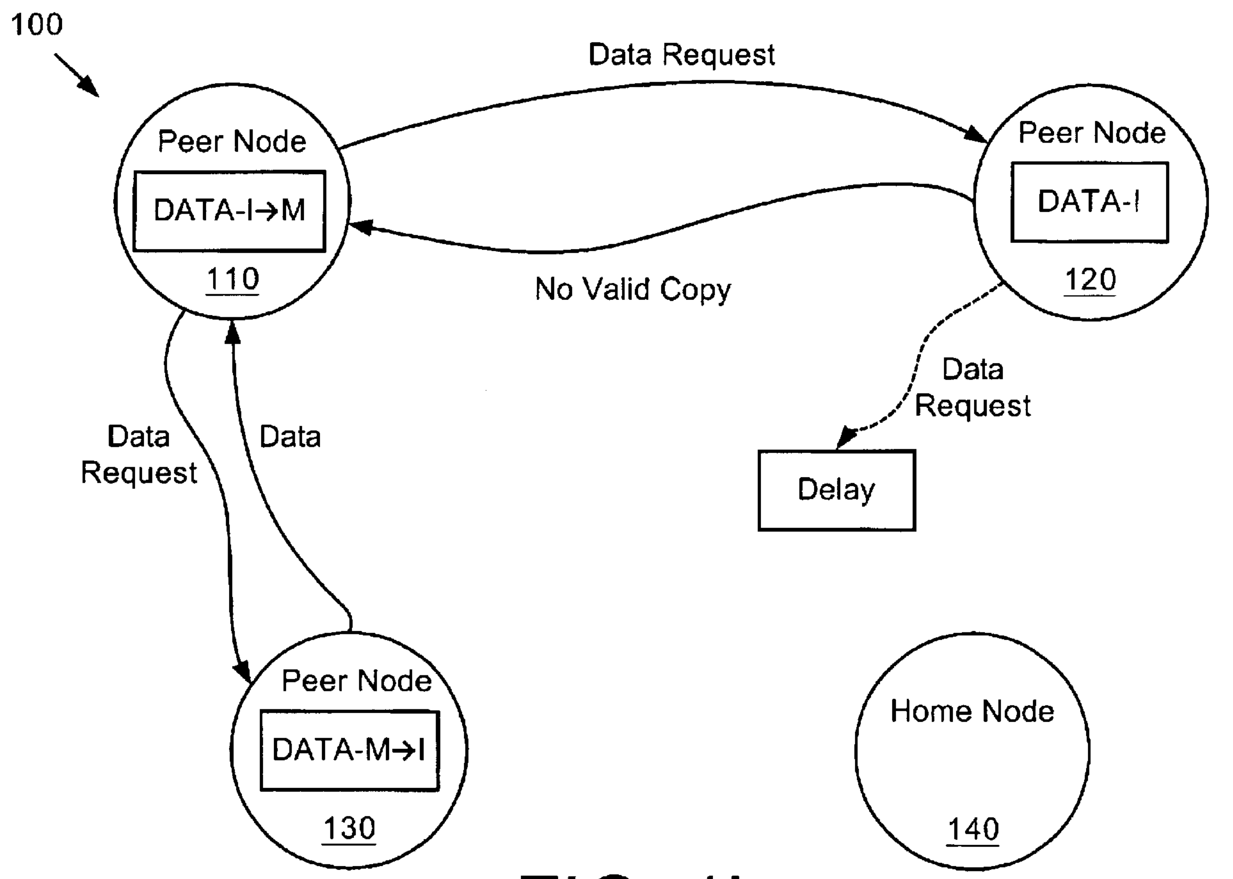


FIG. 1b
(Prior Art)

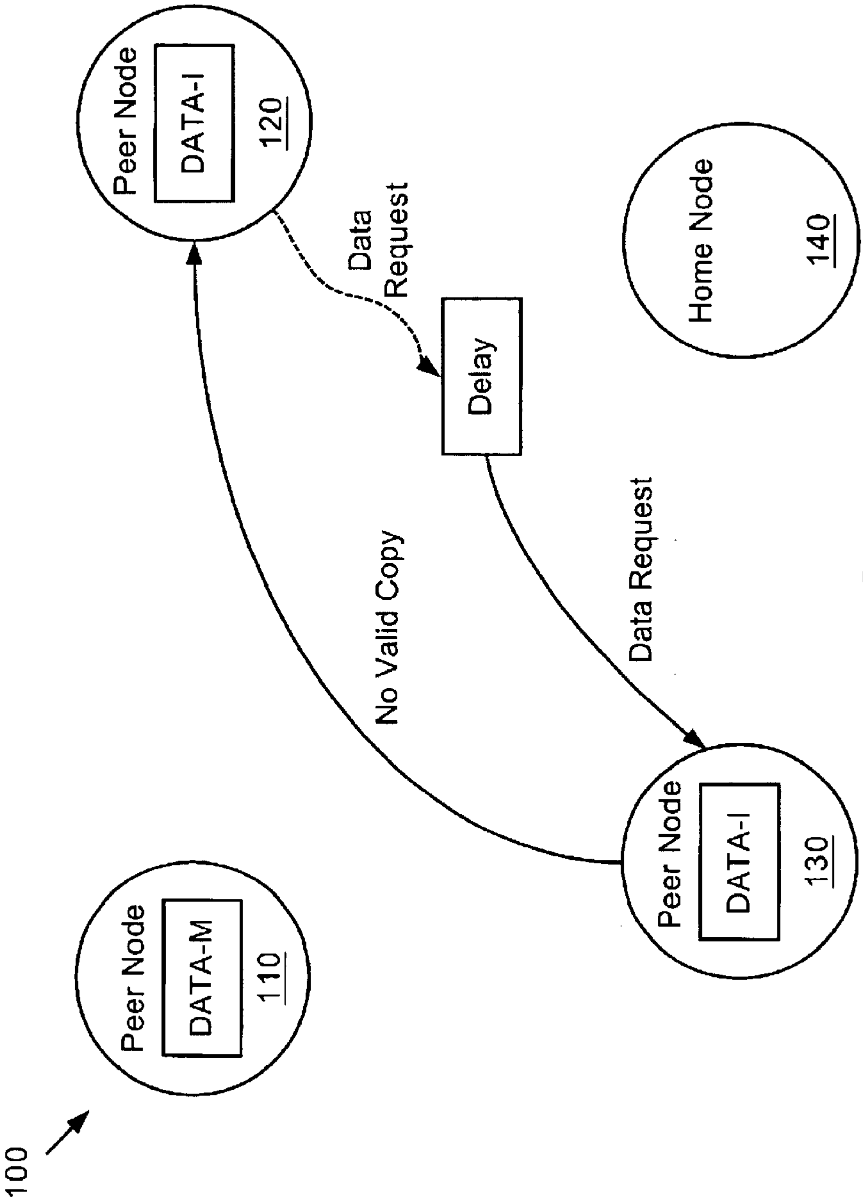


FIG. 1c
(Prior Art)

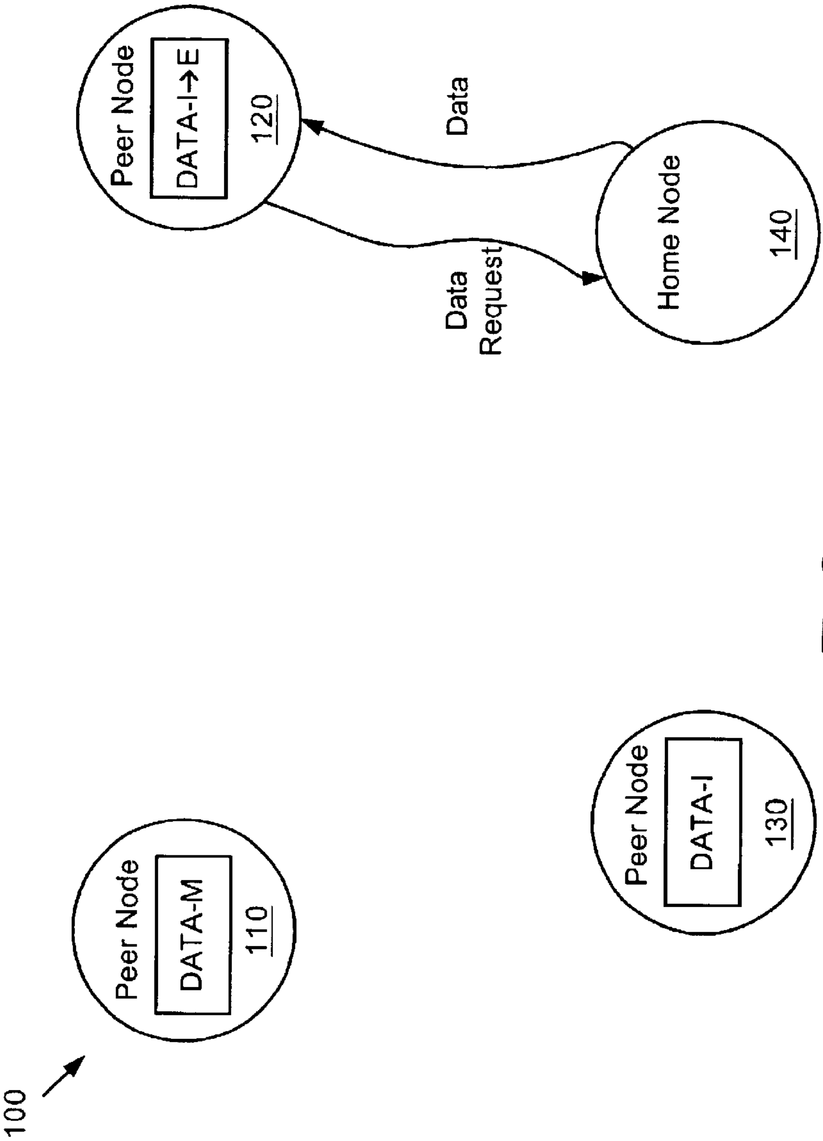


FIG. 1d
(Prior Art)

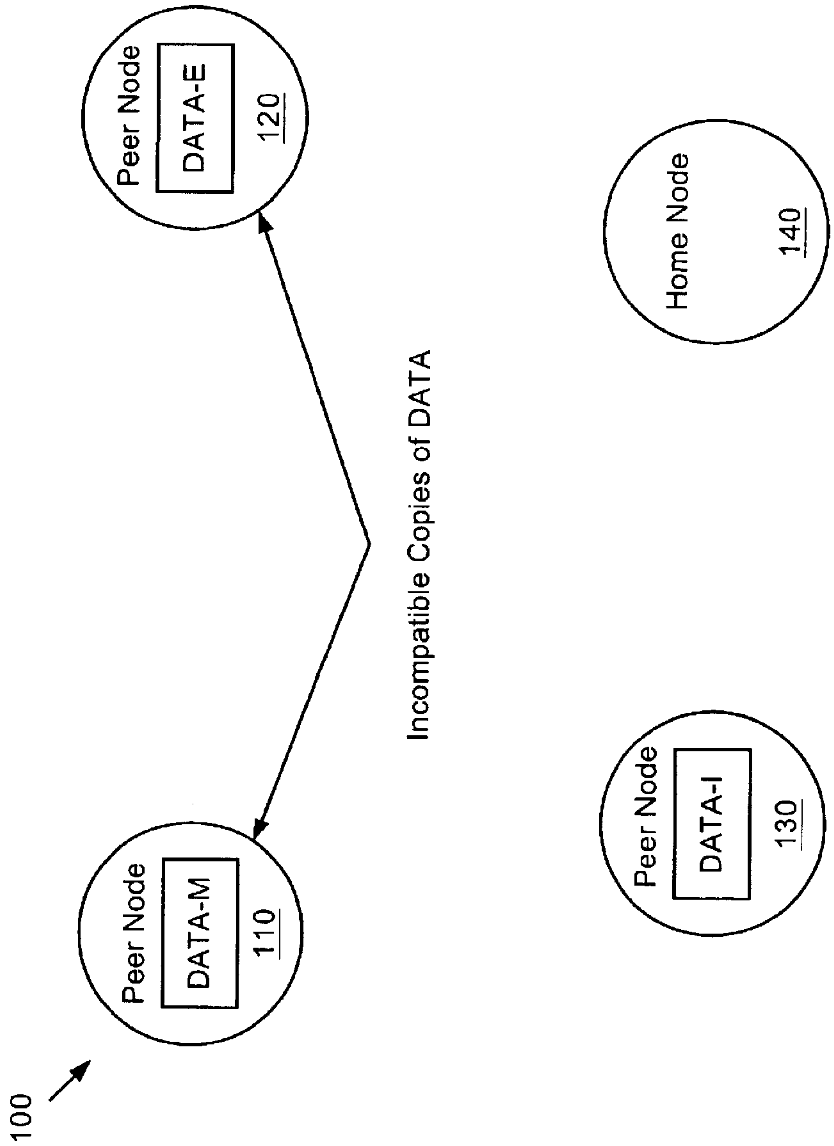


FIG. 1e
(Prior Art)

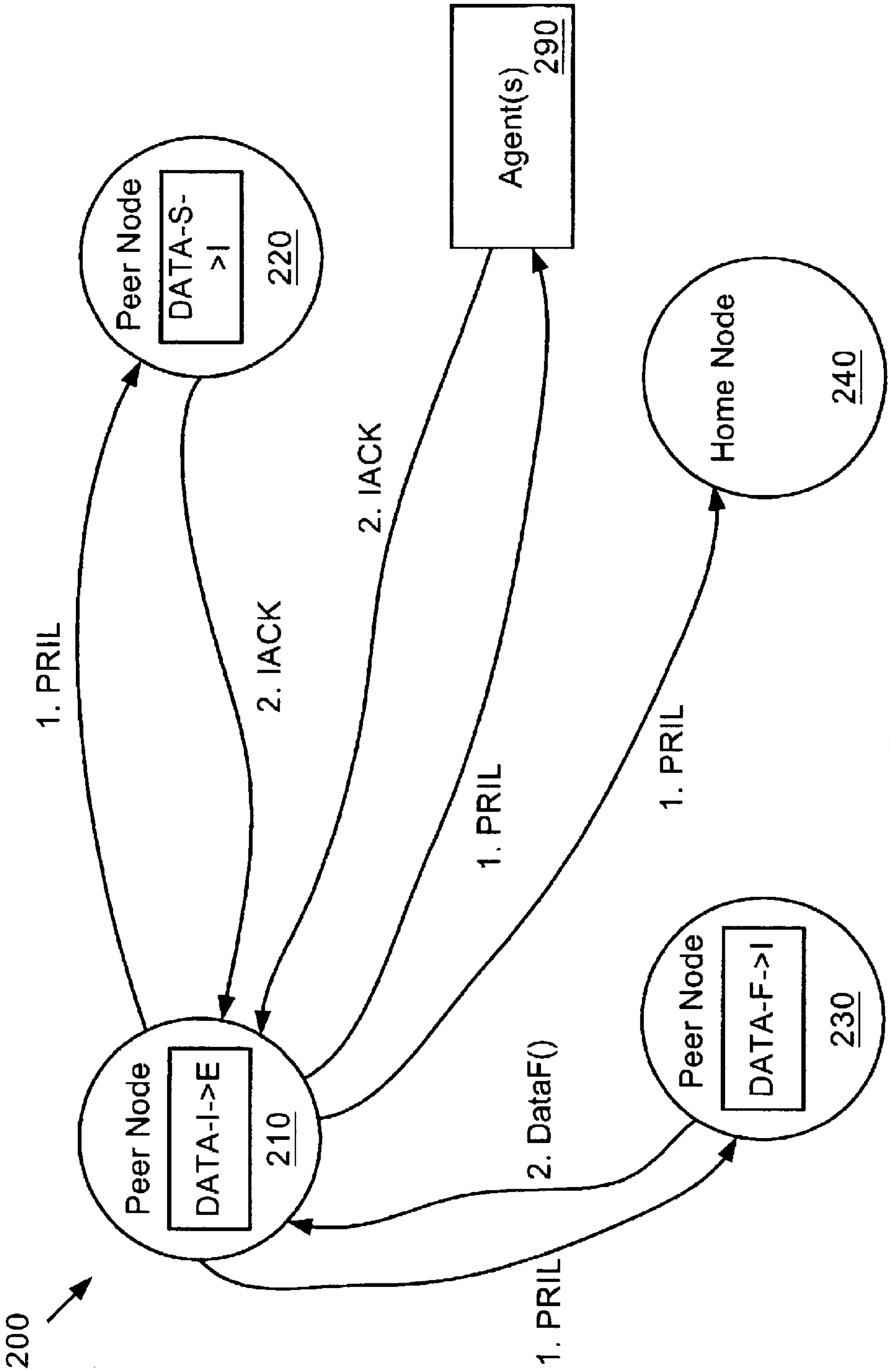

200
Peer Node
DATA-I->E
210
Peer Node
DATA-S->I
220
Peer Node
DATA-F->I
230
Home Node
240
Agent(s)
290
1. PRIL
2. IACK
2. IACK
1. PRIL
1. PRIL
2. DataF()
1. PRIL

FIG. 2a

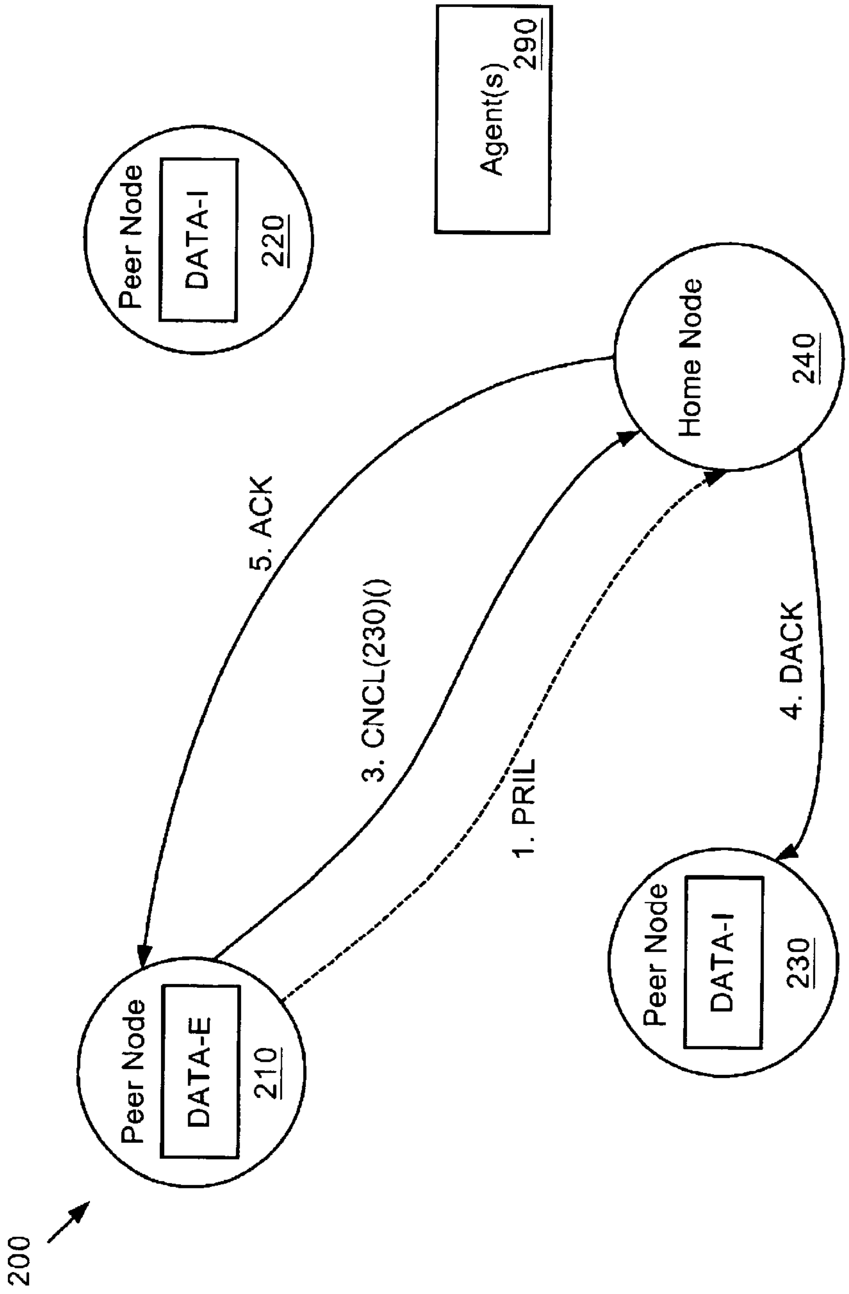

200
Peer Node
DATA-E
210
Peer Node
DATA-I
220
Peer Node
DATA-I
230
Home Node
240
Agent(s)
290
1. PRIL
3. CNCL(230)()
4. DACK
5. ACK

FIG. 2b

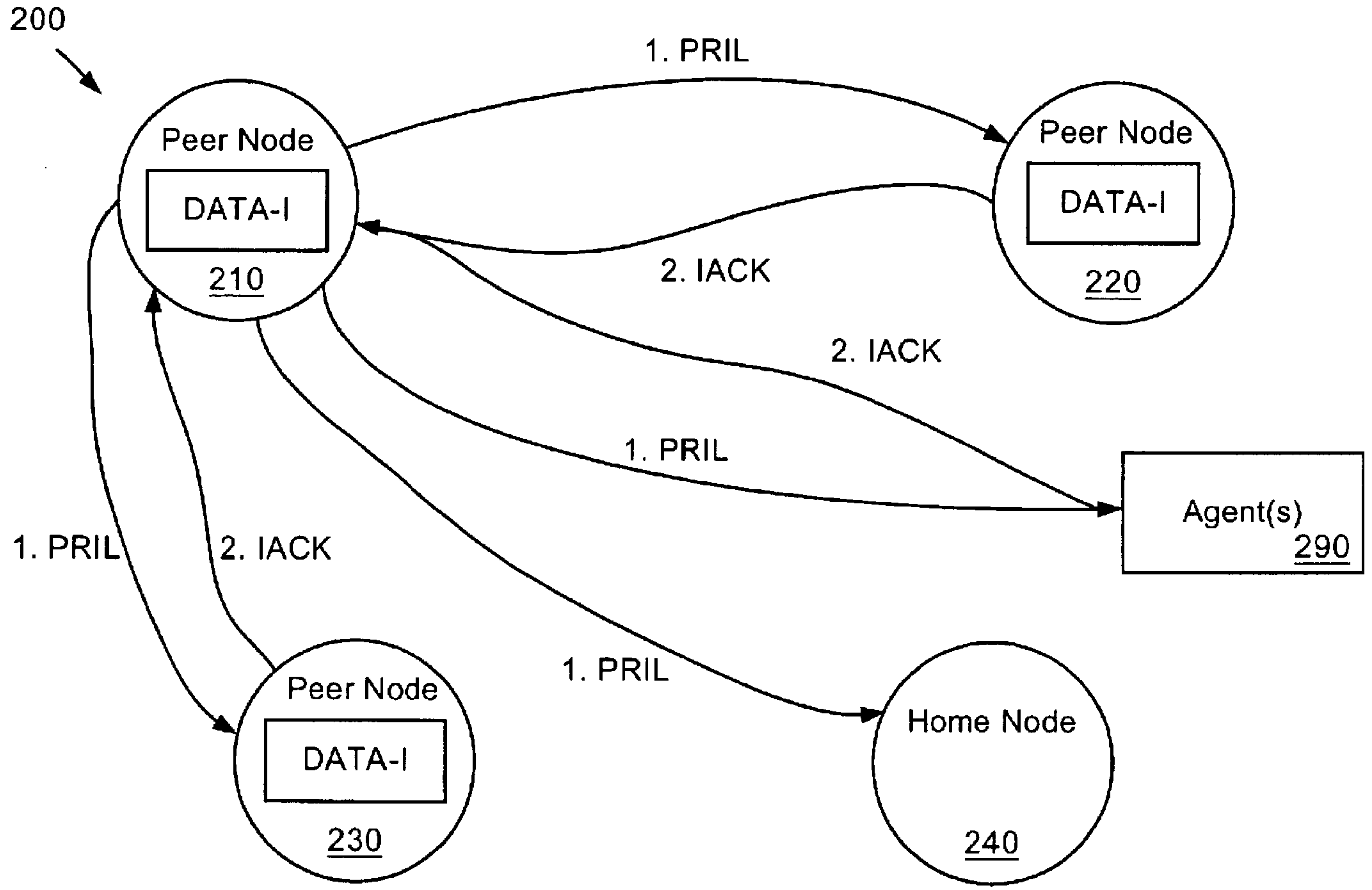

200
1. PRIL
Peer Node
DATA-I
210
Peer Node
DATA-I
220
2. IACK
2. IACK
1. PRIL
Agent(s)
290
1. PRIL
2. IACK
Peer Node
DATA-I
230
1. PRIL
Home Node
240

FIG. 3a

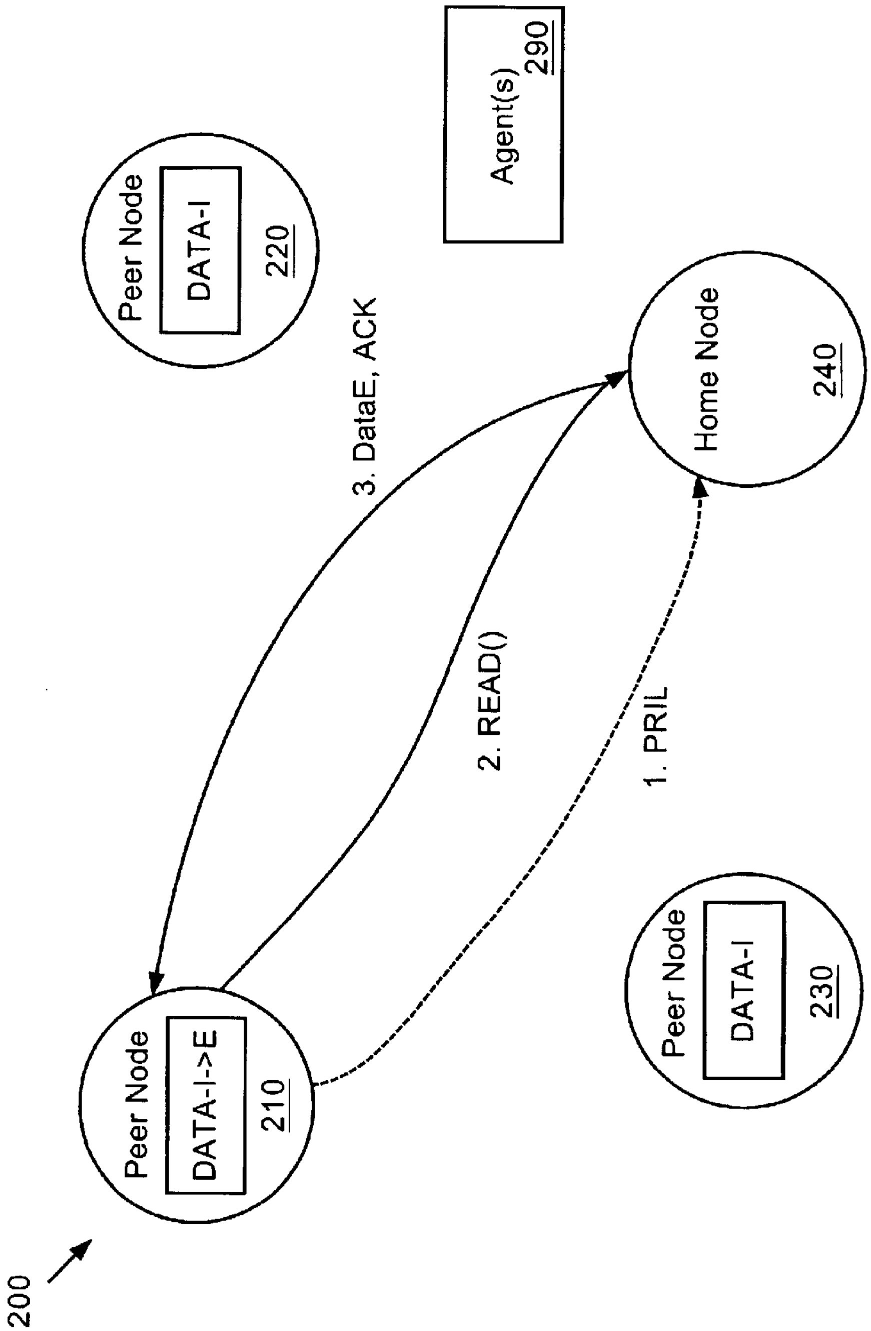

200
Peer Node
DATA-I->E
210
Peer Node
DATA-I
220
Peer Node
DATA-I
230
Home Node
240
Agent(s)
290
1. PRIL
2. READ()
3. DataE, ACK

FIG. 3b

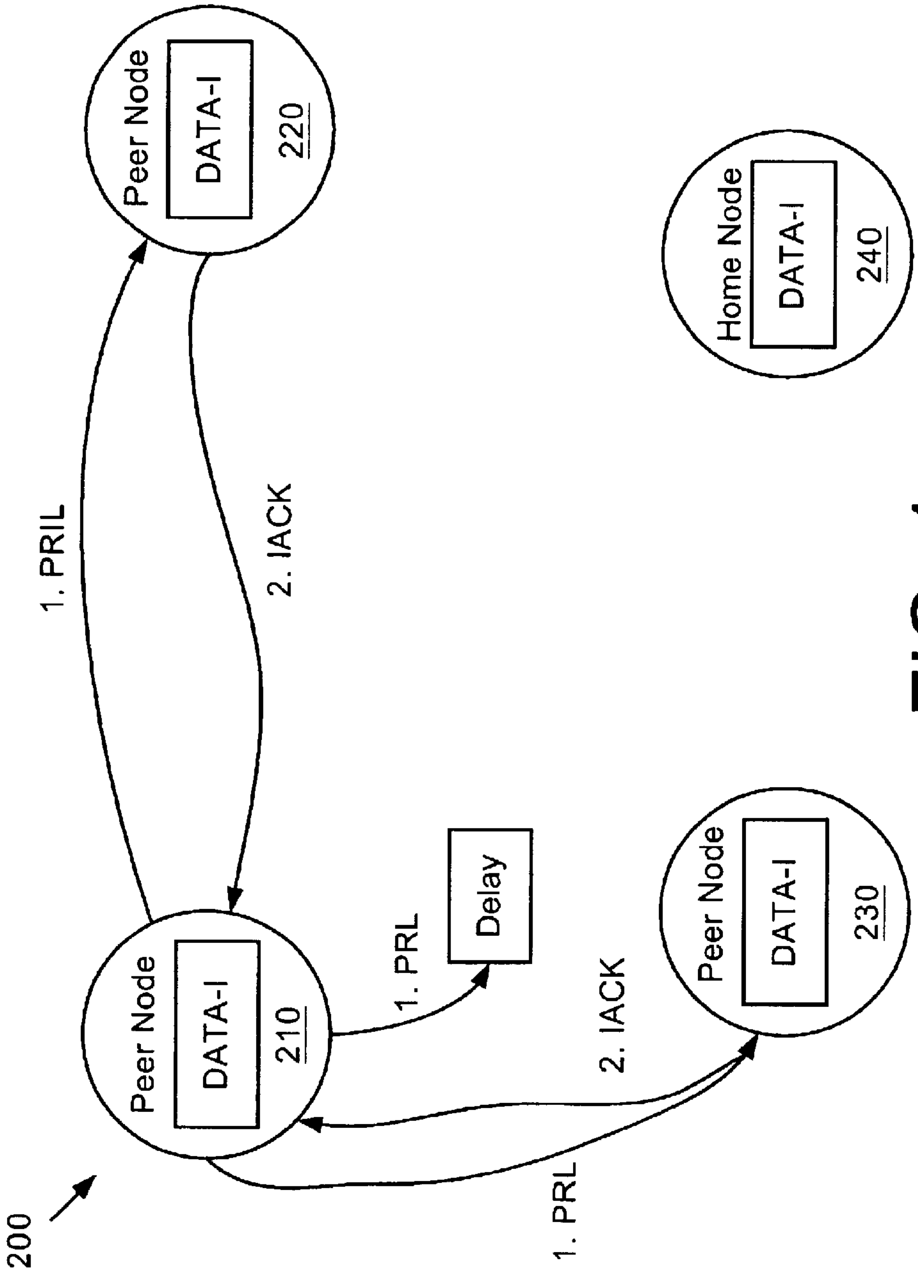

200
Peer Node
DATA-I
210
1. PRIL
2. IACK
Peer Node
DATA-I
220
1. PRL
Delay
2. IACK
1. PRL
Peer Node
DATA-I
230
Home Node
DATA-I
240

FIG. 4a

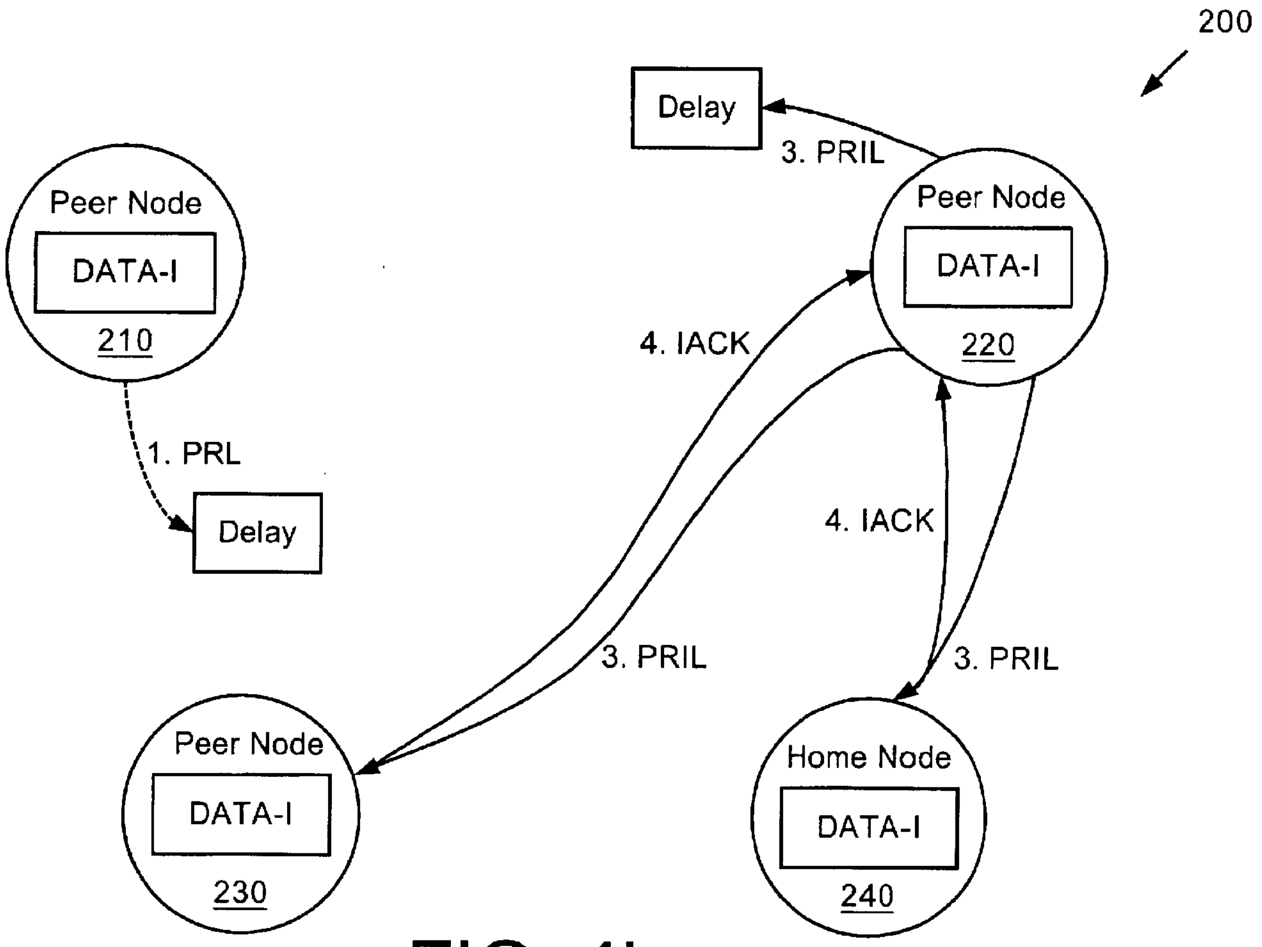

200
Delay
3. PRIL
Peer Node
DATA-I
210
Peer Node
DATA-I
220
1. PRL
Delay
4. IACK
4. IACK
3. PRIL
3. PRIL
Peer Node
DATA-I
230
Home Node
DATA-I
240

FIG. 4b

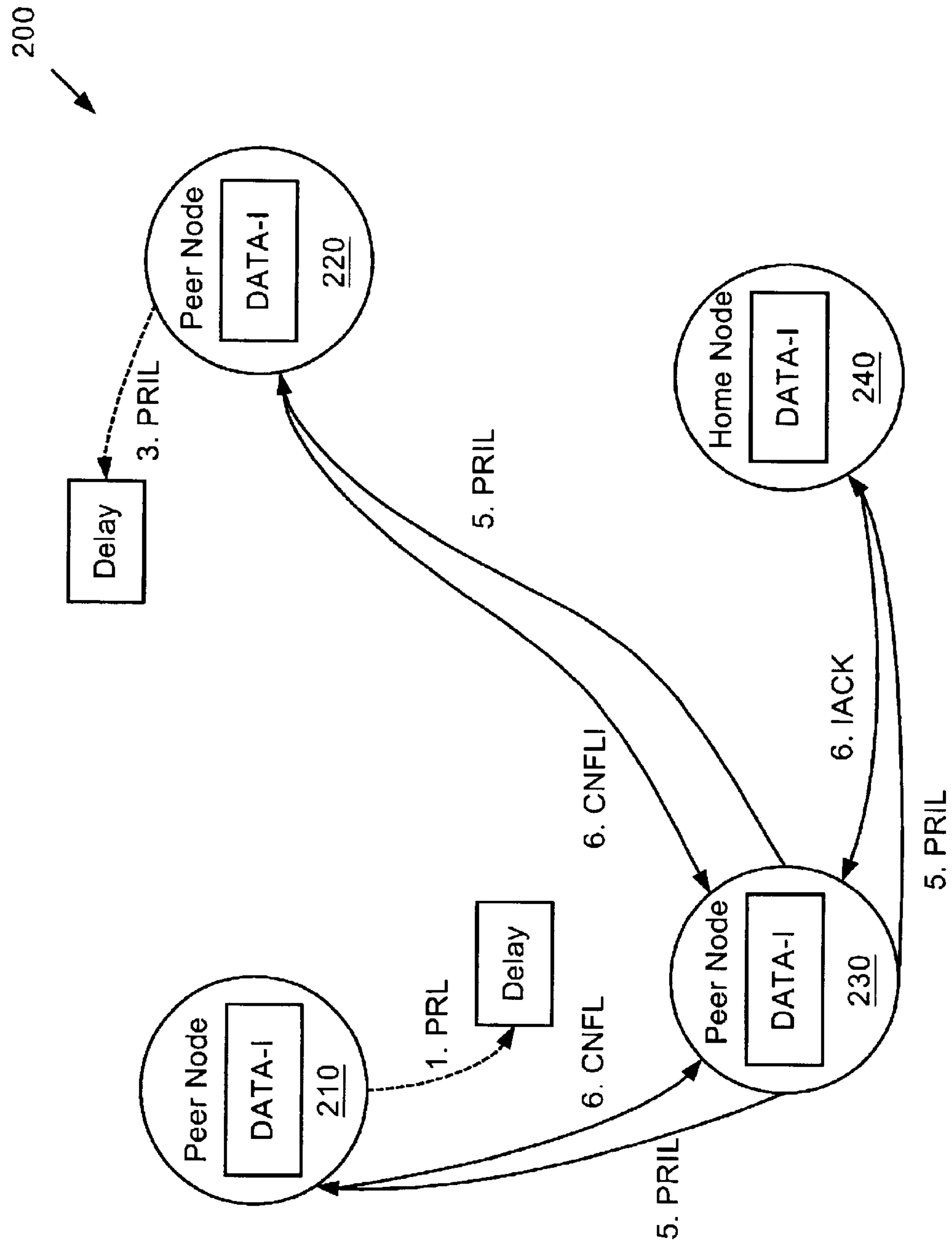

200
Peer Node
DATA-I
210
1. PRL
Delay
6. CNFL
5. PRIL
Peer Node
DATA-I
230
6. CNFLI
5. PRIL
6. IACK
5. PRIL
Peer Node
DATA-I
220
3. PRIL
Delay
Home Node
DATA-I
240

FIG. 4c

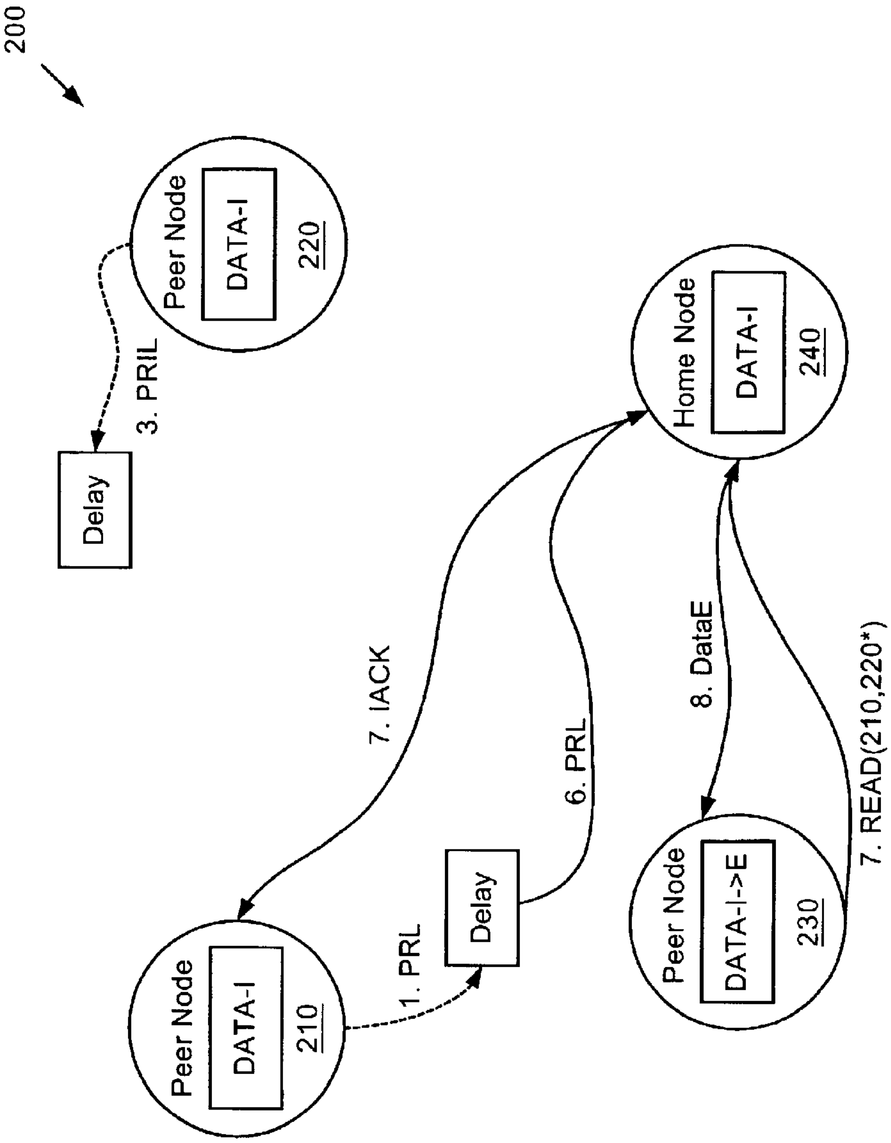

200
Peer Node
DATA-I
220
3. PRIL
Delay
Home Node
DATA-I
240
7. IACK
6. PRL
8. DataE
7. READ(210,220*)
Peer Node
DATA-I
210
1. PRL
Delay
Peer Node
DATA-I->E
230

FIG. 4d

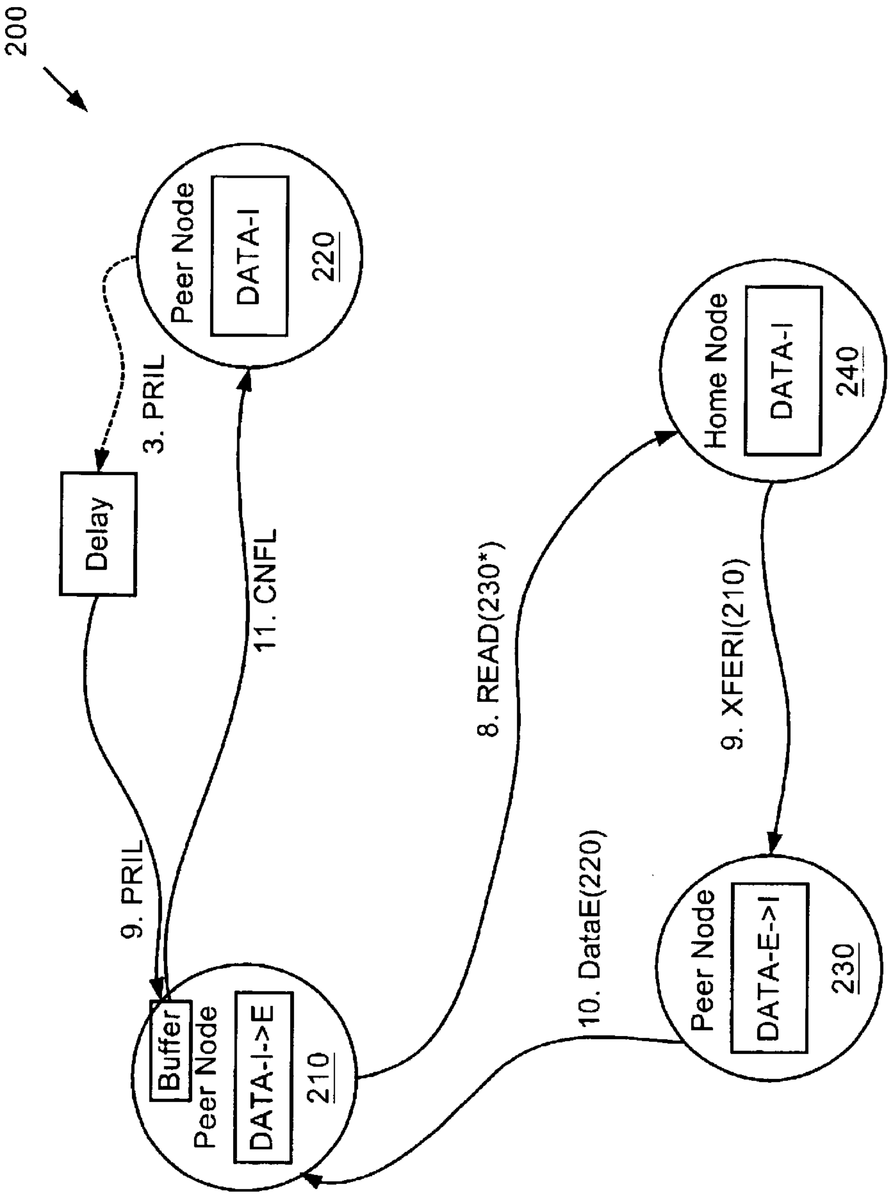

200
Peer Node
DATA-I
220
3. PRIL
Delay
11. CNFL
9. PRIL
Buffer
Peer Node
DATA-I->E
210
8. READ(230*)
Home Node
DATA-I
240
9. XFERI(210)
10. DataE(220)
Peer Node
DATA-E->I
230

FIG. 4e

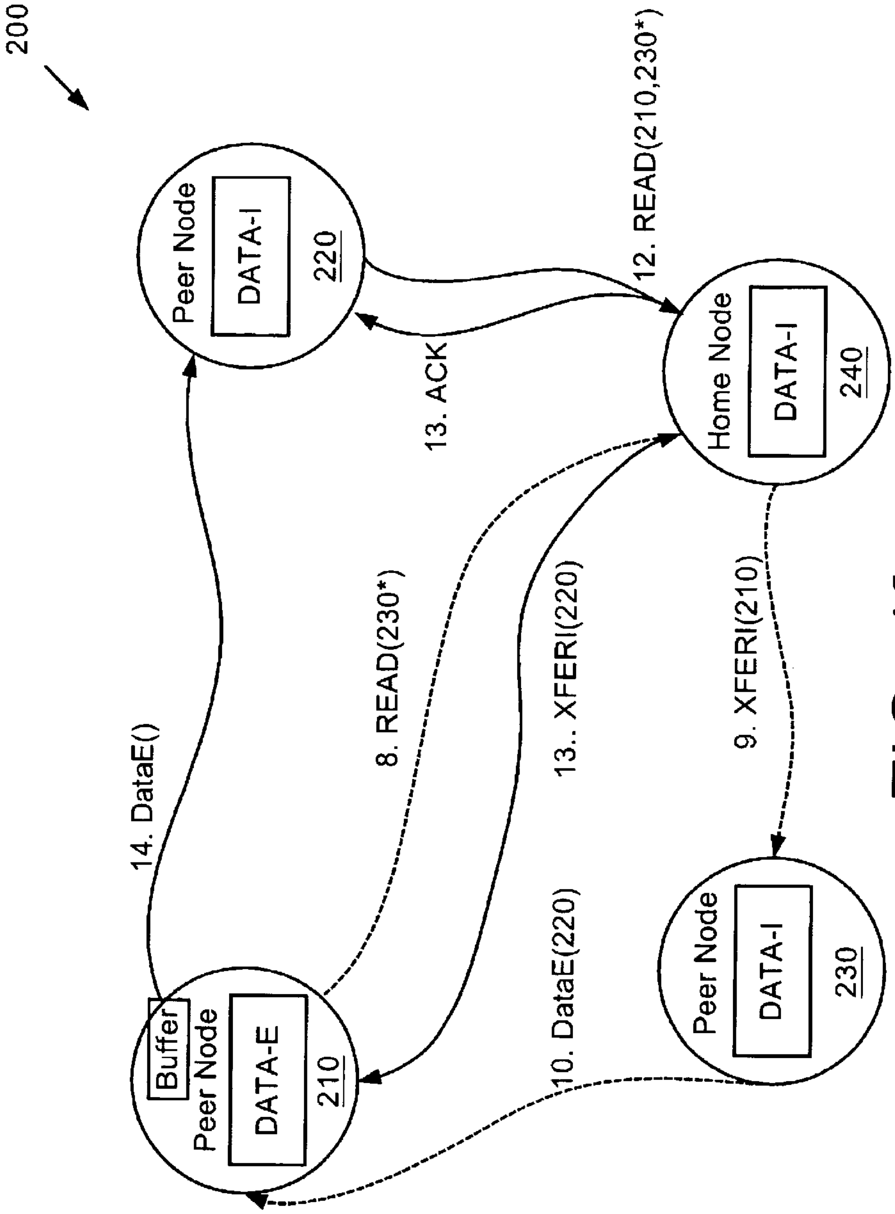

200
Peer Node
DATA-I
220
12. READ(210,230*)
13. ACK
Home Node
DATA-I
240
14. DataE()
8. READ(230*)
13.. XFERI(220)
9. XFERI(210)
Buffer
Peer Node
DATA-E
210
10. DataE(220)
Peer Node
DATA-I
230

FIG. 4f

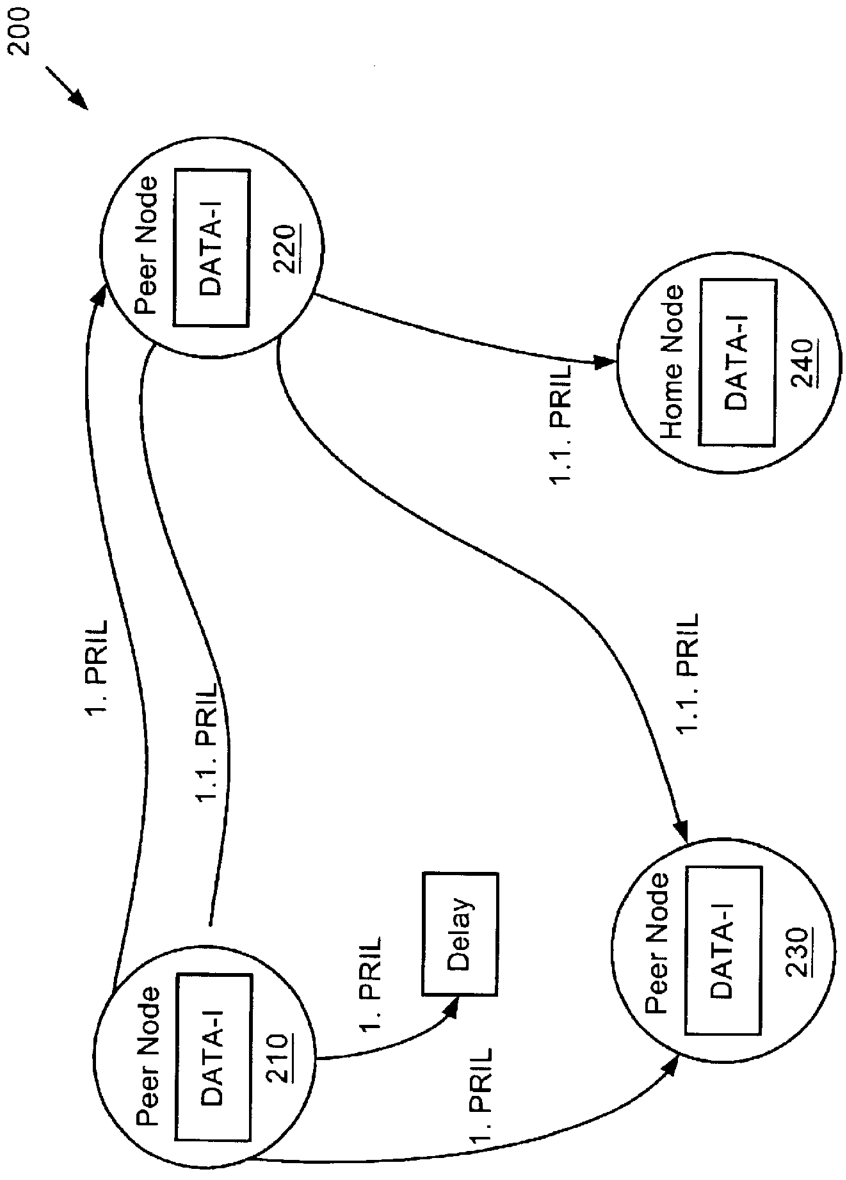

200
Peer Node
DATA-I
220
Home Node
DATA-I
240
1.1. PRIL
1. PRIL
1.1. PRIL
1.1. PRIL
Peer Node
DATA-I
210
1. PRIL
Delay
1. PRIL
Peer Node
DATA-I
230

FIG. 5a

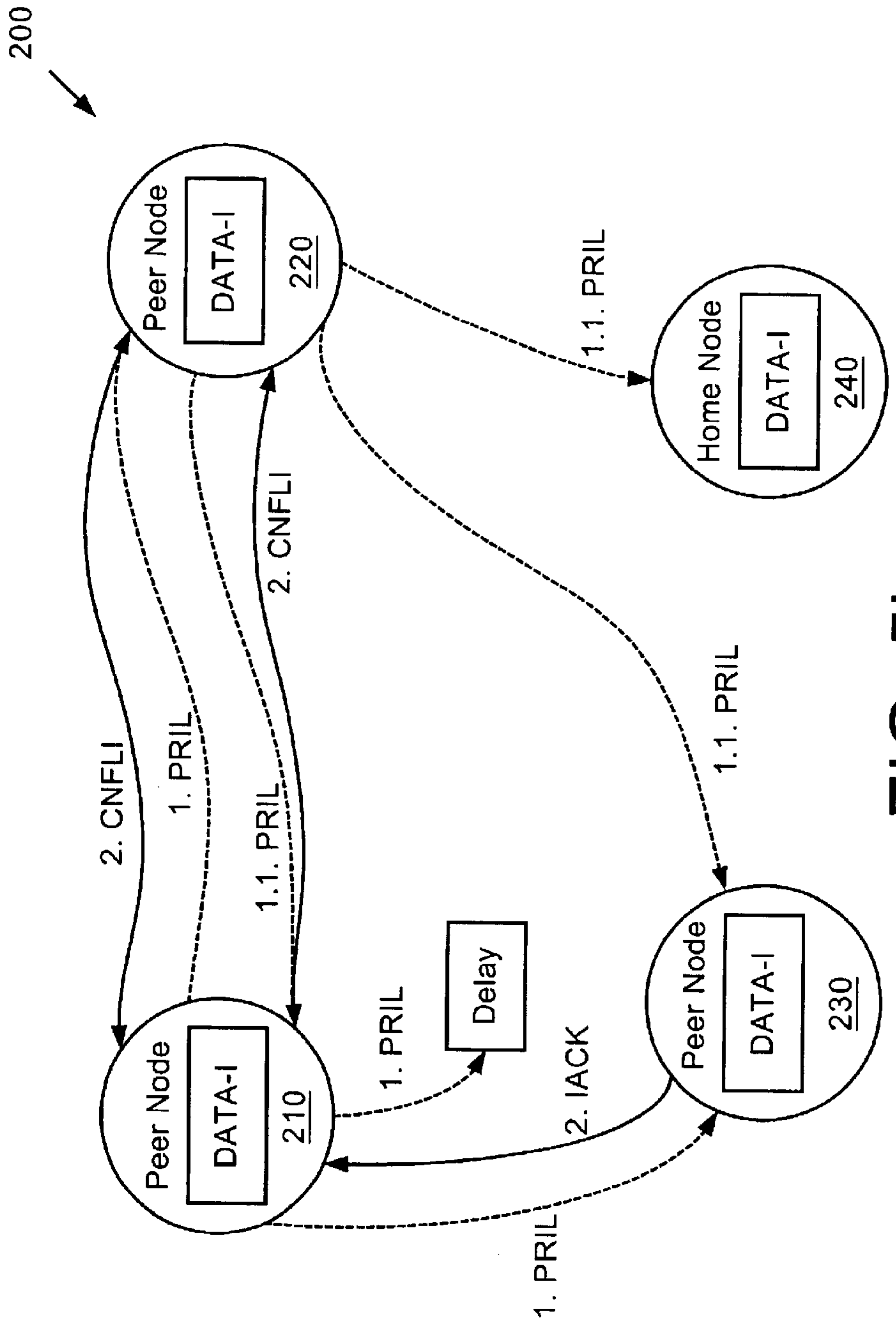

200
Peer Node
DATA-I
220
1.1. PRIL
Home Node
DATA-I
240
2. CNFLI
1. PRIL
1.1. PRIL
2. CNFLI
1.1. PRIL
Peer Node
DATA-I
210
1. PRIL
Delay
2. IACK
Peer Node
DATA-I
230
1. PRIL

FIG. 5b

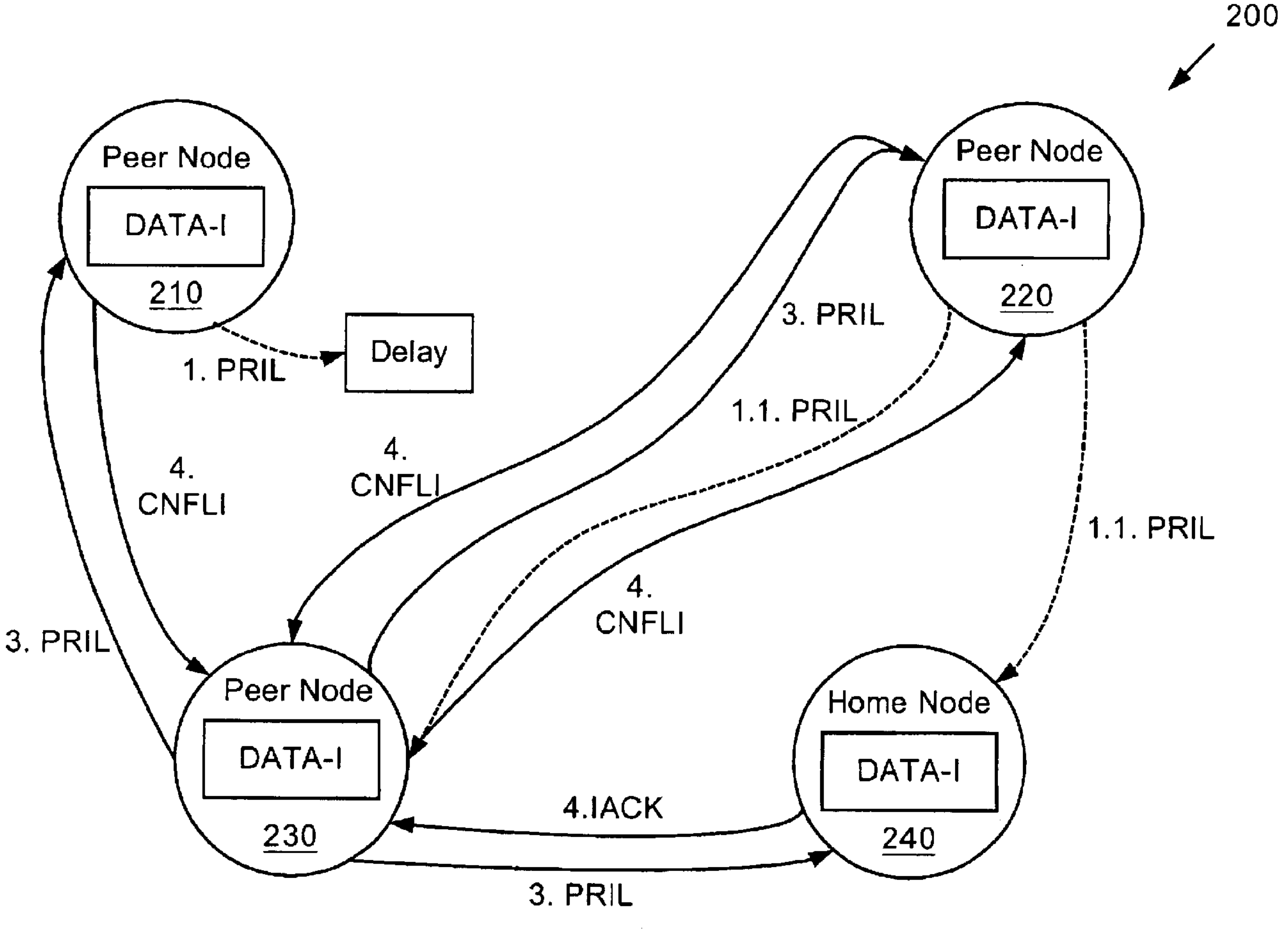

200
Peer Node
DATA-I
210
1. PRIL
Delay
4.
CNFLI
3. PRIL
4.
CNFLI
Peer Node
DATA-I
220
3. PRIL
1.1. PRIL
1.1. PRIL
4.
CNFLI
Peer Node
DATA-I
230
Home Node
DATA-I
240
4.IACK
3. PRIL

FIG. 5c

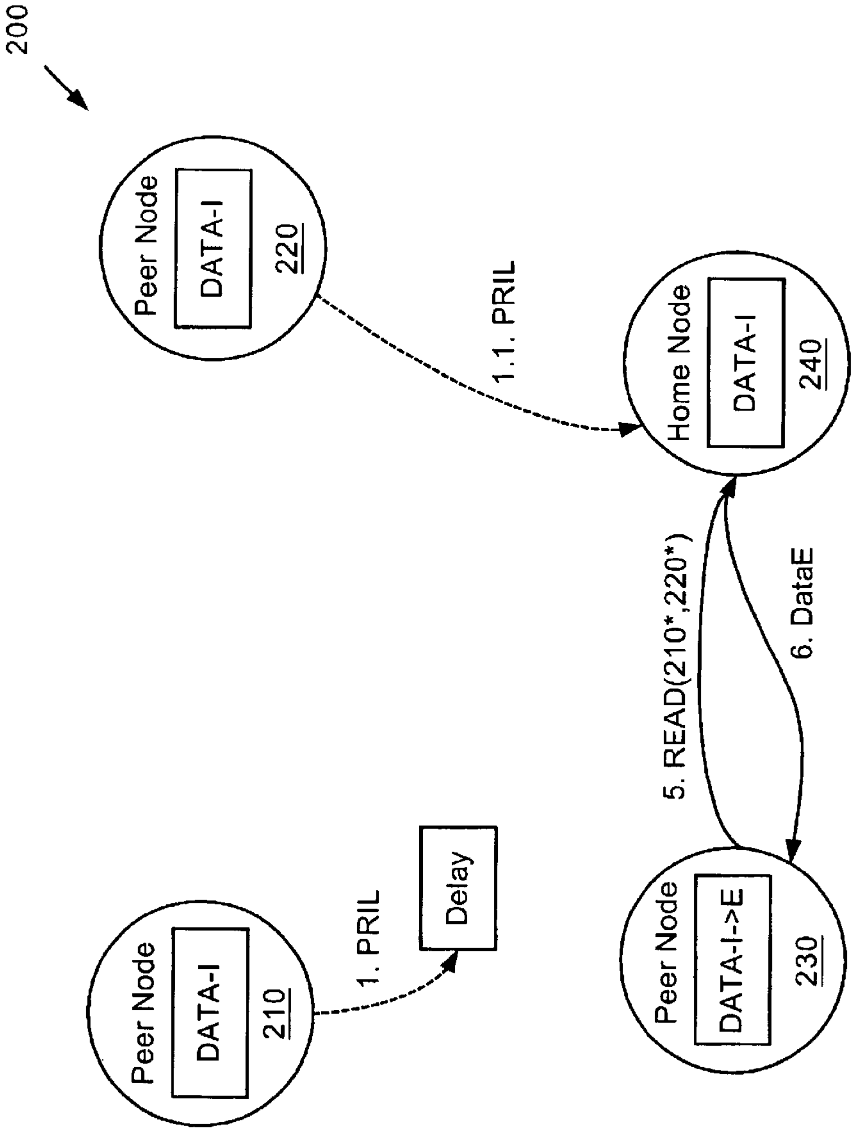

200
Peer Node
DATA-I
220
1.1. PRIL
Home Node
DATA-I
240
5. READ(210*,220*)
6. DataE
Peer Node
DATA-I->E
230
Peer Node
DATA-I
210
1. PRIL
Delay

FIG. 5d

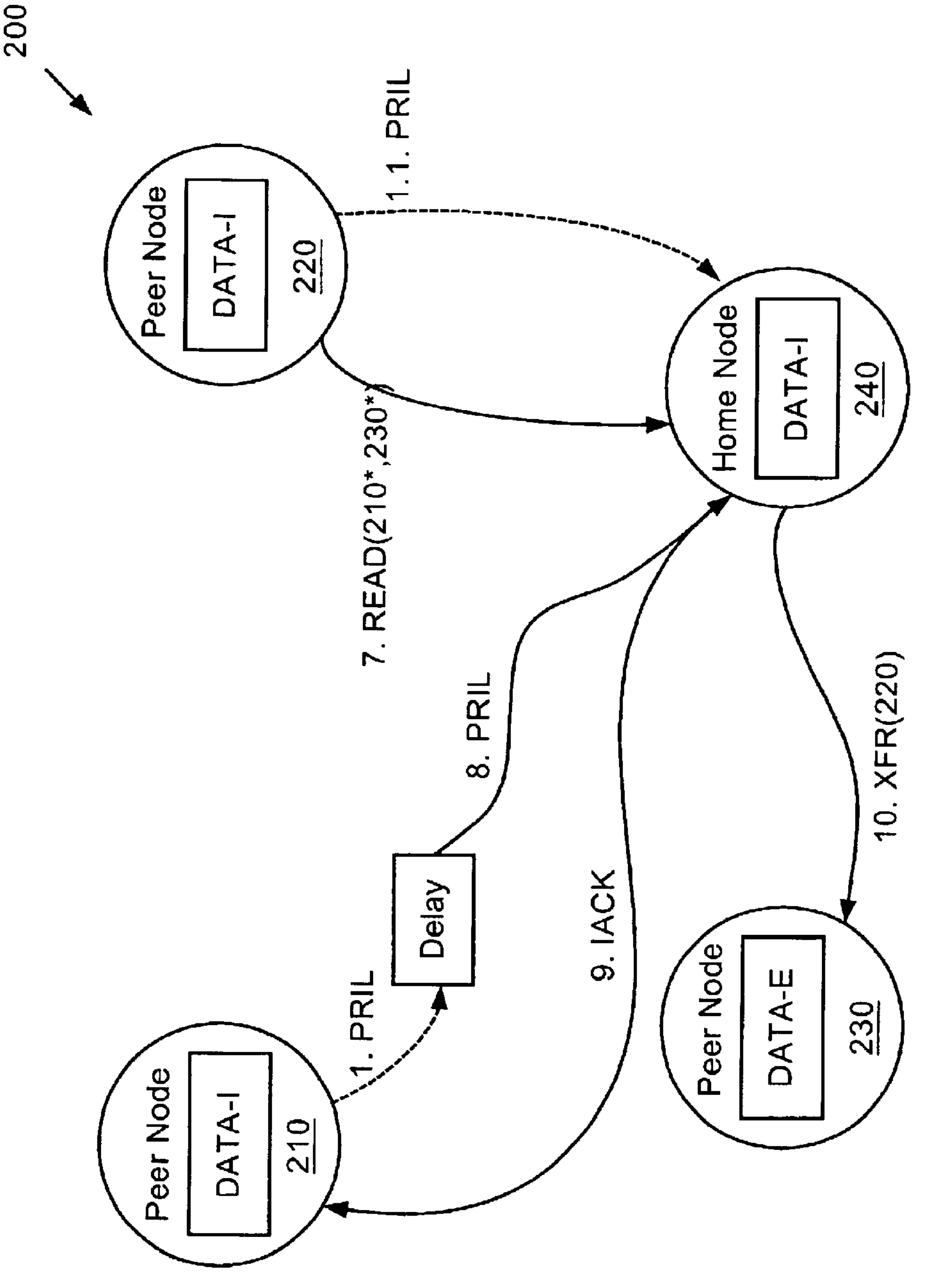

200
Peer Node
DATA-I
220
1.1. PRIL
7. READ(210*,230*)
Home Node
DATA-I
240
Peer Node
DATA-I
210
1. PRIL
Delay
8. PRIL
9. IACK
10. XFR(220)
Peer Node
DATA-E
230

FIG. 5e

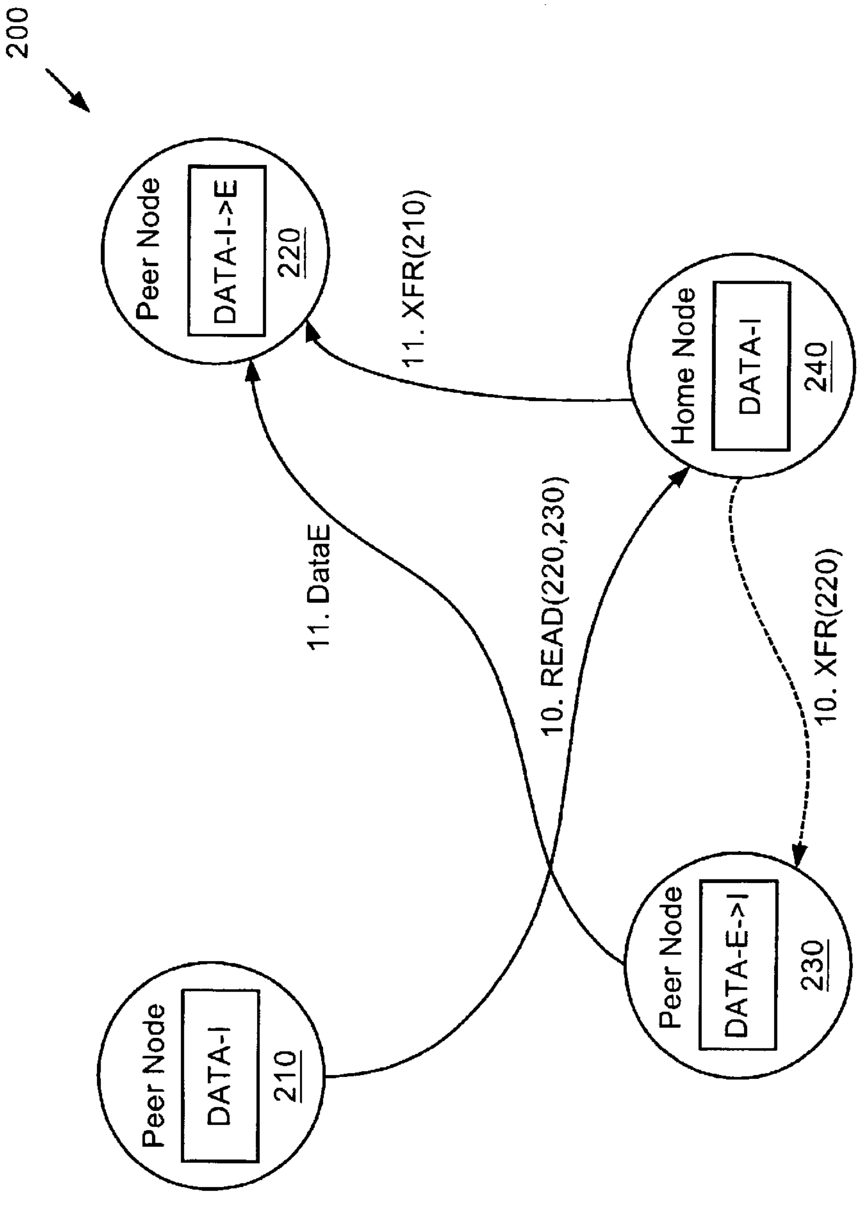

200
Peer Node
DATA-I->E
220
11. XFR(210)
Home Node
DATA-I
240
11. DataE
10. READ(220,230)
10. XFR(220)
Peer Node
DATA-E->I
230
Peer Node
DATA-I
210

FIG. 5f

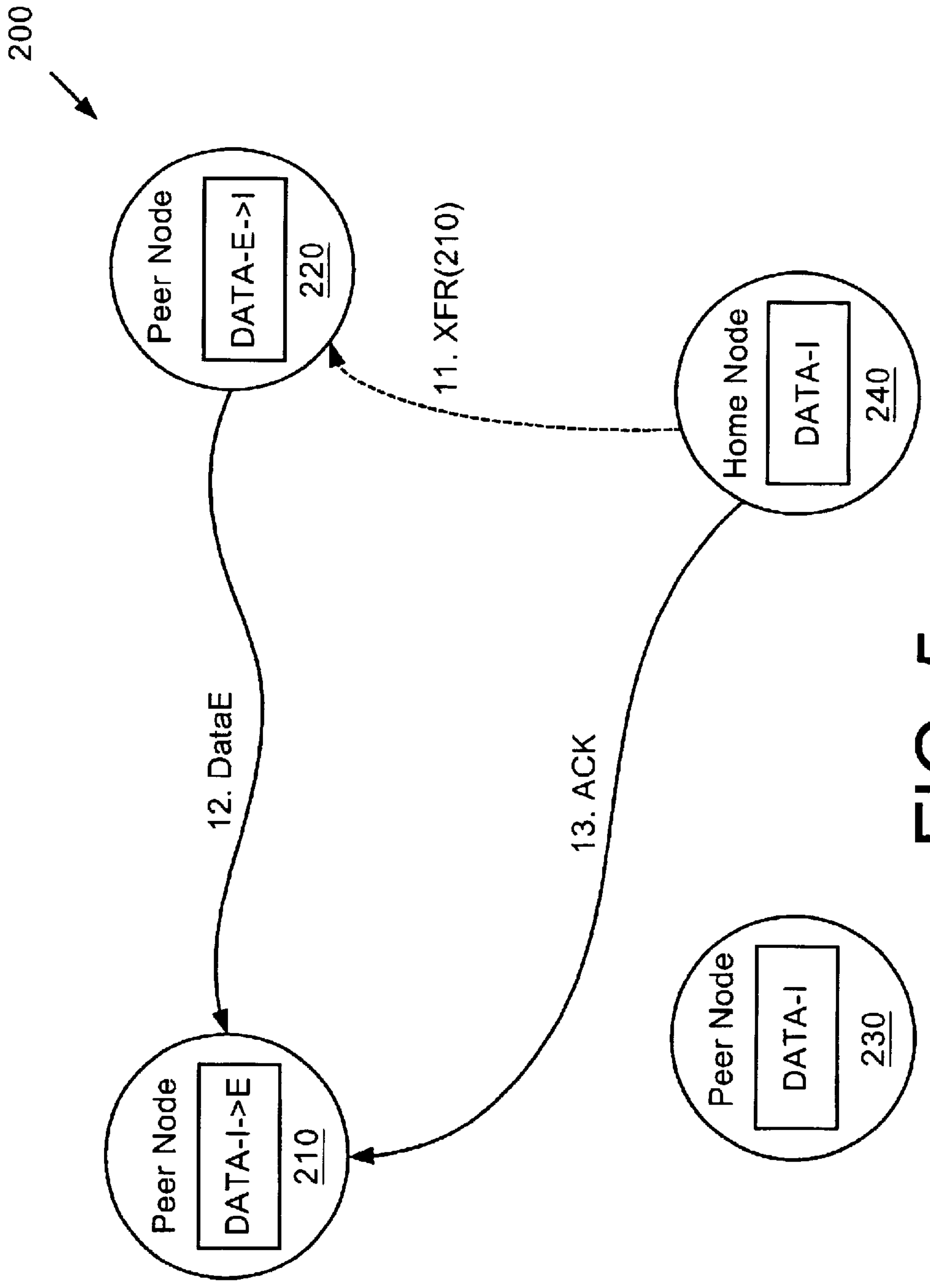

200
Peer Node
DATA-E->I
220
11. XFR(210)
Home Node
DATA-I
240
12. DataE
13. ACK
Peer Node
DATA-I->E
210
Peer Node
DATA-I
230

FIG. 5g

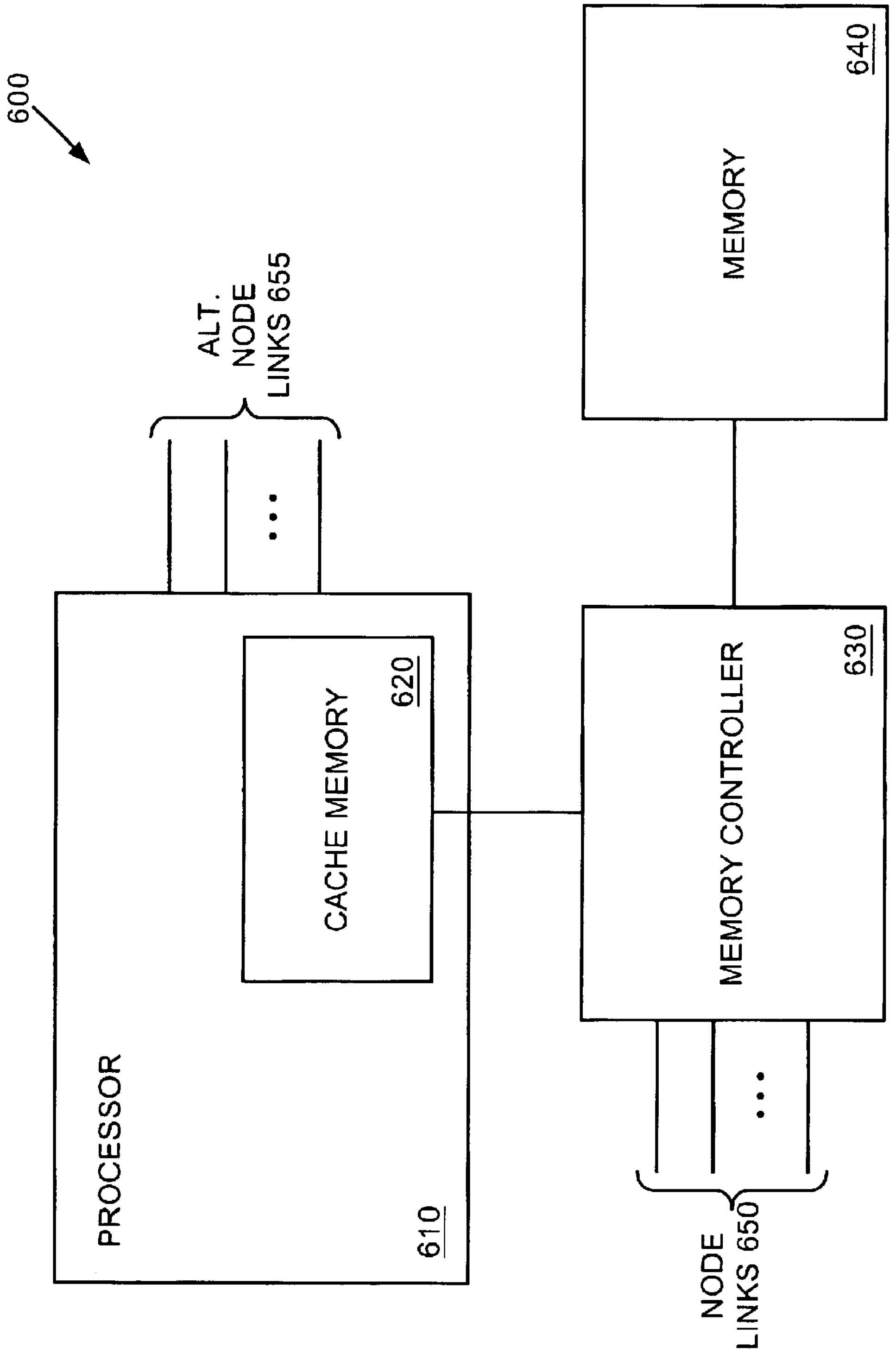

600
ALT. NODE LINKS 655
PROCESSOR
610
CACHE MEMORY
620
MEMORY CONTROLLER
630
MEMORY
640
NODE LINKS 650

FIG. 6

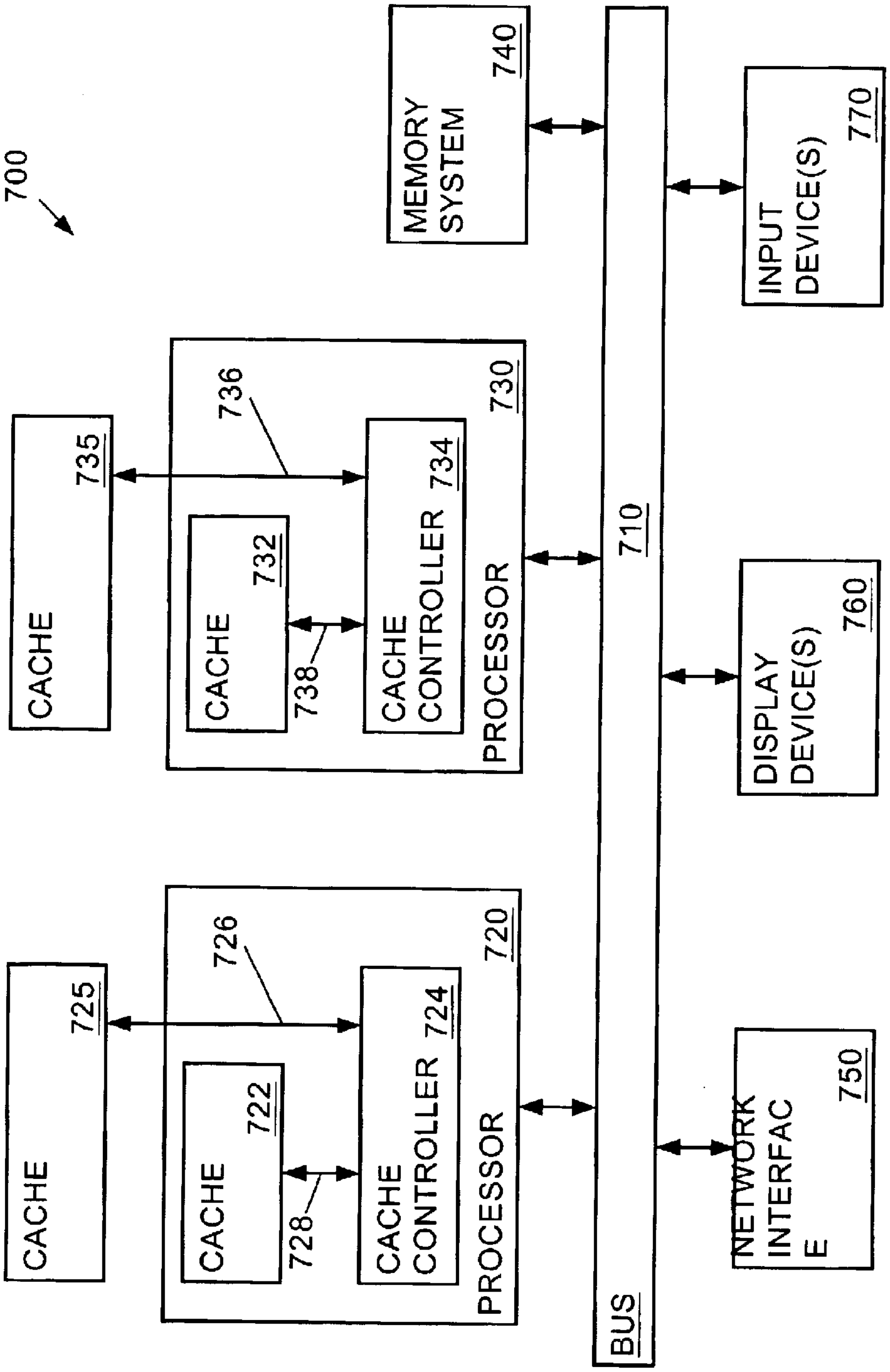

700
MEMORY SYSTEM
740
BUS
710
INPUT DEVICE(S)
770
DISPLAY DEVICE(S)
760
NETWORK INTERFAC E
750
CACHE
735
PROCESSOR
730
CACHE
732
CACHE CONTROLLER
734
736
738
CACHE
725
PROCESSOR
720
CACHE
722
CACHE CONTROLLER
724
726
728

FIG. 7

NON-SPECULATIVE DISTRIBUTED CONFLICT RESOLUTION FOR A CACHE COHERENCY PROTOCOL

RELATED APPLICATIONS

The present U.S. Patent application is related to the following U.S. Patent applications filed concurrently:

(1) Application No. 10/525,069, filed Dec. 19, 2002, entitled "FORWARD STATE FOR USE IN CACHE COHERENCY IN A MULTI-NODE SYSTEM"

(2) Application No. 10/326,234, filed Dec. 19, 2002, entitled "HIERARCHICAL DIRECTORIES FOR CACHE COHERENCY IN A MULTIPROCESSOR SYSTEM"

(3) Application No. 10/324,711, filed Dec. 19, 2002, entitled "HIERARCHICAL VIRTUAL MODEL OF A CACHE HIERARCHY IN A MULTIPROCESSOR SYSTEM"

(4) Application No. 10/325,427, filed Dec. 19, 2002, entitled "SPECULATIVE DISTRIBUTED CONFLICT RESOLUTION FOR A CACHE COHERENCY PROTOCOL"

FIELD

The invention relates to cache memories. More particularly, the invention relates to distributed conflict resolution in a multi-cache multiprocessor system.

BACKGROUND

When an electronic system includes multiple cache memories, the validity of the data available for use must be maintained. This is typically accomplished by manipulating data according to a cache coherency protocol. As the number of caches and/or processors increases, the complexity of maintaining cache coherency also increases.

When multiple components (e.g., a cache memory, a processor) request the same block of data the conflict between the multiple components must be resolved in a manner that maintains the validity of the data. Current cache coherency protocols typically have a single component that is responsible for conflict resolution. However, as the complexity of the system increases reliance on a single component for conflict resolution can decrease overall system performance.

FIGS. **1***a* through **1***e* provide a conceptual illustration of a conflict condition in a multi-node system. Nodes **110**, **120** and **130** are peer nodes that can store a copy of the requested data (e.g., a cache line) in cache memory. Home node **140** is the Home (H) node for the requested data. In the example of FIGS. **1***a* through **1***e*, peer nodes **110** and **120** store an invalid copy, or no copy, of the requested data and peer node **130** stores a modified copy of the requested data that has not been written back to memory. The Home node stores the original copy of the data in memory or modified versions of the data when the modifications are written back to memory.

As illustrated in FIG. **1***a*, peer node **120** transmits a Data Request message to request a copy of a block of data, for example, a cache line. The Data Request message is transmitted to peer node **110** and to peer node **130**. However, the Data Request message to peer node **130** is delayed. The delay can be caused, for example, by lack of available bandwidth, buffering considerations, etc.

Peer node **110** responds to the Data Request message from peer node **120** with a No Valid Copy message, which indicates to peer node **120** that peer node **110** does not have a valid copy of the requested data. At some time after peer node **120** transmits Data Request messages, Peer node **110** transmits Data Request messages to peer nodes **120** and **130**, as illustrated in FIG. **1***c*, requesting the same data as was requested by peer node **120**.

Peer node **120** provides a No Valid Copy message to peer node **110** in response to the Data Request message. Peer node **130** provides the requested data to peer node **110**. The copy of the data, if any, maintained by peer node **130** is marked invalid and the copy of the data stored by peer node **110** is marked Modified.

At some time after peer node **130** has responded to the Data Request from peer node **110** and invalidated the copy of the data, peer node **130** receives the delayed Data Request message from peer node **120**, as illustrated in FIG. **1***c*. In response to the Data Request message, peer node **130** provides a No Valid Copy message to peer node **120**. Note that the state of the data stored by peer node **130** changed from the time of the original Data Request message to the time the peer node **130** responds to the Data Request message.

Because peer nodes **110** and **130** respond to the Data Request message from peer node **120** with No Valid Copy messages, peer node **120**, finding no valid cached copy of the requested data, requests a copy of the data from home node **140**. Thus, as illustrated in FIG. **1***d*, peer node transmits a Read message to home node **140**. Home node **140** retrieves the requested data from memory and provides the data to peer node **120**. Peer node **120** then stores the requested data in the Exclusive state.

As illustrated in FIG. **1***e*, the sequence of messages illustrated in FIGS. **1***a* through **1***e* result in two incompatible copies of a data line. In the example provided peer node **110** stores a copy of the data in the Modified state and peer node **120** stores a copy of the data in the Exclusive state. However, the copy stored by peer node **120** is not exclusive to peer node **120**. Thus, multi-node systems can result in incompatible copies of data under certain circumstances unless a mechanism is provided to resolve cache conflicts.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

FIGS. **1***a* through **1***e* provide a conceptual illustration of a conflict condition in a multi-node system.

FIGS. **2***a* and **2***b* are conceptual illustrations of a request for a shared cached line.

FIGS. **3***a* and **3***b* are conceptual illustrations of a request for a shared uncached line.

FIGS. **4***a* through **4***f* are conceptual illustrations of three-way conflicting requests for a shared line.

FIGS. **5***a* through **5***g* are conceptual illustrations of three-way conflicting requests for a shared line.

FIG. **6** is a block diagram of one embodiment of a node.

FIG. **7** is one embodiment of a multiprocessor system.

DETAILED DESCRIPTION

Techniques for piggybacking multiple cache protocol messages into a single message packet are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details. In other instances, structures and devices are shown in block diagram form in order to avoid obscuring the invention.

Request Messages

The following messages are requests for data/action from a requesting node. These messages are broadcast to all nodes of the system.

Port Read Line (PRL): This is a request for a copy of a data segment such as, for example, a cache line.

Port Read Invalidate Line (PRIL): This is a request for a copy of a data segment where the provider node's copy of the data is invalidated. This message can also be referred to as a "request for ownership."

Port Write Line (PWL): This message causes data (e.g., a modified cache line) to be written to memory. This message can also referred to as a "dirty eviction."

Port Invalidate Line (PIL): This message causes the state of designated data to change from Shared to Exclusive.

Port Write Invalidate Line (PWIL): This message causes data to be written to memory and the target copy of the data to be invalidated.

Response Messages

The following messages are messages from sent from Peer (i.e., Non-Home) nodes to the Requesting node in response to requests described above.

Invalid State Acknowledgement (IACK): This message is a response to a request (PRL, PRIL, PIL, PWIL) when the node sending the response has an invalid copy of the requested data or no copy of the requested data.

Shared State Acknowledgement (SACK): This message is a response to a PRL request when the node sending the response has a copy of the requested data in the Shared state.

Acknowledgement of Data Received (DACK): This message acknowledges the receipt of requested data. This node is sent by the home node when the home node receives a CNCL message. The target node for the DACK message is the forwarding node included in the CNCL message. A node receiving the transferred data or memory data from the home node does not respond to the sending node.

Conflict: This message indicates that there is a copending request for the requested cache line.

Data(Conflicts): This message is used to forward data and lists conflicts, if any. The conflict list is empty when data is forwarded or when the home node sends memory data to the first owner. When data is transferred, the sending node attaches a list of conflicts. The receiving node uses the list to send CNFL responses to matching requests stored in a buffer.

Messages to Home Node

These messages are transmitted to the Home node by a Peer node.

Read(Conflicts): This message requests data from the Home node and lists conflicts, if any. This message is sent after all responses are received by a peer node if none of the received messages were a DATA message.

CNCL(Conflicts): This message is sent to the Home node in response to a hit in a Peer node and lists all conflicts, if any. This message cancels the Home node's prefetch operation.

Messages From the Home Node

These messages are sent from the Home node to the Peer and/or Requesting nodes.

Data: This message includes the requested data and can indicate the state of the data (M/E/F) to be used by the Requesting node.

Acknowledge (ACK): This message indicates that the requested data has been sent to the Requesting node. When the home node sends an ACK message, the current epoch is terminated.

Transfer (XFR): This message causes the receiving node to transfer data to the node indicated in the message. The home node sends this message to the current owner of requested data when the home node is informed of a conflict condition requiring that the current owner of the data transfer the data to the target node. A XFRI message is sent instead of a XFR message if the home node determines that an unresolved conflicting request is a PRIL or PWIL message, meaning that the current owner must invalidate the line when initiating the data transfer. In one embodiment, the first node in an epoch to send a CNCL message is the current owner. An epoch is the period between a first request for data and the resolution of all conflicting requests for the data. If the home node sends data to a node from memory, that node is the current owner. Sending a XFR/XFRI message causes the target node to become the current owner. In one embodiment, the target node is selected from a list of conflicts provided to the home node in a READ or a CNCL message. Target nodes are selected from nodes from which the home node has received a READ message. Thus, if the home node has A as the current owner of a cache line because A sent the home node a CNCL(B,C), the home node waits until a READ message comes from B or C to send a XFR/XFRI message to A to cause A to forward the data to the node (B or C) that sent the READ message. The home node then waits for the third node to sent a READ message before sending a XFR/XFRI message causing the data to be sent to the third node.

Overview of the MESIF Protocol

There are two basic schemes for providing cache coherence, snooping (now often called Symmetric Multi-Processing SMP) and directories (often called Distributed Shared Memory DSM). The fundamental difference has to do with placement and access to the meta-information, that is, the information about where copies of a cache line are stored.

For snooping caches the information is distributed with the cached copies themselves, that is, each valid copy of a cache line is held by a unit that must recognize its responsibility whenever any node requests permission to access the cache line in a new way. Someplace—usually at a fixed location—is a repository where the data is stored when it is uncached. This location may contain a valid copy even when the line is cached. However, the location of this node is generally unknown to requesting nodes—the requesting nodes simply broadcast the address of a requested cache line, along with permissions needed, and all nodes that might have a copy must respond to assure that consistency is maintained, with the node containing the uncached copy responding if no other (peer) node responds.

For directory-based schemes, in addition to a fixed place where the uncached data is stored, there is a fixed location, the directory, indicating where cached copies reside. In order to access a cache line in a new way, a node must communicate with the node containing the directory, which is usually the same node containing the uncached data repository, thus allowing the responding node to provide the data when the main storage copy is valid. Such a node is referred to as the Home node.

The directory may be distributed in two ways. First, main storage data (the uncached repository) is often distributed among nodes, with the directory distributed in the same way. Secondly, the meta-information itself may be distributed, keeping at the Home node as little information as whether the line is cached, and if so, where a single copy resides.

Snooping schemes rely on broadcast, because there is no single place where the meta-information is held, so all nodes must be notified of each query, each node being responsible for doing its part to assure that coherence is maintained. This includes intervention messages, informing the Home node not to respond when another node is providing the data.

Snooping schemes have the advantage that responses can be direct and quick, but do not scale well because all nodes are required to observe all queries. Directory schemes are more scalable, but require more complex responses, often involving three nodes in point-to-point communications.

The basic MESIF protocol described herein provides a snooping-like protocol without the limitations of a single, serializing bus. Like a snooping cache protocol, MESIF relies on nodes with cached copies of data to maintain coherence. The use of point-to-point links rather than a synchronous, centralized broadcast introduces the problem of time-warp—the fact that events appear to occur in a different order from the viewpoint of different node. As described in greater detail below, the MESIF protocol handles time-warp, recognizing when potential errors could result and making sure that they are handled correctly. The notion of a Home node is used to determine where the uncached copy reside, but the Home node participates in every transaction—without being on the critical path—in order to resolve conflicts and time-warp issues. Because of the concurrent-broadcast nature of the scheme, MESIF achieves the low latency associated with snooping protocols, acquiring a cacheable copy of the data in most cases in the minimum possible latency: a single roundtrip request-response.

The basic MESIF protocol involves broadcasting an initial request to all peer nodes as well as the Home node. If a copy is cached in state E, F, or M, it is included in the response. A second message is then sent to the Home node, informing it that the request has been satisfied. If the requested line is uncached, or if only S-state copies exist, the second request sent to the Home node is used to confirm the previous request, which the Home node may have by now fetched from its memory. In either case, the Home node must respond to the second request (and to the first, though they can sometimes be combined) for purposes of synchronization and conflict resolution. Note that the Home node may have one or more caches, so it may respond to the initial request just like any other node.

Conflicts are handled in a distributed way. The timewarp problem makes it difficult to detect conflicts, because individual requests can be delayed for an arbitrarily long time. A conflict will be detected, however, if each node monitors for conflicts after making a request. Both nodes may or may not detect a conflict, but at least one will. Because all nodes must respond to a broadcast request either by supplying the data or indicating they do not have a copy (or, under some circumstances, are not supplying the copy they have), the response can include an indication of a conflict, so conflicting nodes will detect the conflict.

Complications arise from allowing a node to use the data as soon as the data arrives rather than waiting for all responses. Thus, a node receiving a copy of the data is allowed to use the data internally immediately upon receipt, but may not make the effects of using the data visible to the rest of the system until the node has received a confirmation from Home. The confirmation may also include instructions that the node must forward its copy to another node, and perhaps evict the node from its own cache.

Finally, when a node responds to a request from another node by supplying the cached data, the node must defer all other requests it receives for the same cache line until the node receives a response from the Home node acknowledging the fact that the node forwarded the data, thus assuring that all nodes observe the same order of transfer of the (possibly writable) cache line.

The Home node is the respository for the uncached data, but the home node also may have a processor generating requests and include one or more caches. Like any other node, when the home nodes processor misses, the home node must broadcast requests to all other (peer) nodes, and the home node must handle the request internally as it would any other request arriving for the Home node. Note that this is a special case in that the home node does not explicitly send messages to itself (the Home node). In addition, when an external request arrives for data that is cached locally, the Home node must respond in a way to assure that a later response from the Home node is not ambiguous. That is, the home node may respond to the initial request by supplying the data, but the home node must also respond to the second request as the Home node.

Variants of the protocol allow the Home node to respond with the uncached copy of the data without knowing whether this data is valid, leaving it to the requesting node and the second response from the Home node to sort out the case where data was supplied inappropriately.

A more detailed, pseudo-code-based description of various embodiments of the MESIF protocol suitable for use as described herein are attached hereto as Appendix A.

Overview of Non-Speculative Distributed Conflict Resolution

In general, a cache coherency protocol requires conflict resolution in order to provide orderly changes of state for the various cache lines or other blocks of data. A conflict resolution technique described herein provides sequential consistency, which means that only a single, modifiable copy of a cache line can be present at any time, and that no copy of a cache line can be modified while other copies are readable. Conflicting requests to modify a copy of a cache line must, therefore, be resolved in order to maintain sequential consistency.

In one embodiment, a conflict is resolved using the property of time. That is, regardless of delays, two nodes cannot both request a cache line before the other. Thus, conflicts can be detected by at least one of the conflicting requestors if each node monitors all requests after that node has made its own request.

In one embodiment, if a line is in the Exclusive (E), Modified (M) or Forward (F) state, conflicts are resolved at the node holding the unique copy. The winner of the conflict resolution, and possibly the losers, report the conflict to the home node, which pairs conflict reports and issues forwarding instructions to assure that all requesting nodes eventually receive the requested data. In one embodiment, if a requested cache line is either uncached or present only in the Shared (S) state, the home node for the requested cache line provides a copy of the requested data and resolves conflicts.

In one embodiment, the distributed conflict resolution described herein is part of a cache protocol referred to as the MESIF protocol in which one of five states is associated with a cached copy of a cache line (Modified, Exclusive, Shared, Invalid, Forward). In one embodiment, a blackout period after all responses for a request until an acknowledgement message has been received from the home node allows all conflicting nodes to be aware of conflicts in which the nodes are involved. The blackout period does not limit use of the data in the cache, but does prevent the data from being propagated to other caches.

The discussion that follows is provided in terms of nodes within a multi-node system. In one embodiment, a node includes a processor having an internal cache memory, an external cache memory and/or an external memory. In an alternate embodiment, a node is an electronic system (e.g., computer system, mobile device) interconnected with other electronic systems. Other types of node configurations can also be used. In the examples that follow, dashed lines represent messages previously sent and solid lines represent messages being described. In order to keep the figures more clear, when a set of messages (e.g., PRIL and corresponding IACK) are resolved, the lines representing the messages are no longer included in the figures.

FIGS. **2***a* and **2***b* are conceptual illustrations of a request for a shared cached line. The numbering associated with the various messages of FIGS. **2***a* and **2***b* (e.g., 1. PRIL, 7.IACK) provide approximate ordering for purposes of explaining an example of a conflict. The exact timing relationships illustrated in FIGS. **2***a* and **2***b*, as well as the other examples provided (i.e., FIGS. **3***a*–**3***f*) are not required.

As illustrated in FIG. **2***a*, peer node **210** transmits a PRIL message to peer nodes **220** and **230** and to home node **240**. Peer node **210** could also request the same block of data using a PRL message, in which case peer node **230** would not invalidate its copy in response to the request message. Peer node **220** responds to the PRIL message with an IACK message indicating that peer node **220** cannot provide a valid copy of the requested data. Peer node **220** is illustrated as initially having a copy of the requested data in the S state, which is a valid copy of the data, but not a copy that can be provided in response to a request for a copy of the data.

Because a PRIL message requests a copy of data and causes all other copies to invalidate remaining copies of the data, peer node **220** transitions the copy of the data in the S state to the I state. Because peer node **230** has a copy of the requested data in the F state (the only valid copy to be provided to a requestor), peer node **230** provides a copy of the requested data to peer node **210**. Peer node **230** also transitions its copy of the requested data to the I state. Peer node **210** stores the requested data in the E state. Alternatively, peer node **210** stores the requested data in the F state.

As illustrated in FIG. **2***b*, in response to receiving the requested data from peer node **230**, peer node **210** sends a CNCL(**230**)( ) message to home node **240**, which causes home node **240** to cancel retrieval of the requested data from memory (or to not transmit the data, if already retrieved). The CNCL(**230**)( ) message also indicates to home node **240** that a copy of the requested data was received from peer node **230** and that peer node **210** has not identified any conflicts with the PRIL message requesting the data.

In response to the CNCL(**230**)( ) message from peer node **210**, home node **240** transmits an ACK message to peer node **210** and a DACK message to peer node **230**. The ACK message indicates to peer node **210** that home node **240** acknowledges receipt of the CNCL(**230**)( ) message. The DACK message from peer node **240** to peer node **230** acknowledges receipt of the data by peer node **210** and concludes the process for requesting data by peer node **210**.

FIGS. **3***a* and **3***b* are conceptual illustrations of a request for a shared uncached line. As illustrated in FIG. **3***a*, peer node **210** transmits a PRIL message to request a copy of a block of data. Because peer nodes **220** and **230** do not store a valid copy of the requested data, nodes **220** and **230** respond with an IACK message.

As illustrated in FIG. **3***b*, because peer node **210** has received an IACK message from all peer nodes, peer node **210** sends a READ( ) message to home node **240**, which requests a copy of the previously requested data that has been retrieved from memory by home node **240**. The READ( ) message also indicates to home node **240** that peer node **210** has not identified any conflicts with the PRIL message. Home node **240** provides a copy of the requested data with a DataE message. In one embodiment, home node **240** also includes an ACK message with the DataE message (i.e., in the same message packet). In an alternate embodiment, the DataE and ACK messages are transmitted separately.

FIGS. **4***a* through **4***f* are a conceptual illustration of three-way conflicting requests for a shared, cached line. As illustrated in FIG. **4***a*, peer node **210** sends a PRL message to peer nodes **220** and **230** and to home node **240**. The PRL message to home node **240** is delayed for some reason, for example, latency delays within system **200**. In response to the PRL message, because neither peer node **220** nor peer node **230** can provide a valid copy of the requested data, peer node **220** and peer node **230** send IACK messages to peer node **210**.

The PRL message from peer node **210** does not require any of the receiving peer nodes to invalidate copies, if any, stored by the receiving peer nodes. Peer node **210** can also use a PRIL message to request the data, which would require that all peer nodes storing the requested data, whether provided to the requesting node or not, to invalidate the copy of the requested data stored by the node. Conflicts can be caused by any combination of messages that would otherwise cause an inconsistent result.

As illustrated in FIG. **4***b*, as some time after peer node **220** receives the PRL message from peer node **210** and before home node **240** receives the PRL message from peer node **210**, peer node **220** transmits a PRIL message requesting the same block of data. Peer node **220** transmits the PRIL message to peer nodes **210** and **230** and to home node **240**; however, the PRIL message to peer node **210** is delayed. Peer node **230** and home node **240** respond to the PRIL message from peer node **220** with IACK messages.

As illustrated in FIG. **4***c*, peer node **230** subsequently transmits a PRIL message to peer nodes **210** and **220** and to home node **240**. Home node **240** responds to the PRIL message with an IACK message indicating that home node **240** does not store a valid copy of the requested data. Peer node **210** responds to the PRIL message from peer node **230** with a CNFL message indicating that peer node **210** has a conflict with the PRIL message received from peer node **230**.

Peer node **220** responds to the PRIL message from peer node **230** with a CNFLI message indicating that peer node **220** has a conflict with the PRIL message received from peer node **230**. The CNFLI message indicates that the conflicting message from peer node **220** is a PRIL message, which requires invalidation. The CNFL message from peer node **210** indicates that the conflicting message from peer node **210** is a PRL message, which does not require invalidation.

As illustrated in FIG. **4***d*, when home node **240** receives the delayed PRL message from peer node **210**, home node **240** responds with an IACK message indicating that home node **240** does not have a valid copy of the requested data.

In response to receiving the CNFL message from peer node **210** and the CNFLI message from peer node **220**, peer node **230** transmits a READ(**210**, **220***) message to home node **240**. The READ(**210**, **220***) message requests a copy of the data from memory controlled by home node **240**. The READ(**210**, **220***) message also indicates that the request for data from peer node **230** conflicts with requests from peer nodes **210** and **220** and that the request from peer node **220** requires invalidation (as indicated by the asterisk). Because peer node **230** is the first node with a conflicting request to send a READ message to home node **240**, peer node **230** is the first peer node to received a copy of the requested data and is the current owner of the requested data.

In response to the READ(**210**, **220***) message, home node provides the requested data to Peer node **230** with a DataE message. The DataE message causes peer node **230** to store the data in the E state. Alternatively, other data messages (e.g., DataF, DataS) could be used. Home node **240** retains the conflict list provided by peer node **230** to respond to subsequent READ/CNCL messages from peer nodes **210** and **220**.

As illustrated in FIG. **4***e*, in response to the IACK message from home node **240**, peer node **210** transmits a READ(**230***) message to home node **240**. The READ message does not indicate the conflict with the PRIL message from peer node **220** because peer node **210** has not received the PRIL message yet. If peer node **210** had received the PRIL message from peer node **220**, the READ message would indicate a conflict with peer node **220**.

In response to the READ(**230***) message, home node **240** sends a XFRI(**210**) message to peer node **230**, which causes peer node **230** to send the requested data to peer node **210**. The XFRI message also indicates to peer node **230** that a conflicting message from a peer node that has not yet received its data (peer node **220**) requires invalidation of the data after the data is sent to peer node **210**.

Peer node **230** sends the requested data to peer node **210** with a DataE(**220**) message, which causes peer node **210** to store the requested data in the E mode, and informs peer node **210** that the requesting message might conflict with a message from peer node **220**. Peer node **230** had detected the conflict with the message from peer node **220**. Before peer node **210** receives the requested data, the delayed PRIL message is received from peer node **220**.

Because peer node **210** has already delivered its list of conflicts to home node **240** and has not received the requested data from peer node **230**, peer node **210** stores the PRIL message received from peer node **220** in a buffer. In response to receiving the Data message from peer node **230** containing the conflict with peer node **220**, peer node **210** responds with a CNFL message indicating that peer node **210** has a conflict with the PRIL message from peer node **220**.

As illustrated in FIG. **4***f*, in response to the CNFL message from peer node **210**, peer node **230** sends a READ (**210**, **230***) message to home node **240**, which indicates a conflict with the PRL message from peer node **210** and the PRIL message from peer node **230**. Home node **240** responds to the READ(**210**,**230***) message with an ACK message to peer node **220** and a XFRI(**220**) message to peer node **210**. Peer node **210** sends the requested data to peer node **220** with a DataE(**230**) message. Peer node **220** previously detected the conflict with peer node **230**. Delivery of the ACK message to peer node **220** indicates that no further conflicts exist with the PRIL message from peer node **220**.

FIGS. **5***a* through **5***g* are conceptual illustrations of three-way conflicting requests for a uncached line. As illustrated in FIG. **5***a*, peer node **210** transmits a PRIL message to peer nodes **220** and **230** and to home node **240** to request a block of data. The PRIL message for home node **240** is delayed.

Soon after peer node **210** transmits a PRIL message, peer node **220** transmits a PRIL message to peer nodes **210** and **230** and to home node **240** requesting a copy of the same block of data.

As illustrated in FIG. **5***b*, peer node **230** responds to the PRIL message from peer node **210** with an IACK message. Peer node **230** indicates a delay in processing its response to the PRIL message from peer node **220**. Peer node **210** responds to the PRIL message from peer node **220** with a CNFLI message. Similarly, peer node **220** responds to the PRIL message from peer node **210** with a CNFLI message.

As illustrated in FIG. **5***c*, before nodes **210** and **220** receive a copy of the requested data, peer node **230** transmits a PRIL message to peer nodes **210** and **220** and to home node **240**. Peer nodes **210** and **220** respond to the PRIL message from peer node **230** with CNFLI messages. Due to its delay in processing the PRIL from peer node **220**, peer node **230** sends a CNFLI message, instead of an IACK message, to peer node **220** to indicate a conflict with the newly generated request for the same block of data by peer node **230**.

As illustrated in FIG. **5***d*, after receiving responses from all peer nodes, peer node **230** sends a READ(**210***,**220***) message to home node **240**. Home node **240** responds with a DataE message to provide the requested data to peer node **230** because peer node **230** is the first peer node to send a READ message to home node **240**. Peer node **230** is the current owner of the data.

As illustrated in FIG. **5***e*, after receiving responses from all peer nodes, peer node **210** sends a READ(**210***,**230***) message to home node **240**. Home node **240** also receives the delayed PRIL message from peer node **210**. Home node **240** responds to the PRIL message from peer node **210** with an IACK message, which indicates that home node **240** will not provide a copy of the requested data. This is because peer node **230** has a valid, cached copy of the requested data.

Home node **240** sends a XFRI(**220**) message to peer node **230** to cause peer node **230** to provide a copy of the requested data to peer node **210**. Peer node **230** is the second peer node to receive a copy of the requested data because peer node **230** is the second peer node to send a READ message to home node **240** requesting the data.

As illustrated in FIG. **5***f*, peer node **230** sends a copy of the requested data to peer node **220** with a DataE message. Peer node **210** sends a READ(**220***, **230***) message to home node **240**. Home node **240** responds to the READ message from peer node **210** by sending a XFRI(**210**) message to peer node **220**, which causes peer node **220** to send a copy of the requested data to peer node **210**.

As illustrated in FIG. **5***g*, peer node **220** sends a copy of the requested data to peer node **210** with a DataE message. Peer node **220** also invalidates the copy of the data stored by peer node **220**. Home node **240** sends an ACK message to peer node **210** to indicate that all requests for the block of data have been resolved and satisfied.

Example Systems to Support Non-Speculative Distributed Conflict Resolution

FIG. **6** is a block diagram of one embodiment of a node. Node **600** is illustrated with a single processor, cache memory, memory controller and memory; however, any number of any of these components can be included in a node. Further, additional and/or different components (e.g., a bus bridge) can also be included in a node.

Processor **610** can be any type of processor known in the art. In one embodiment, processor **610** includes cache memory **620**. In alternate embodiments, cache memory **620** is external to processor **610**, or additional cache memories can be included that are internal or external to processor **610**.

Memory controller **630** is coupled with cache memory **620** and memory **640**. Memory controller **630** operates as an interface between cache memory **620** and memory **640**. In one embodiment, memory controller **630** maintains cache coherency according to the cache coherency protocol described herein. Memory controller **630** interacts with other nodes via node links **650**. In an alternate embodiment, processor **610** interacts with memory controller **630** to maintain cache coherency as described herein and processor **610** interacts with other nodes via alternative node links **655**.

In one embodiment, node links **650** include a dedicated interface for each node with which node **600** interacts. In an alternate embodiment, node links **650** include a number of interfaces that is different than the number of nodes with which node **600** interacts. In one embodiment, node **600** interacts with one or more agents that represent multiple nodes.

FIG. **7** is one embodiment of a multiprocessor system. Multiprocessor system **700** is intended to represent a range of systems having multiple processors, for example, computer systems, real-time monitoring systems, etc. Alternative multiprocessor systems can include more, fewer and/or different components. In certain situations, the cache management techniques described herein can be applied to both single processor and to multiprocessor systems. Multiprocessor system **700** can be configured to operate as a multi-node system.

Multiprocessor system **700** includes bus system **710** or other communication device(s) to communicate information. Bus system **710** can include any number of buses and associated interconnection circuitry, for example, bus bridges. Processor **720** is coupled with bus system **710** to process information. Processor **720** can include cache memory **722**, for example a level zero (L**0**) cache memory, and cache controller **724**. In one embodiment, processor **720** is also coupled with cache **725**, which can be any type of cache memory. In an alternate embodiment, cache **725** can be coupled with bus system **710**. Other types of processor-cache configurations can also be used.

In one embodiment, cache controller **724** is coupled with cache memory **722** via cache memory interface **728**, which can be, for example, a bus internal to processor **720**. Cache controller is coupled with cache memory **725** via cache interface **726**, which provides an interface between processor **720** and an external cache memory.

Multiprocessor system **700** further includes processor **730** with cache memory **732** and cache controller **734**. Cache controller **734** is coupled with cache memory **732** via cache interface **738**. Similarly, cache controller **734** is coupled with cache memory **735** via cache interface **736**. In one embodiment, cache memory **735** is coupled with processor **730**.

While multiprocessor system **700** is illustrated with two processors, multiprocessor system **700** can include any number of processors and/or co-processors. Multiprocessor system **700** further includes memory system **740** coupled with bus system **710**. Memory system **740** can include any combination of dynamic (e.g., random access memory) and static (e.g., read-only memory, CD-ROM, disk storage, flash memory) memory devices and associated drives, where appropriate. The memory devices of memory system **740** are used to store information and instructions to be executed by processors of multiprocessor system **700**. Memory system **740** also can be used to store temporary variables or other intermediate information during execution of instructions by the processors.

Instructions can be provided to memory system **740** from a static or remote storage device, such as magnetic disk, a read-only memory (ROM) integrated circuit, CD-ROM, DVD, via a remote connection that is either wired or wireless, etc. In alternative embodiments, hard-wired circuitry can be used in place of or in combination with software instructions. Thus, execution of sequences of instructions is not limited to any specific combination of hardware circuitry and software instructions.

Multiprocessor system **700** further includes network interface **750** to provide access to a network, such as a local area network and/or the Internet. Network interface **750** can provide wireless and/or wired network interfaces, which can include communication of instructions to and/or from remote electronically-accessible media. An electronically-accessible medium includes any mechanism that provides (i.e., stores and/or transmits) content (e.g., computer executable instructions) in a form readable by an electronic device (e.g., a computer, a personal digital assistant, a cellular telephone).

For example, a machine-accessible medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals).

Multiprocessor system **700** can also include display device **760**, such as a cathode ray tube (CRT) or liquid crystal display (LCD), to display information. Input device(s) **770**, including, for example, a keyboard having alphanumeric and other keys, is typically coupled to bus **710** to communicate information and command selections to processors **720** and/or **730**. Another type of user input device is a cursor control device, such as a mouse, a trackball, or cursor direction keys to communicate direction information and command selections to processors **720** and **730** and to control cursor movement on display device **760**.

Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

APPENDIX A

Below are example descriptions of MESIF algorithms in pseudo-code format. Descriptions are packet-based; i.e., each routine is executed in response to an incoming or outgoing packet. Alternatively, the algorithms can be described as a reaction to a state change due to a packet being received or generated.

To simplify the descriptions, the following assumptions are made:

1. Each peer/requester node has a single caching agent;
2. Home nodes do not have caching agents; and
3. Algorithms for memory requests in home nodes can be more complex than indicated and handle all of the corner cases MESIF induces (more than one read, multiple epochs, write forwarding, etc.).

The case of a home node having a caching agent (as can occur in some embodiments) is derived from the given algorithms, namely, by combining the routines for received packets by internalizing routines involving transmissions to/from the home node by a local caching agent (or proxy).

In one embodiment, caches obey the following constraints:

1. Cache will generate PRL only if line is in I state.
2. Cache will generate PRIL only if line is in I or S state.
3. Cache will generate PWL only if line is in M state.
4. Cache may freely transition to I state from S, F, and E states.
5. Cache may freely transition to M state from E state (presuming a write occurred.)
6. Cache may otherwise transition only upon completion of a request it issued or in receipt of a request from a peer.

The base protocol described below covers only PRL, PRIL, and PWL requests and uses the conflict resolution method involving a conflict list being passed along with transferred data. Extensions and options to this base protocol are covered in the following sections.

BASE MESIF PROTOCOL

Request Generation

Invocation:

Cache generated new request for an (inactive) address

Algorithm:

Mark address as being active

If request is PRL or PRIL

Send request to all other peer nodes and to home node

If request is PWL

Send request to home node

Request Reception by Home Node

Invocation:

Request received at home node

Algorithm:

If request is PWL

Initiate write to memory

(Handle forwarding, canceling outstanding reads, etc.)

Send ACK back to requester

If request is PRL or PRIL

Initiate read from memory

(Buffer data if read finishes before a READ is received, etc.)

Request Reception by Peer Node

Invocation:

Request (PRL or PRIL) received at a peer node

Algorithm:

Snoop Responses

Invocation:

Request submitted (as a snoop) to cache for proper response generation

Algorithm:

Look up response and next state in table below based upon current cache state and incoming request type (next state of S/I means the cache can move the line to either state; note: still send DATA_F for PRLs even if invalidating local copy—see PRL DATA_E/M Reponse option below)

	Response		Next State	
State	PRL	PRIL	PRL	PRIL
I	IACK	IACK	I	I
S	SACK	IACK	S/I	I
F	DATA_F	DATA_E	S/I	I
E	DATA_F	DATA_E	S/I	I
M	DATA_F	DATA_M	S/I	I

Gathering Responses

Invocation:

PRL/PRIL request received a response from a peer node

Algorithm:

Cancel at Home Node

Invocation:

Home node received a CNCL (contains forwarding node and conflict list)

Algorithm:

Read Request at Home Node

Invocation:

Home node received a READ (contains conflict list)

Algorithm:

Transfer Reception

Invocation:

Requester received a XFR or XFRI (contains target node)

Algorithm:

Transferred Data Reception

Invocation:

Requester received DATA (contains conflict list) as a result of a XFR

Algorithm:

Send data to processor

Incorporate conflict list into current conflict list

If buffering requests matching entry in conflict list

Respond with CNFL to each matching request

Forwarding DACK

Invocation:

Forwarding node received a DACK

Algorithm:

Unmark address as being forwarded

Service buffered requests per algorithm for receiving peer requests

Request ACK

Invocation:

Requester received an ACK from home node

Algorithm:

PIL Request

In the protocol algorithms above the only way for a node to move a cache line from the F state to the E state is to invalidate the line (change line to I state) and then request a PRIL. This involves a DATA transfer.

To support direct F->E transition, a PIL request can be used. This request, sent to all peer nodes and the home node, and causes other caches to invalidate their shared copies of the line. To prevent in-flight PRILs and/or PRLs from interfering with the transition, PILs can be given higher priority.

Changes to Base Protocol:

Request Generation

Invocation:

Cache generated new request for an (inactive) address

Algorithm:

Request Reception by Peer Node

Invocation:

Request (PRL or PRIL) received at a peer node

Only change to algorithm is to buffer request if there is an active PIL request, as is done with an active PWL.

PIL Request Reception by Peer Node

Invocation:

PIL request received at a peer node

Algorithm:

Send PIL snoop to cache

Snoop Responses

Same algorithm used with new Response/Next-State table (no entries for F, E, and M with PILs because requester in F and F, E, and M are mutually exclusive)

	Response			Next State		
State	PRL	PRIL	PIL	PRL	PRIL	PIL
I	IACK	IACK	IACK	I	I	I
S	SACK	IACK	IACK	S/I	I	I
F	DATA_F	DATA_E		S/I	I	
E	DATA_F	DATA_E		S/I	I	
M	DATA_F	DATA_M		S/I	I	

Gathering Responses

Invocation:

PIL request received a response from a peer node

Algorithm:

M->S PWL and Immediate Response

A performance issue with PRLs hitting a line in M state is the need to writeback (issue a PWL) before forwarding the data. With some subtle changes, the data can be forwarded and written back simultaneously. The home node does not send the DACK until it has received both the PWL request and the CNCL from the requester/winner.

Changes to Base Protocol:

Request Reception by Home Node

Invocation:

Request received at home node

Algorithm:

Snoop Responses

Invocation:

Request submitted (as a snoop) to cache for proper response generation

Algorithm:

Gathering Responses

Algorithm:

Cancel at Home Node

Invocation:

Home node received a CNCL (contains forwarding node and conflict list)

Algorithm:

Only difference is in determining whether to send a DACK:

Forwarding DACK

No differences. The issued PWL is treated as a one-shot packet (or a request finished by the DACK.)

FM State

Another alternative for PRLs hitting a line in M state is to introduce the FM state to MESIF. This state indicates shared copies of a modified line. As with the M state, the data must be written back (PWL) if evicted from the cache. As with the F state, the data cannot be changed and the node responds with hits for read requests on the line.

When a node with the line in M state receives a PRL, it responds with DATA_FM instead of issuing a PWL and responding with DATA_F.

Transitioning from FM to M is not allowed except via a PIL. Changing from FM to E directly is not allowed.

Changes to Base Protocol:

Snoop Response

Invocation:

Request submitted (as a snoop) to cache for proper response generation

Algorithm:

Look up response and next state in table below based upon current cache state and incoming request type (next state of S/I means the cache can move the line to either state; note: still send DATA_F(M) for PRLs even if invalidating local copy—see PRL DATA_E/M Reponse option below)

	Response			Next State		
State	PRL	PRIL	PIL	PRL	PRIL	PIL
I	IACK	IACK	IACK	I	I	I
S	SACK	IACK	IACK	S/I	I	I
F	DATA_F	DATA_E		S/I	I	
E	DATA_F	DATA_E		S/I	I	
M	DATA_FM	DATA_M		S/I	I	
FM	DATA_FM	DATA_M		S/I	I	

Conflictless Data

Sending a conflict list with transferred data is problematic for hardware. It is possible to avoid this conflict list if requests in the middle of transfer chains know they are in the middle and are allowed to respond to buffered requests (with IACK/SACK) after receiving transferred data. This allows all other conflicting nodes to make further progress, thus getting remaining READs to the home node.

With this option, requests (PRL and PRIL, i.e., read requests) go through four phases:

1) Send phase—sending requests
2) Gather phase—gathering responses (followed by sending READ or CNCL)
3) Data phase—waiting for data
4) Hold phase—in middle of conflict chain, hold data until XFR, send IACK/SACK to buffered and incoming requests

In this embodiment, a request will know it is in the middle of the chain if there is no piggybacked ACK on the transfered DATA. Only this hold phase is different from the base protocol. In fact, the base protocol's data phase either stays the same (for conflictless requests or requests at the end of the epoch/conflict chain) or is divided into two phases, the first still being the data phase and the second now being the hold phase ending when the XFR is received.

Changes to Base Protocol:

Request Reception by Peer Node

Only change to algorithm is to check if there is an active request in its hold phase:

Gathering Response

Only change to this algorithm is that the request is complete if it sends a CNCL and its conflict list is empty. In other words, the system did a cache-to-cache transfer and there were no conflicts; the only thing left to do is notify home node, which does not need to be ACK'd.

Note: a CNCL'ing request (with conflicts) stays in data phase waiting for a XFR, i.e., it does not enter the hold phase.

Cancel at Home Node

Invocation:

Home node received a CNCL (contains forwarding node and conflict list)

Algorithm:

Read Request at Home Node

Invocation:

Home node received a READ (contains conflict list)

Algorithm:

Transfer Reception

A change here (in addition to handling a XFR+ACK) is determining if during the hold phase an IACK response was spoofed for a PRIL. If so, the line is invalidated using the snoop.

Invocation:

Requester received a XFR, XFR+ACK, or XFRI (contains target node)

Algorithm:

Transferred Data Reception

Invocation:

Requester received DATA or DATA+ACK as a result of a XFR (requester is in data phase, so it knows this is via a XFR)

Algorithm:

PRL DATA_E/M Response

When a PRL snoop hits a cache line, it has to respond with DATA_F to maintain correctness, regardless of whether the cache transitions the line to S or I. It is possible to support sending a DATA_E when transitioning to I, but it requires an additional communication with the cache to let it know that the E state it received needs to be demoted to F. Basically, the algorithm is that if the node has already received a DATA_E then receives an SACK, it must change the cache state from E to F.

What is claimed is:

1. A node comprising:

a memory to store original and written-back copies of data corresponding to a predetermined range of addresses;

a cache memory to store copies of requested blocks of data; and

a control circuit coupled with the memory and with the cache memory to receive, from a peer node, a message having a request for a copy of a block of data stored in the memory and a list of nodes, if any, having issued conflicting request messages, the control circuit to provide a copy of the requested data to the requesting peer node, the control circuit further to receive a subsequent, conflicting request from a node from the list, and to send a message to the previous requesting node to send a copy of the requested data to the subsequently requesting node.

2. The node of claim **1** wherein the block of data comprises a cache line.

3. The node of claim **1** wherein the control circuit comprises a processor.

4. The node of claim **1** wherein the control circuit comprises a memory controller.

5. The node of claim **1** wherein the control circuit further provides a copy of the list to the requesting peer node.

6. A method comprising:

receiving, from a peer node, a message having a request for a copy of a block of data stored in the memory and list of nodes, if any, having issued conflicting request messages;

providing a copy of the requested data to the requesting peer node;

receiving a subsequent, conflicting request from a node from the list; and

sending a message to previous requesting node to send a copy of the requested data to the subsequently requesting node.

7. The method of claim **6** wherein the block of data comprises a cache line.

8. The method of claim **6** further comprising sending an acknowledgement message to the forwarding node.

9. An apparatus comprising:

means for receiving, from a peer node, a message having a request for a copy of a block of data stored in the memory and a list of nodes, if any, having issued conflicting request messages;

means for providing a copy of the requested data to the requesting peer node;

means for receiving a subsequent, conflicting request from a node from the list; and

means for sending a message to the previous requesting node to send a copy of the requested data to the subsequently requesting node.

10. The apparatus claim **9** wherein the block of data comprises a cache line.

11. The apparatus of claim **9** further comprising means for sending an acknowledgement message to the forwarding node.

12. A method comprising:

receiving, from a node in a multi-node system, a message indicating one or more nodes that have issued conflicting requests for a copy of a block of data;

receiving, from each the nodes that have issued the conflicting requests, a message indicating a conflict with another request;

sending a copy of the requested data to a first node sending a message indicating a conflict; and

sending a message to a current owner node to cause the current owner node to send a copy of the requested data to another node in response to a subsequent message indicating a conflict.

13. The method of claim **12** wherein the block of data comprises a cache line.

14. The method of claim **12** wherein a current owner comprises a node having a copy of the block of data received from a corresponding home node or a node having a most recently forwarded copy of the block of data from a previously current owner.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO.	: 6,954,829 B2	Page 1 of 1
APPLICATION NO.	: 10/326232	
DATED	: October 11, 2005	
INVENTOR(S)	: Beers et al.	

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 18, at line 41 after "each", insert --of--.

Signed and Sealed this

Twenty-second Day of January, 2008

Jon W. Dudas

JON W. DUDAS
Director of the United States Patent and Trademark Office