25 APR 2005

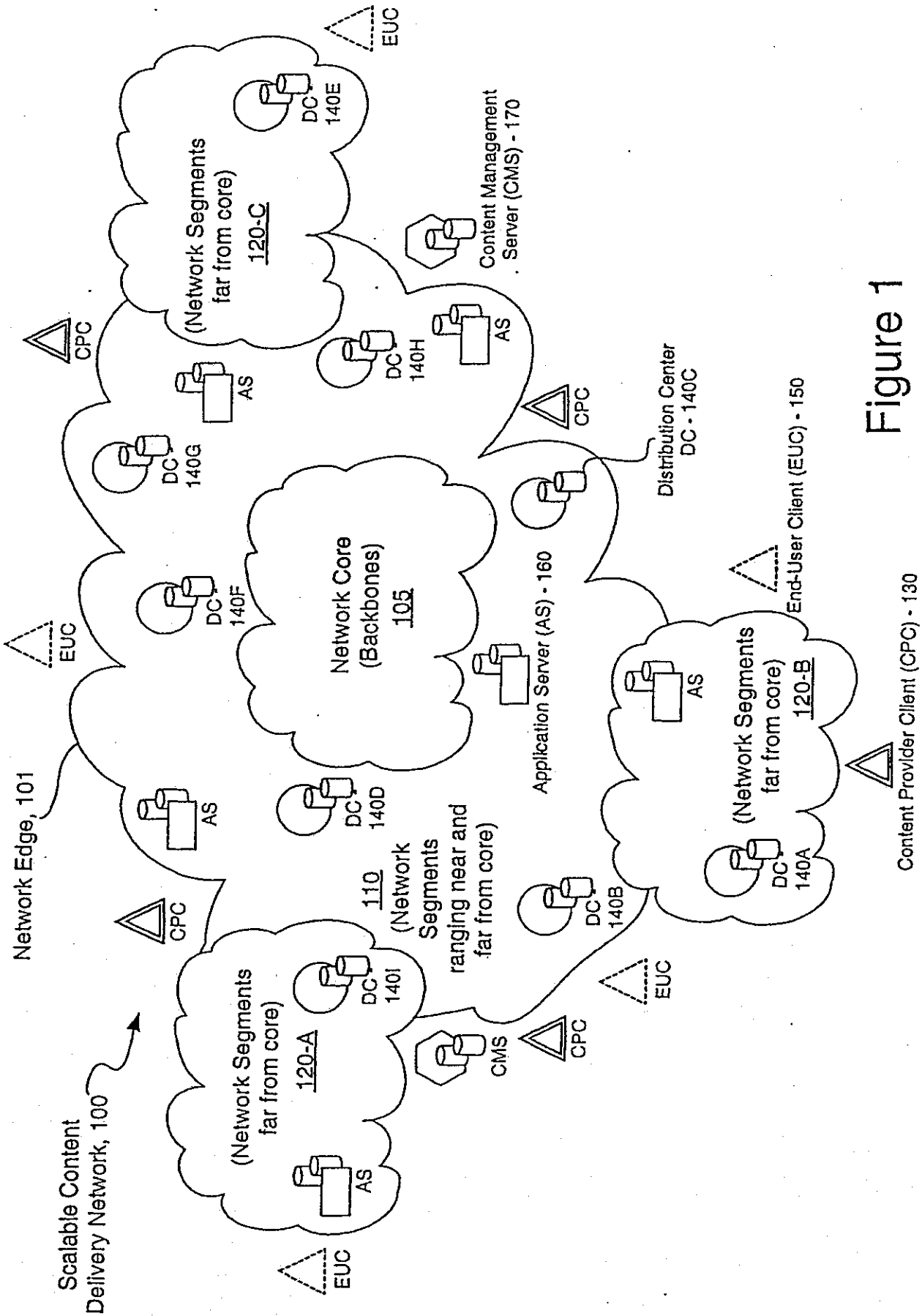


Figure 1

25 APR 2005

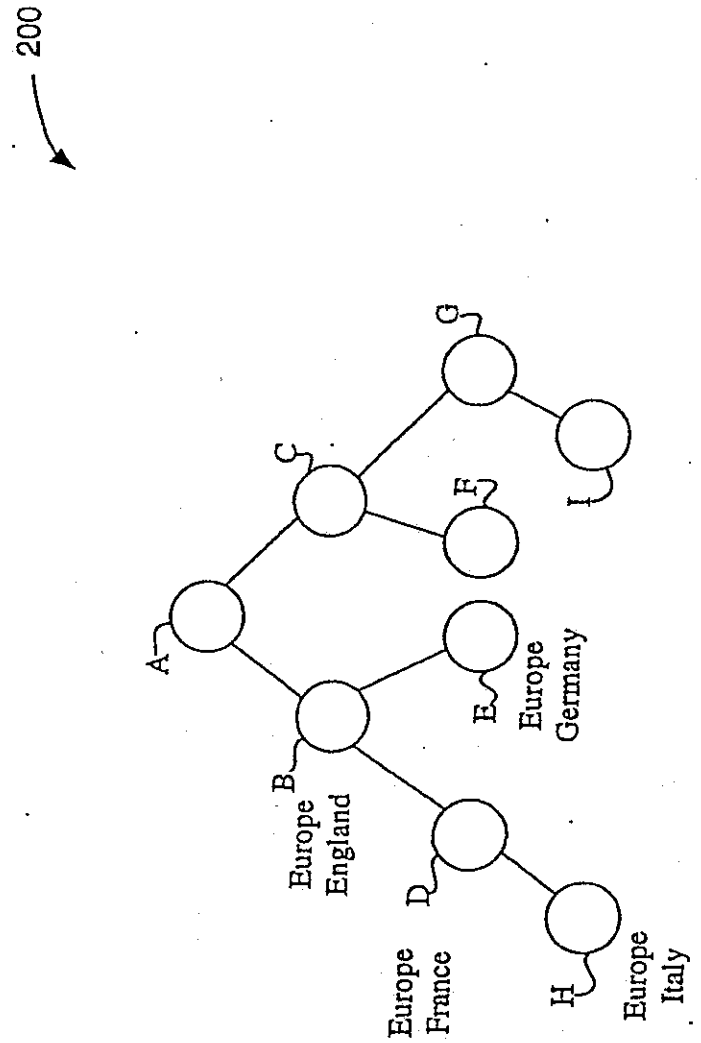


Figure 2

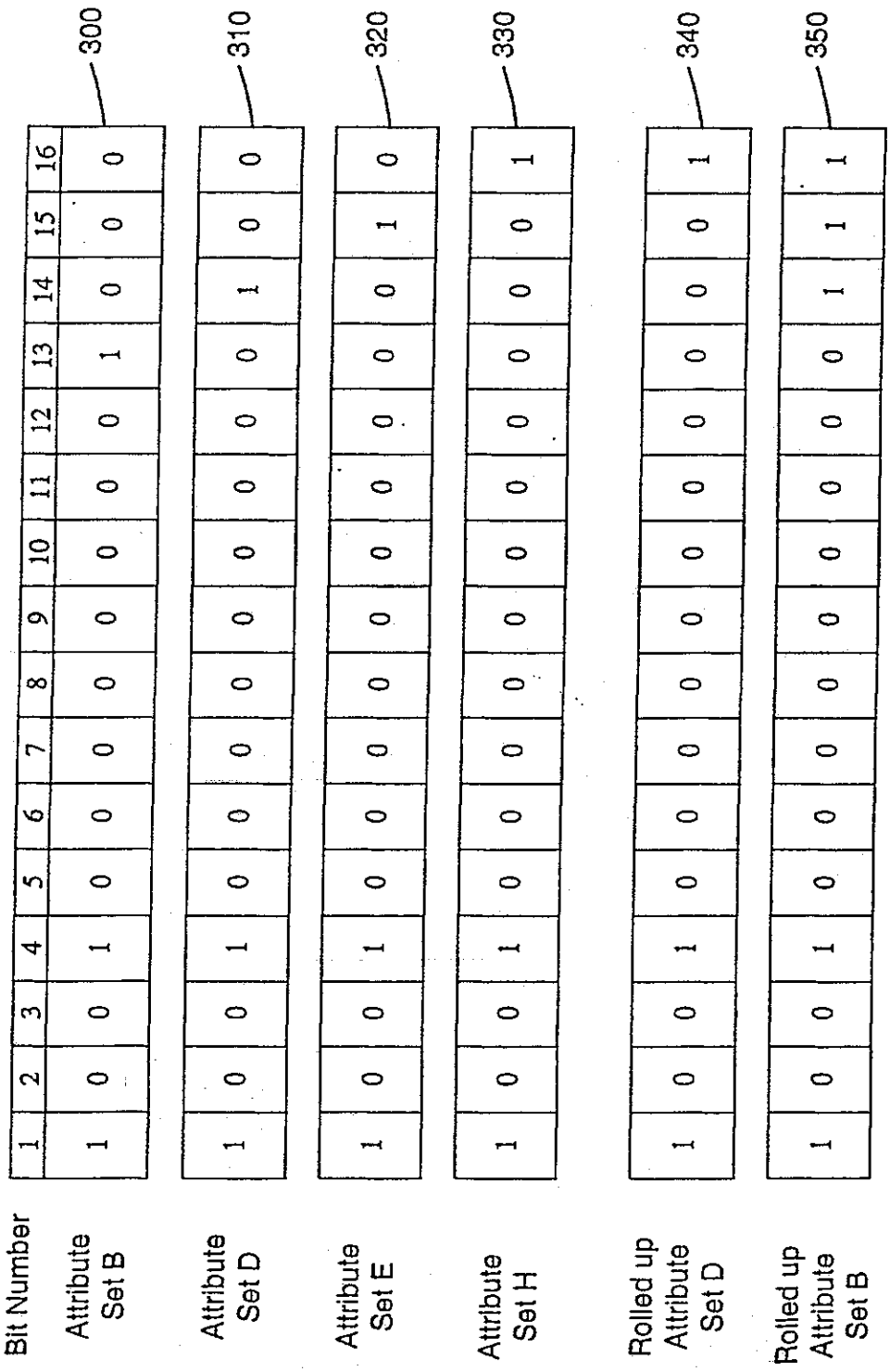

Bit Number
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Attribute Set B
1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
300
Attribute Set D
1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
310
Attribute Set E
1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
320
Attribute Set H
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
330
Rolled up Attribute Set D
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
340
Rolled up Attribute Set B
1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1
350

Figure 3

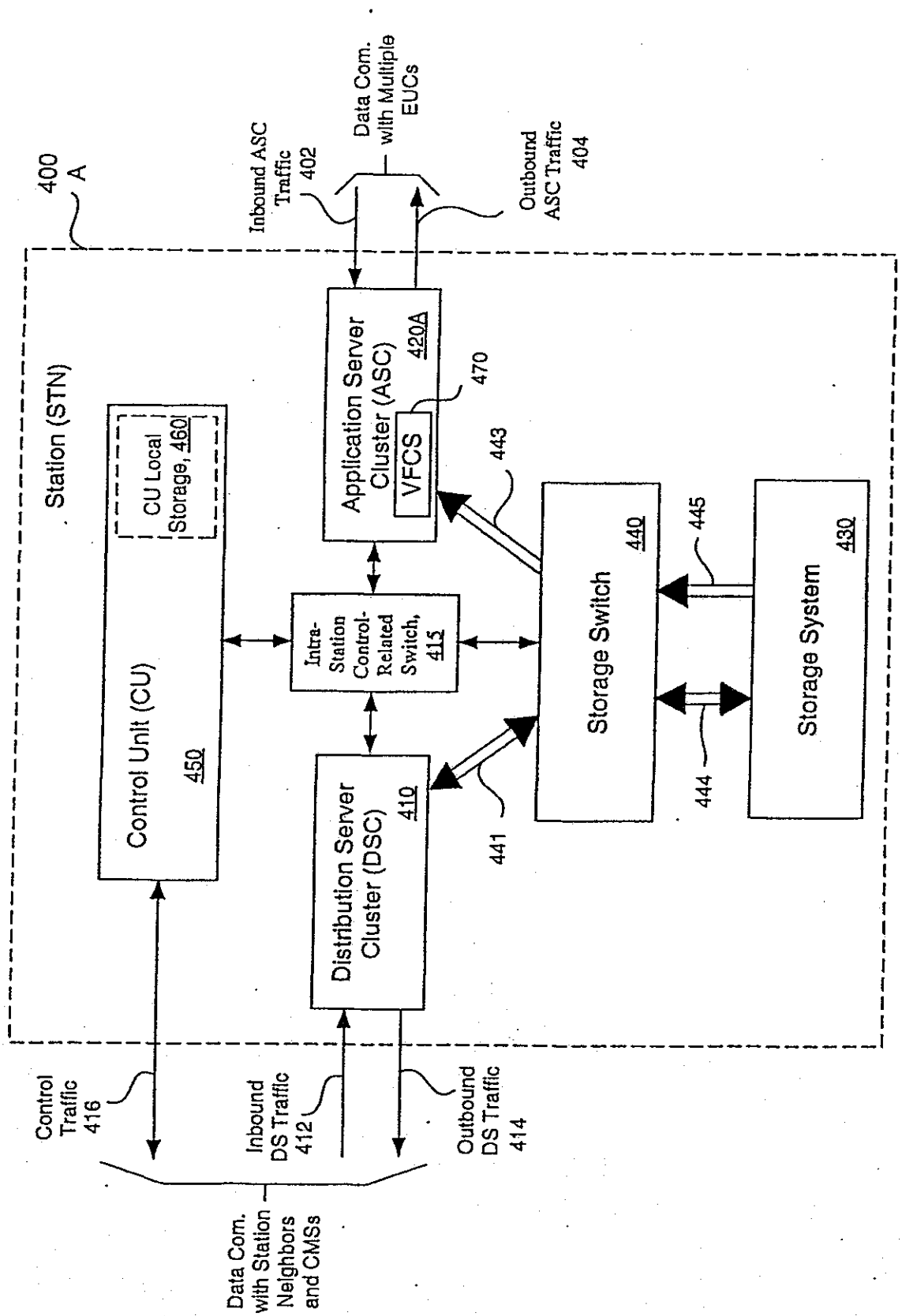

400
A
Station (STN)
Inbound ASC Traffic 402
Data Com. with Multiple EUCs
Outbound ASC Traffic 404
Application Server Cluster (ASC) 420A
VFCS
470
443
CU Local Storage, 460
Control Unit (CU) 450
Intra-Station Control-Related Switch, 415
Storage Switch 440
445
Storage System 430
Distribution Server Cluster (DSC) 410
441
444
Control Traffic 416
Inbound DS Traffic 412
Outbound DS Traffic 414
Data Com. with Station Neighbors and CMSs

Figure 4A

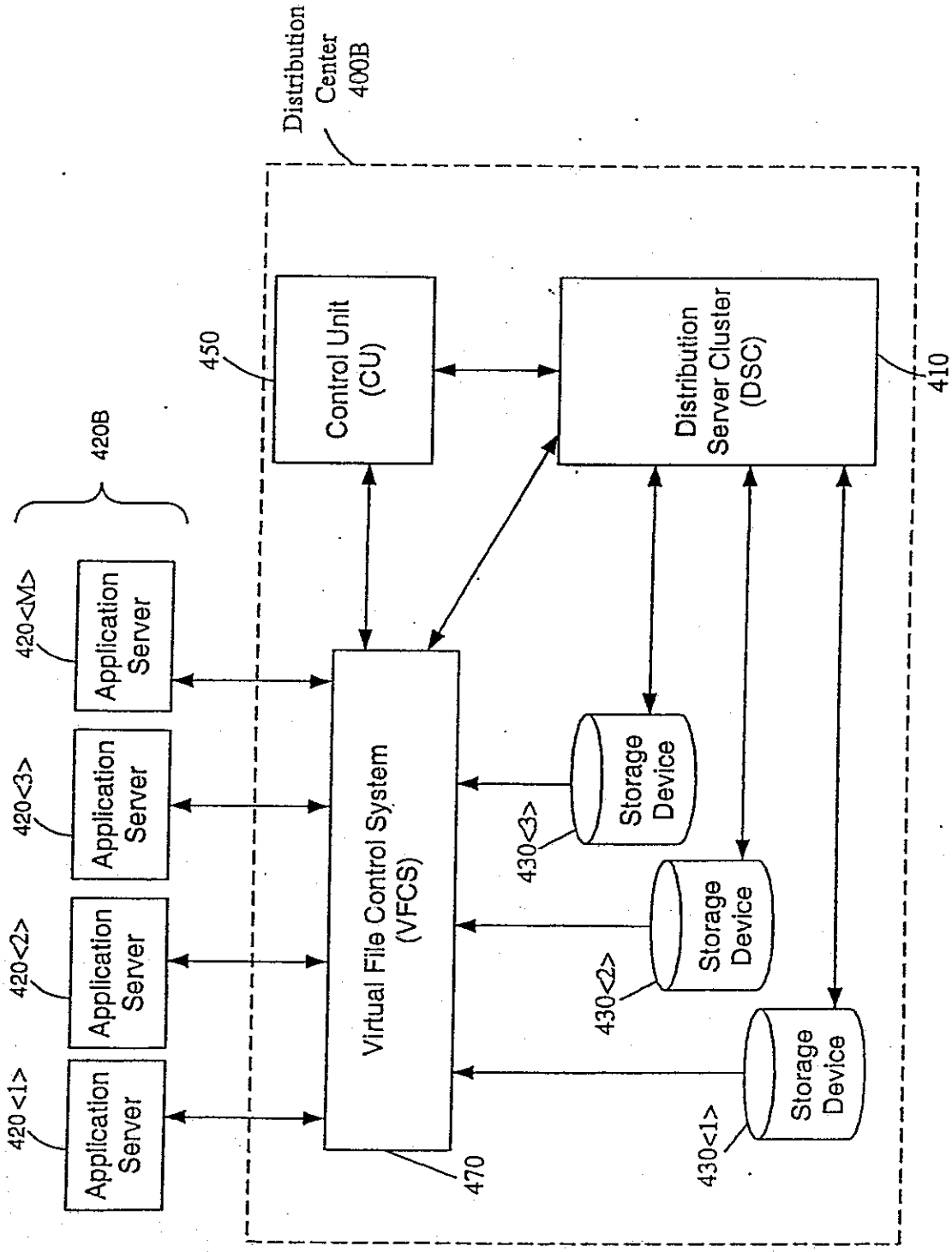

Distribution Center 400B
Control Unit (CU)
450
Distribution Server Cluster (DSC)
410
420B
Application Server
420<M>
Application Server
420<3>
Application Server
420<2>
Application Server
420<1>
Virtual File Control System (VFCS)
470
Storage Device
430<3>
Storage Device
430<2>
Storage Device
430<1>

Figure 4B

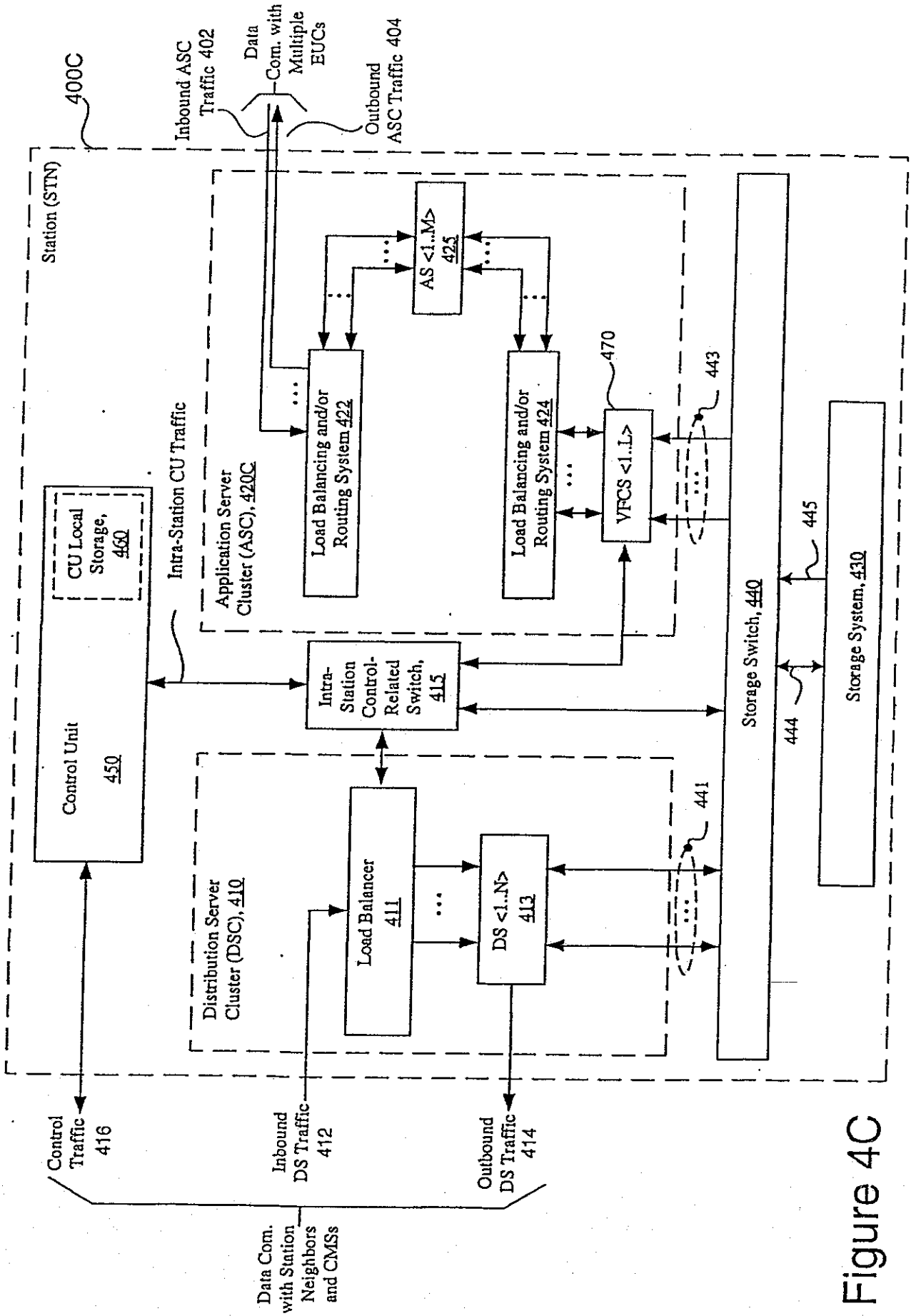

400C
Station (STN)
Inbound ASC Traffic 402
Data Com. with Multiple EUCs
Outbound ASC Traffic 404
Application Server Cluster (ASC), 420C
Load Balancing and/or Routing System 422
AS <1..M> 425
Load Balancing and/or Routing System 424
VFCS <1..L>
470
443
445
Storage System, 430
Storage Switch, 440
444
Intra-Station CU Traffic
CU Local Storage, 460
Control Unit 450
Intra-Station Control-Related Switch, 415
Distribution Server Cluster (DSC), 410
Load Balancer 411
DS <1..N> 413
441
Control Traffic 416
Inbound DS Traffic 412
Outbound DS Traffic 414
Data Com. with Station Neighbors and CMSs

Figure 4C

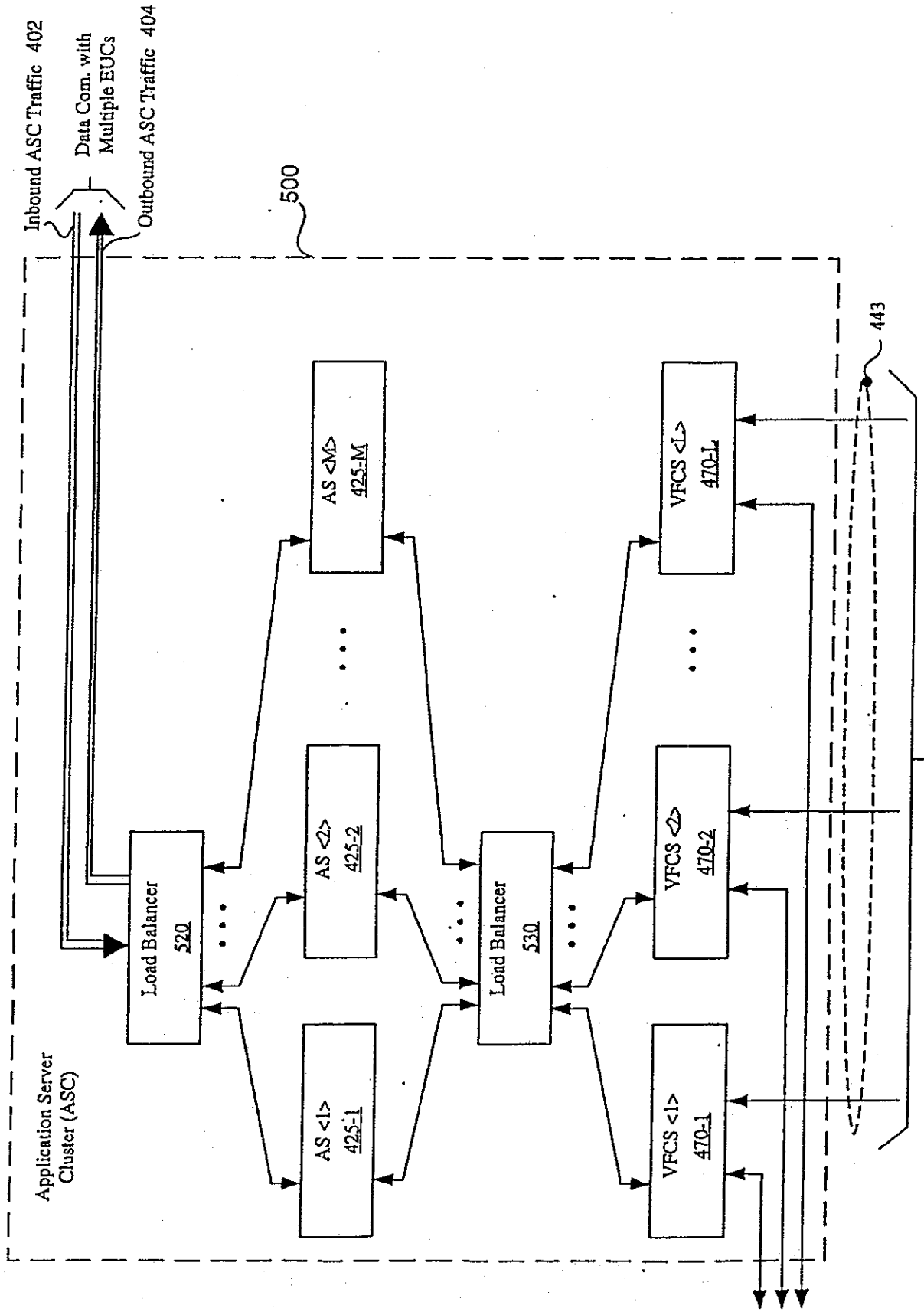

Inbound ASC Traffic 402
Data Com. with Multiple EUCs
Outbound ASC Traffic 404
500
Application Server Cluster (ASC)
Load Balancer 520
Load Balancer 530
AS <1> 425-1
AS <2> 425-2
AS <M> 425-M
VFCS <1> 470-1
VFCS <2> 470-2
VFCS <L> 470-L
443
From Switch 440 and Storage System 430

Figure 5A

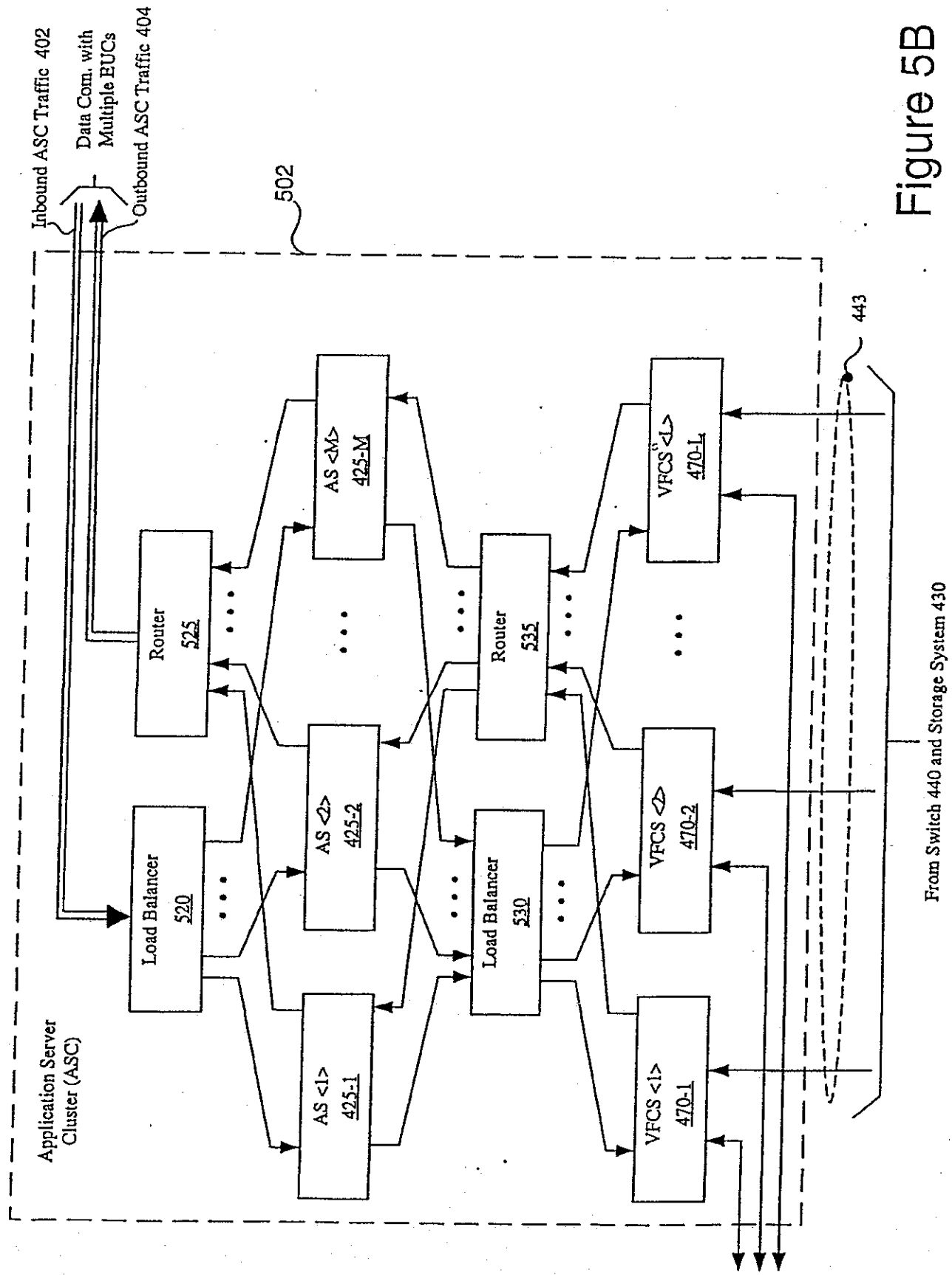

Inbound ASC Traffic 402
Data Com. with Multiple EUCs
Outbound ASC Traffic 404
502
Application Server Cluster (ASC)
Load Balancer 520
Router 525
AS <1> 425-1
AS <2> 425-2
AS <M> 425-M
Load Balancer 530
Router 535
VFCS <1> 470-1
VFCS <2> 470-2
VFCS <L> 470-L
443
From Switch 440 and Storage System 430

Figure 5B

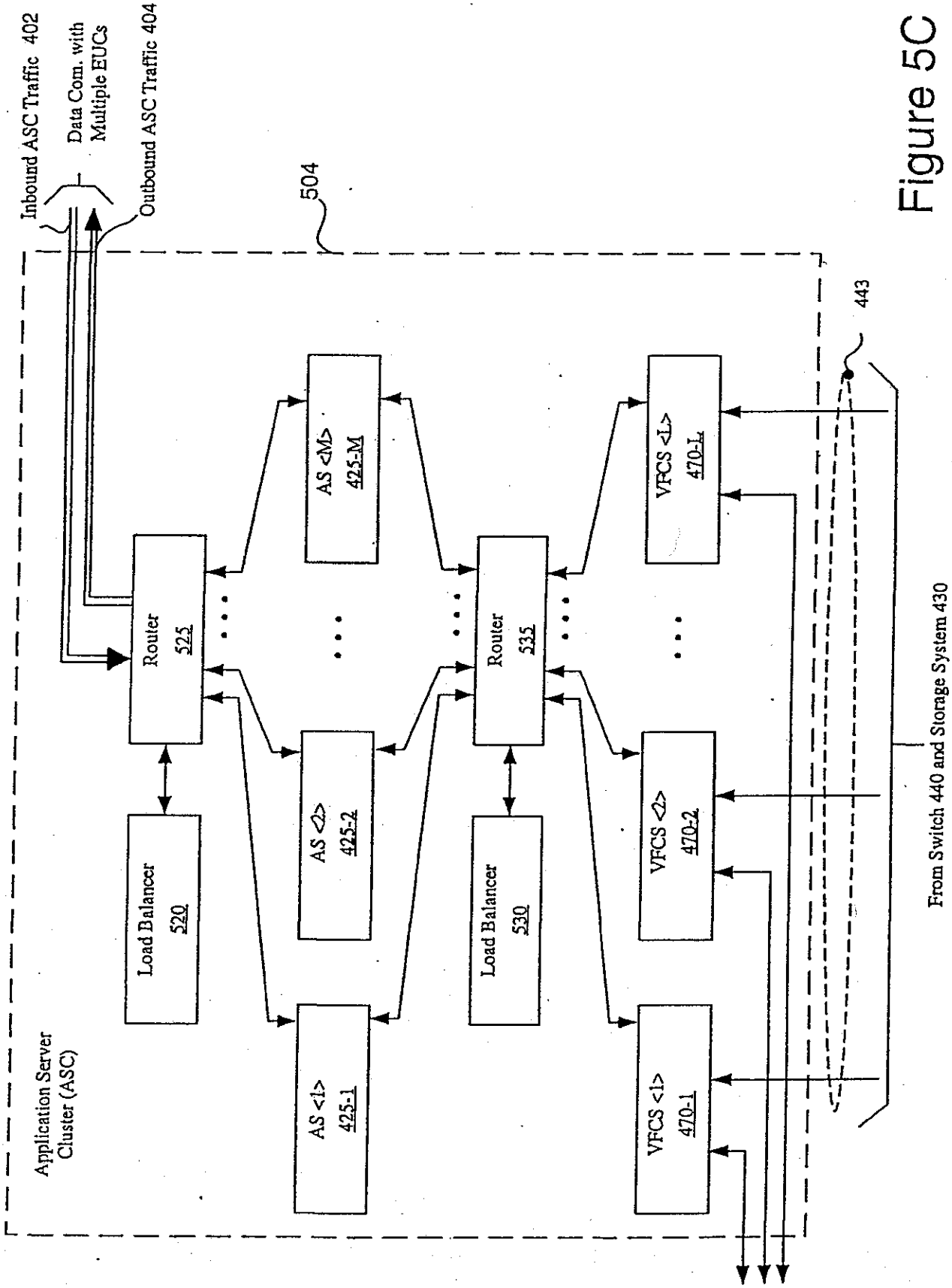

Inbound ASC Traffic 402
Data Com. with Multiple EUCs
Outbound ASC Traffic 404
504
443
Router 525
Router 535
Load Balancer 520
Load Balancer 530
AS <1> 425-1
AS <2> 425-2
AS <M> 425-M
VFCS <1> 470-1
VFCS <2> 470-2
VFCS <L> 470-L
Application Server Cluster (ASC)
From Switch 440 and Storage System 430

Figure 5C

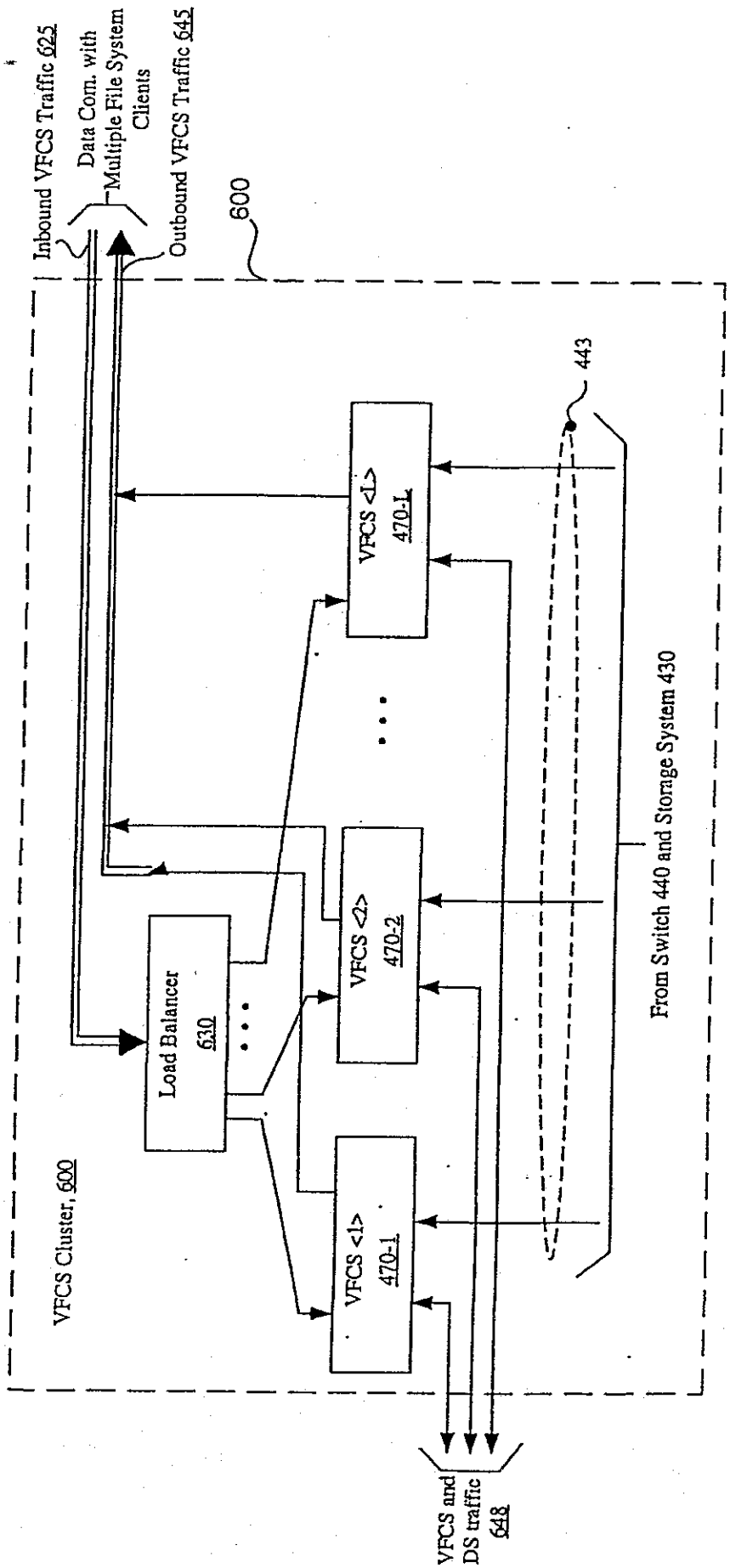

Inbound VFCS Traffic 625
Data Com. with Multiple File System Clients
Outbound VFCS Traffic 645
600
443
VFCS <L> 470-L
VFCS <2> 470-2
VFCS <1> 470-1
Load Balancer 630
From Switch 440 and Storage System 430
VFCS Cluster, 600
VFCS and DS traffic 648

Figure 6

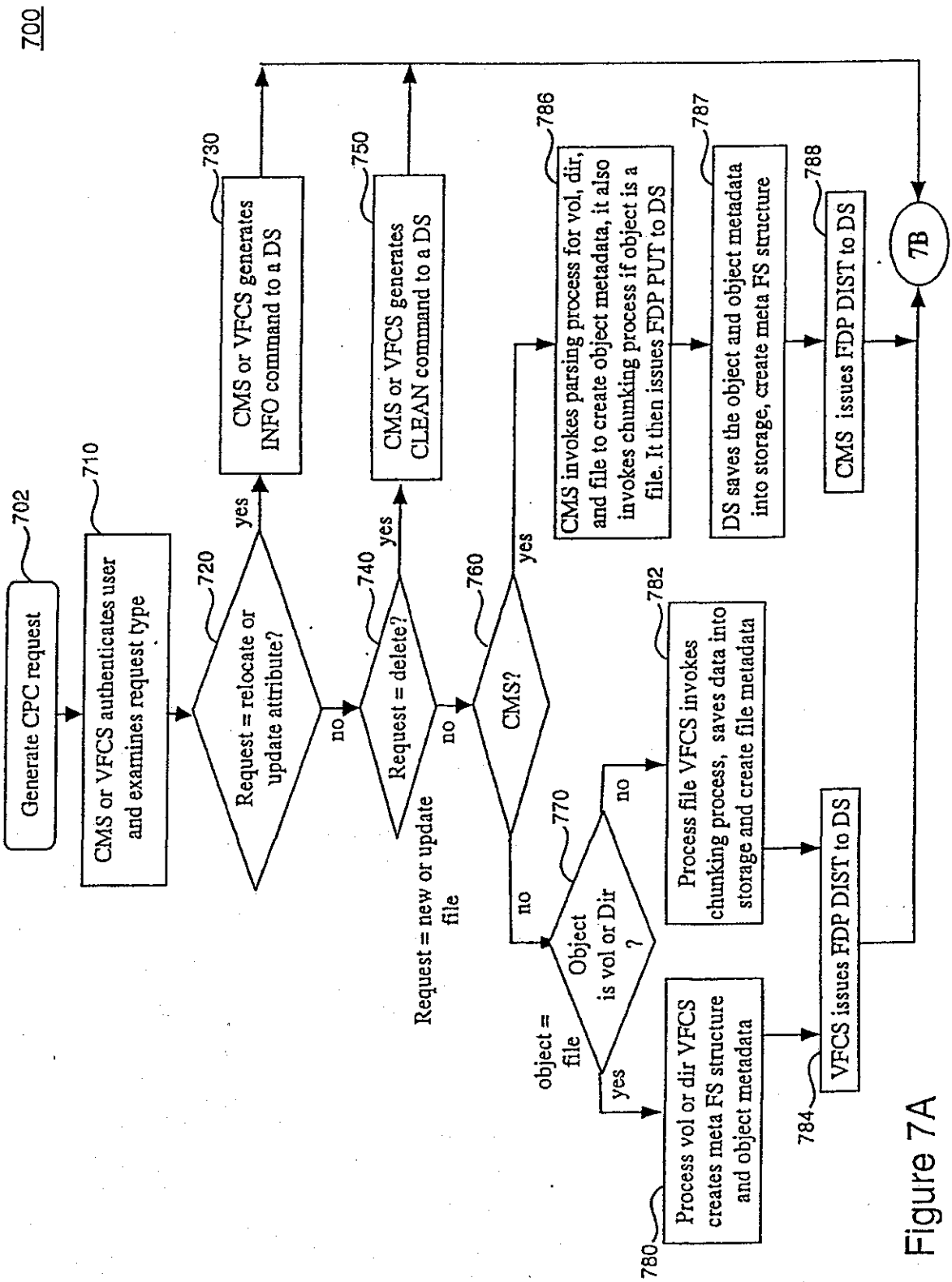

700
Generate CPC request
702
CMS or VFCS authenticates user and examines request type
710
Request = relocate or update attribute?
720
yes
CMS or VFCS generates INFO command to a DS
730
no
Request = delete?
740
yes
CMS or VFCS generates CLEAN command to a DS
750
Request = new or update file
no
CMS?
760
yes
CMS invokes parsing process for vol, dir, and file to create object metadata, it also invokes chunking process if object is a file. It then issues FDP PUT to DS
786
DS saves the object and object metadata into storage, create meta FS structure
787
CMS issues FDP DIST to DS
788
7B
no
Object is vol or Dir ?
770
object = file
yes
Process file VFCS invokes chunking process, saves data into storage and create file metadata
782
no
Process vol or dir VFCS creates meta FS structure and object metadata
780
VFCS issues FDP DIST to DS
784

Figure 7A

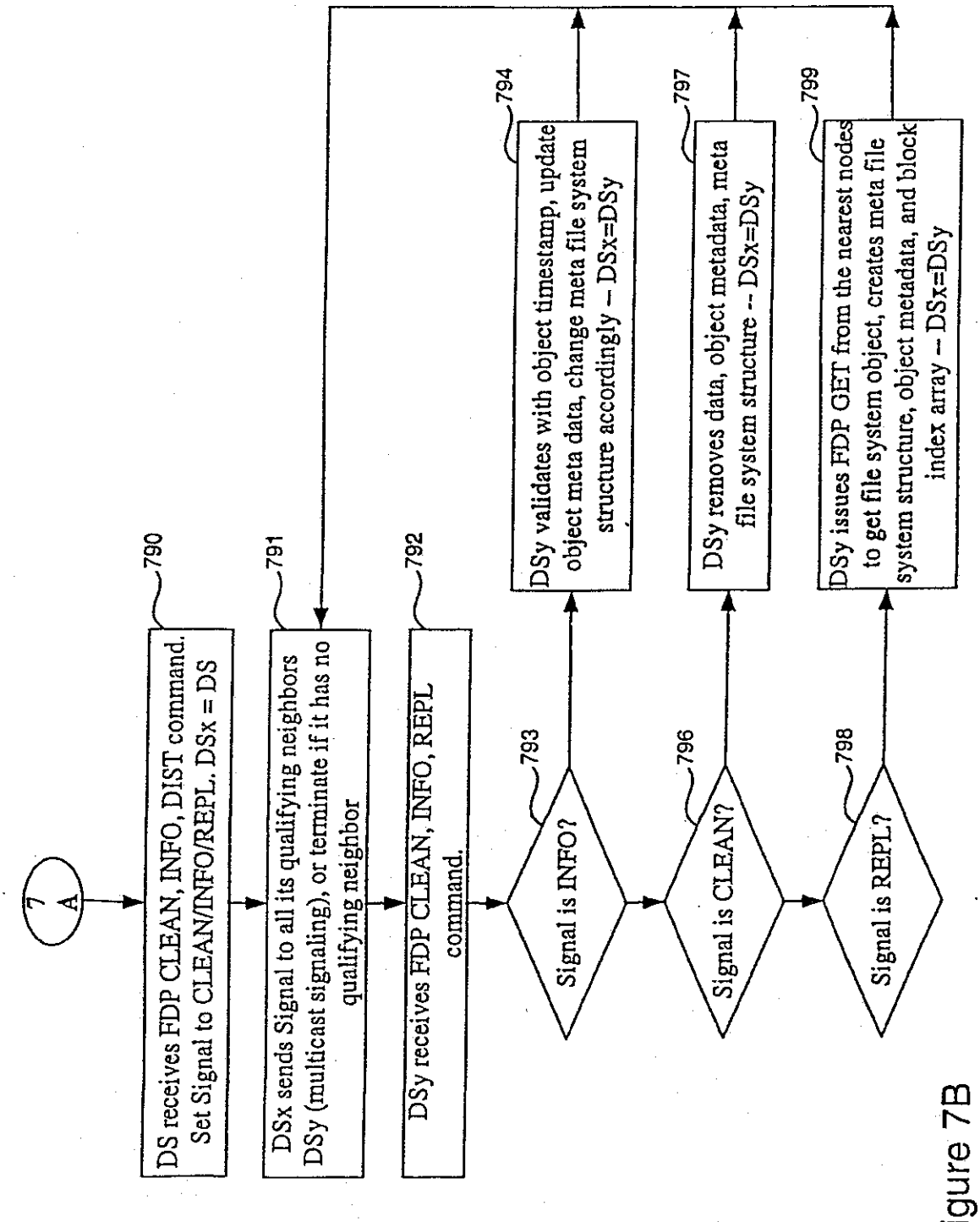

7
A
790
DS receives FDP CLEAN, INFO, DIST command. Set Signal to CLEAN/INFO/REPL, DSx = DS
791
DSx sends Signal to all its qualifying neighbors DSy (multicast signaling), or terminate if it has no qualifying neighbor
792
DSy receives FDP CLEAN, INFO, REPL command.
793
Signal is INFO?
794
DSy validates with object timestamp, update object meta data, change meta file system structure accordingly -- DSx=DSy
796
Signal is CLEAN?
797
DSy removes data, object metadata, meta file system structure -- DSx=DSy
798
Signal is REPL?
799
DSy issues FDP GET from the nearest nodes to get file system object, creates meta file system structure, object metadata, and block index array -- DSx=DSy

Figure 7B

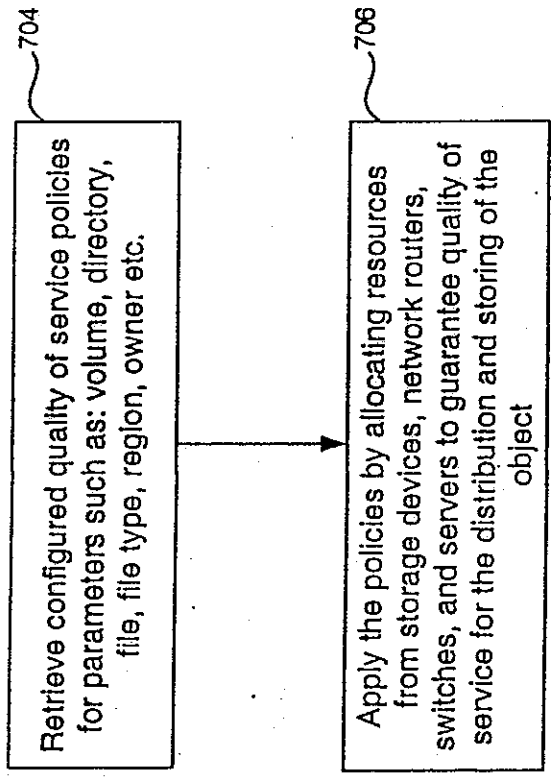

704
Retrieve configured quality of service policies for parameters such as: volume, directory, file, file type, region, owner etc.
706
Apply the policies by allocating resources from storage devices, network routers, switches, and servers to guarantee quality of service for the distribution and storing of the object

Figure 7C

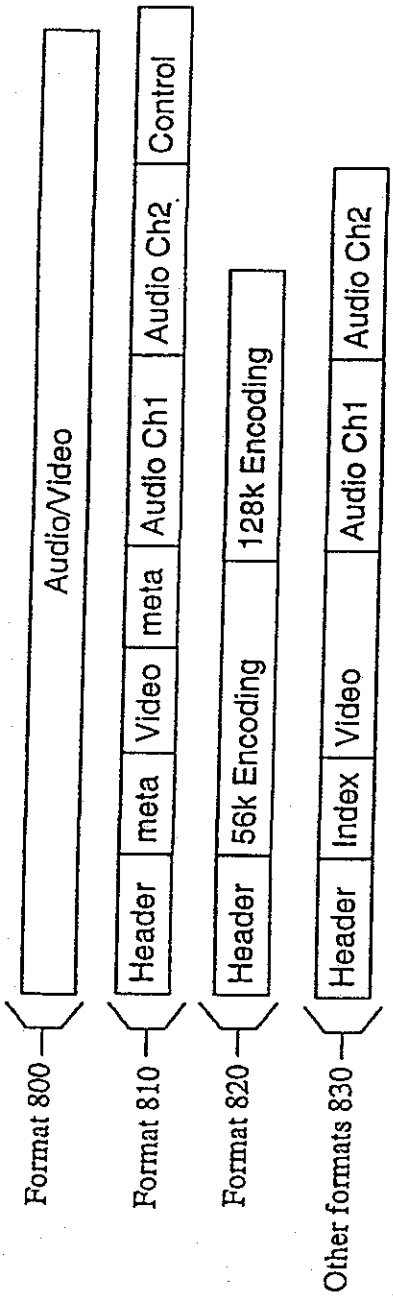

Format 800
Audio/Video
Format 810
Header
meta
Video
meta
Audio Ch1
Audio Ch2
Control
Format 820
Header
56k Encoding
128k Encoding
Other formats 830
Header
Index
Video
Audio Ch1
Audio Ch2

Figure 8

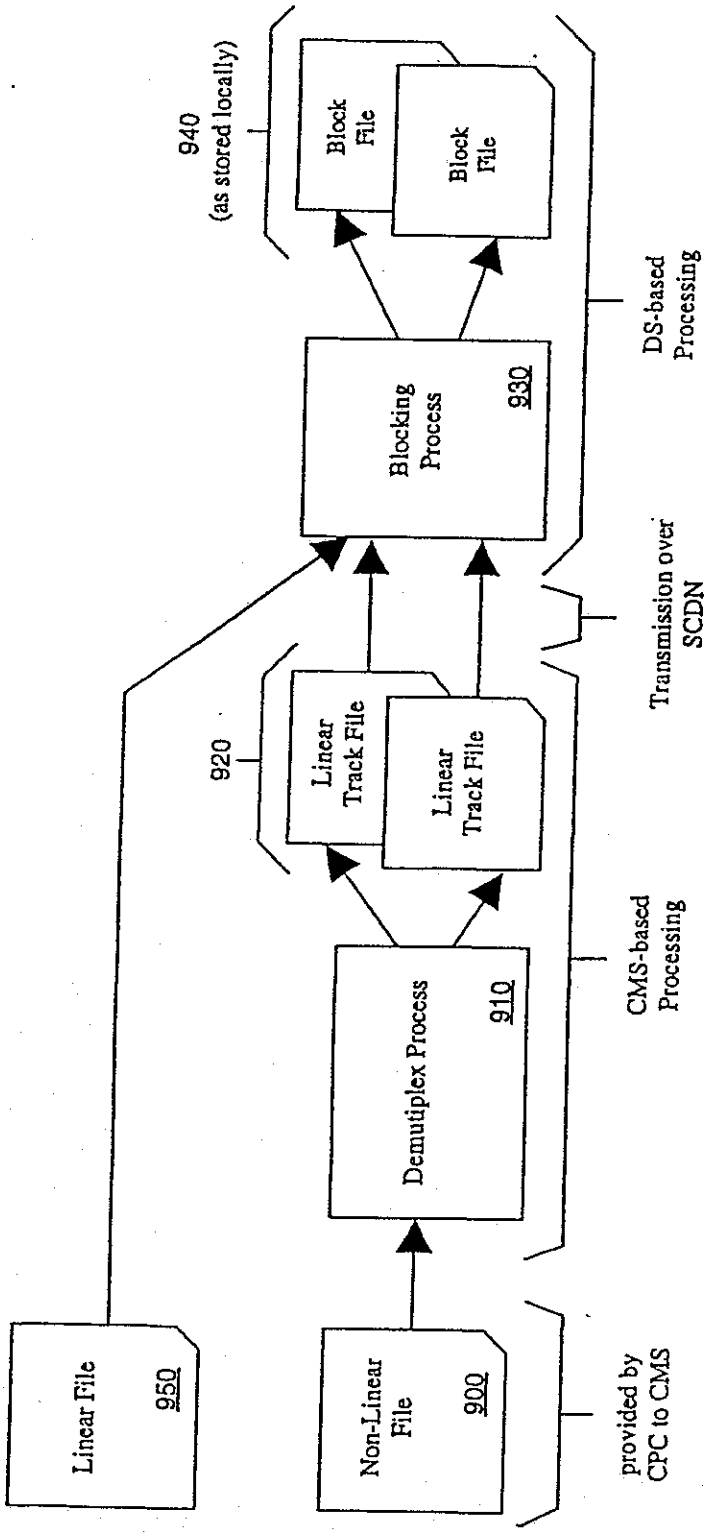

Linear File
950
Non-Linear File
900
provided by CPC to CMS
Demutiplex Process
910
CMS-based Processing
920
Linear Track File
Linear Track File
Transmission over SCDN
Blocking Process
930
DS-based Processing
940
(as stored locally)
Block File
Block File

Figure 9

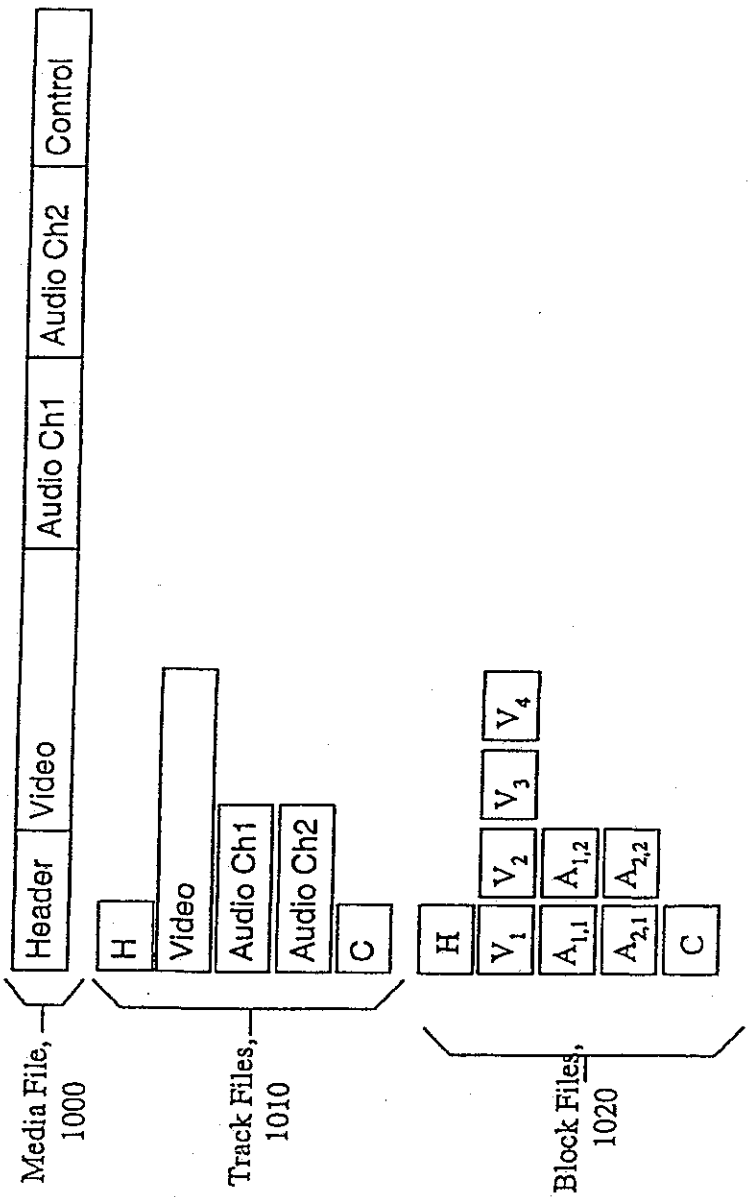

Media File, 1000
Header
Video
Audio Ch1
Audio Ch2
Control
Track Files, 1010
H
Video
Audio Ch1
Audio Ch2
C
Block Files, 1020
H
V1
V2
V3
V4
A1,1
A1,2
A2,1
A2,2
C

Figure 10A

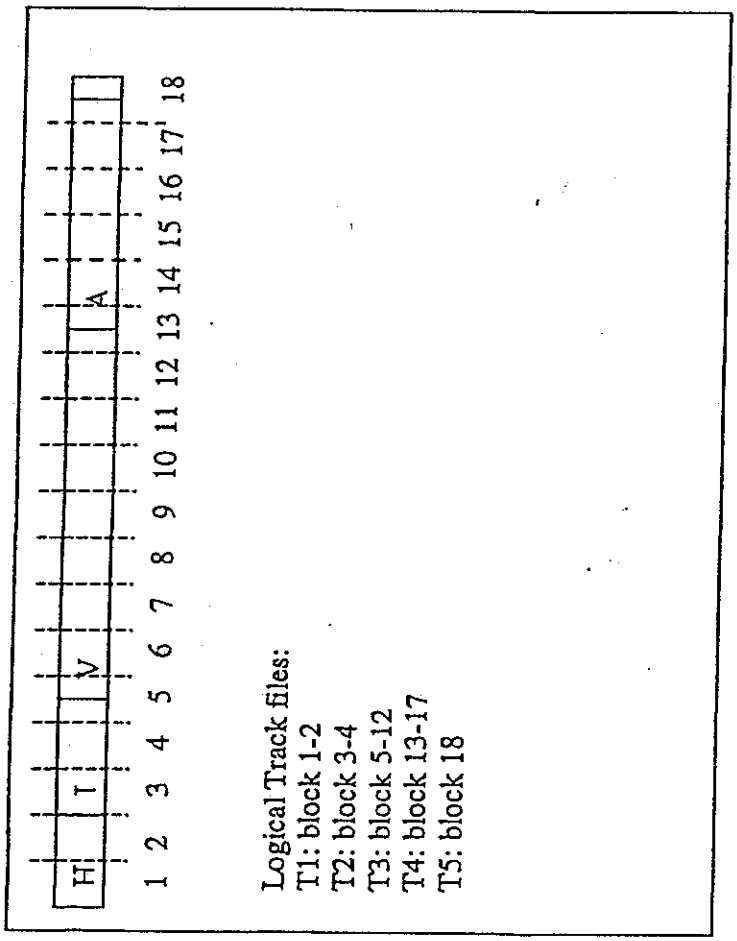

H
T
V
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Logical Track files:
T1: block 1-2
T2: block 3-4
T3: block 5-12
T4: block 13-17
T5: block 18

Figure 10B

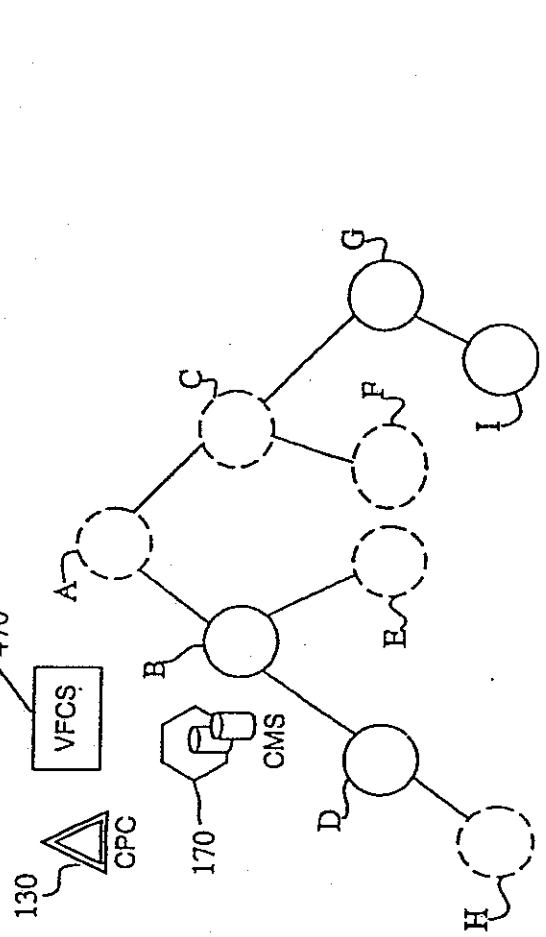


Figure 11A

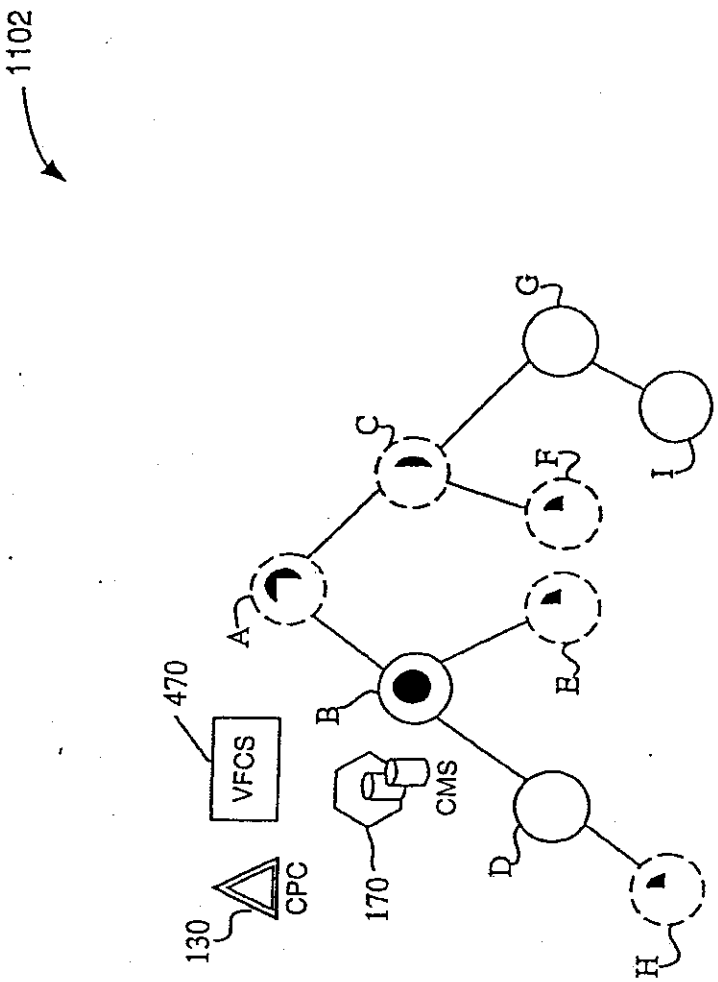


Figure 11B

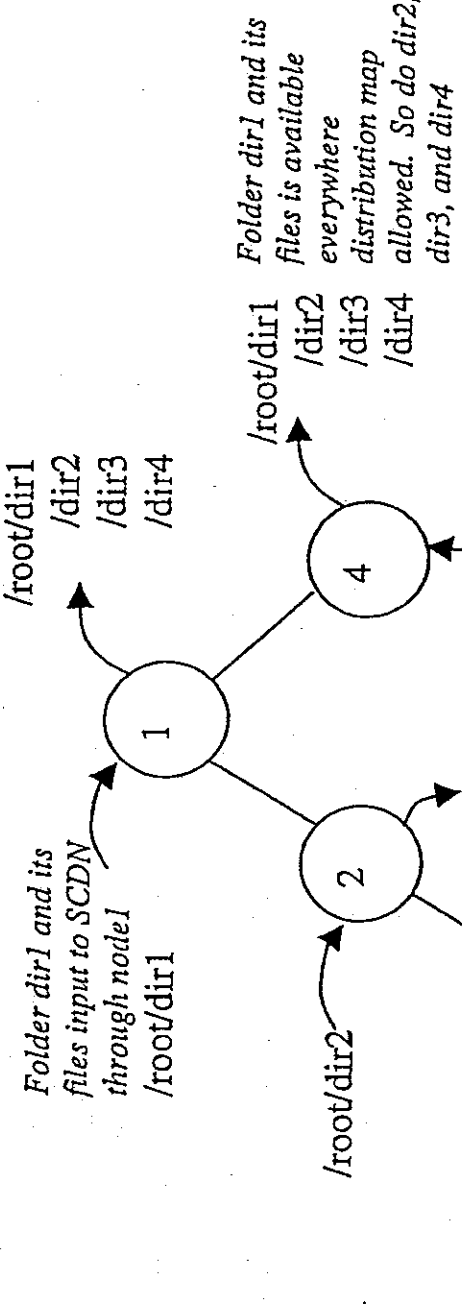

Folder dir1 and its files input to SCDN through node1 /root/dir1
/root/dir2
/root/dir3
/root/dir1 /dir2 /dir3 /dir4
/root/dir1 /dir2 /dir3 /dir4
/root/dir1 /dir2 /dir3 /dir4
/root/dir4
Folder dir1 and its files is available everywhere distribution map allowed. So do dir2, dir3, and dir4
1
2
3
4

Figure 11C

Meta File System Structure 1210

/MetaPath/SCDN ID/Volumes/Directories & Files 1220

Each volume, directory, and file has an object metadata file in the Meta File System Structure

eg:
SCDN ID = 0
Meta Path = "meta"
Volumes = "HR", "Marketing", "Engineering"
Directories= "Policy", "Strategy", "Project", "Design"

Volume metadata = hr.vdat, marketing.vdat engineering.vdat
Directory metadata = policy.ddat, strategy.ddat project.ddat, design.ddat
File metadata = PTO.doc, product.ppt

/meta/0/HR/hr.vdat
Policy/policy.ddat
PTO.doc
Marketing/marketing.vdat
Strategy/strategy.ddat
Product.ppt
Engineering/engineering.vdat
Project/project.ddat
Design/design.ddat

Figure 12A

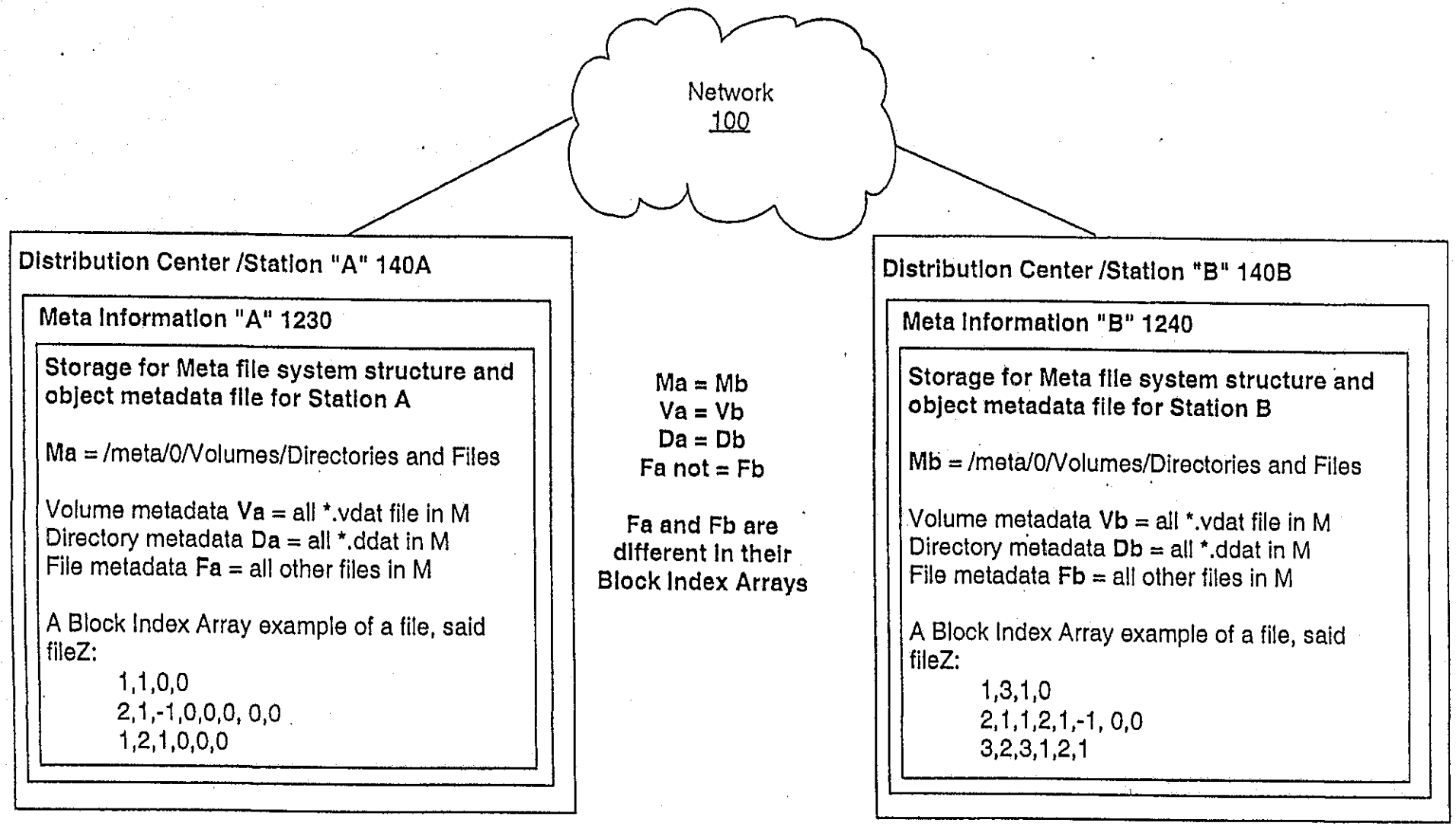

Network
100
Distribution Center /Station "A" 140A
Meta Information "A" 1230
Storage for Meta file system structure and object metadata file for Station A
Ma = /meta/0/Volumes/Directories and Files
Volume metadata Va = all *.vdat file in M
Directory metadata Da = all *.ddat in M
File metadata Fa = all other files in M
A Block Index Array example of a file, said fileZ:
1,1,0,0
2,1,-1,0,0,0, 0,0
1,2,1,0,0,0
Ma = Mb
Va = Vb
Da = Db
Fa not = Fb
Fa and Fb are different in their Block Index Arrays
Distribution Center /Station "B" 140B
Meta Information "B" 1240
Storage for Meta file system structure and object metadata file for Station B
Mb = /meta/0/Volumes/Directories and Files
Volume metadata Vb = all *.vdat file in M
Directory metadata Db = all *.ddat in M
File metadata Fb = all other files in M
A Block Index Array example of a file, said fileZ:
1,3,1,0
2,1,1,2,1,-1, 0,0
3,2,3,1,2,1

Figure 12B

Block file path name 1250

/{blockdataRoot(VolumeIndex)}/{network ID}/blocks/{node_limiting_path(media ID)}/

Eg:
Assuming that there are two volumes:
BlockDataRoot(1)=BlockRoot1, and BlockDataRoot(2)=BlockRoot2.

SCDN network ID = 0

FileZ with unique file ID = 00000000001234567890,
FileZ has 2 segments, segment 1 has 5 blocks, segment 2 has 8 blocks

When all blocks are in local storage:

/BlockRoot1/0/blocks/000/000/000/012/345/678/90/1/0/1234567890.1.0001.block
1234567890.1.0003.block
1234567890.1.0005.block
1234567890.2.0002.block
1234567890.2.0004.block
1234567890.2.0006.block
1234567890.2.0008.block

/BlockRoot2/0/blocks/000/000/000/012/345/678/90/1/0/1234567890.1.0002.block
1234567890.1.0004.block
1234567890.2.0001.block
1234567890.2.0003.block
1234567890.2.0005.block
1234567890.2.0007.block

Figure 12C

25 APR 2005

Volume Metadata file - {volume name}.vdat 1300

type	description
Object ID	Network ID
Object ID	Origin Node ID
Object ID	Volume ID
Object ID	Content Owner ID
String Buffer [255]	Content Owner Name
String Buffer [255]	Origin CM Address
String Buffer [255]	Origin CU Address
String Buffer [255]	Origin DS Address
Int 64	Reserved Space
String Buffer [255]	Authentication Server Address
Int 32	Authentication Server Port
Instant	Space Reservation Update Time
Instant	Record Update Time
Instant	Creation Time
Instant	Last Mod Time
Int 32	Number of directories
Int 32	Number of files
Int 64	Distribution map
Array of access control objects	Access control list
...	...

Figure 13A

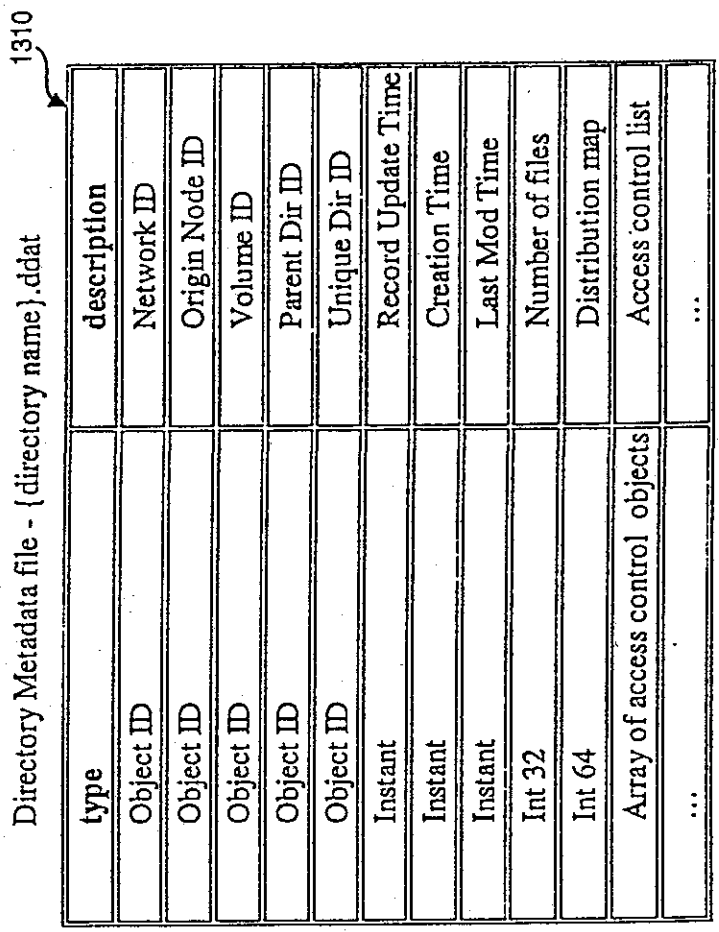

Directory Metadata file - {directory name}.ddat
1310
type
description
Object ID
Network ID
Object ID
Origin Node ID
Object ID
Volume ID
Object ID
Parent Dir ID
Object ID
Unique Dir ID
Instant
Record Update Time
Instant
Creation Time
Instant
Last Mod Time
Int 32
Number of files
Int 64
Distribution map
Array of access control objects
Access control list
...
...

Figure 13B

File Metadata – {file name}

type	description
Object ID	Network ID
Object ID	Dir ID
Object ID	File ID
String Buffer [255]	File Name
String Buffer [255]	Object Code
Int 32	Duration in seconds for time based content
Int 64	File Size
Int 32	Block Size
Int 32	Popularity
Int 32	Number of segments in file
Array (# segments)	Number of blocks for each segment
Instant	Time Stamp
Instant	Expiration Date
Instant	Record Update Time
Instant	Last Use Time
Instant	Creation Date
Int 64	Distribution map
Array	Access control list
Array	Access log
Int 32	Version
Int 32	File type
Array	Block Index of segment #1
Array	Block Index of segment #2
...	...
(end of data)	

1320

Figure 13C

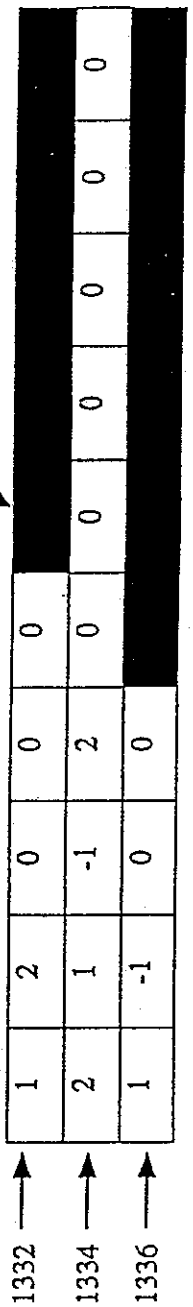

Example of Block Index Array
1330
1332
1334
1336
1 2 0 0 0 0 0 0
2 1 -1 2 0 0 0 0
1 -1 0 0 0 0 0 0
1 = Storage device #1
2 = Storage device #2
-1 = In the process of downloading from other server
0 = not in local storage

Figure 13D

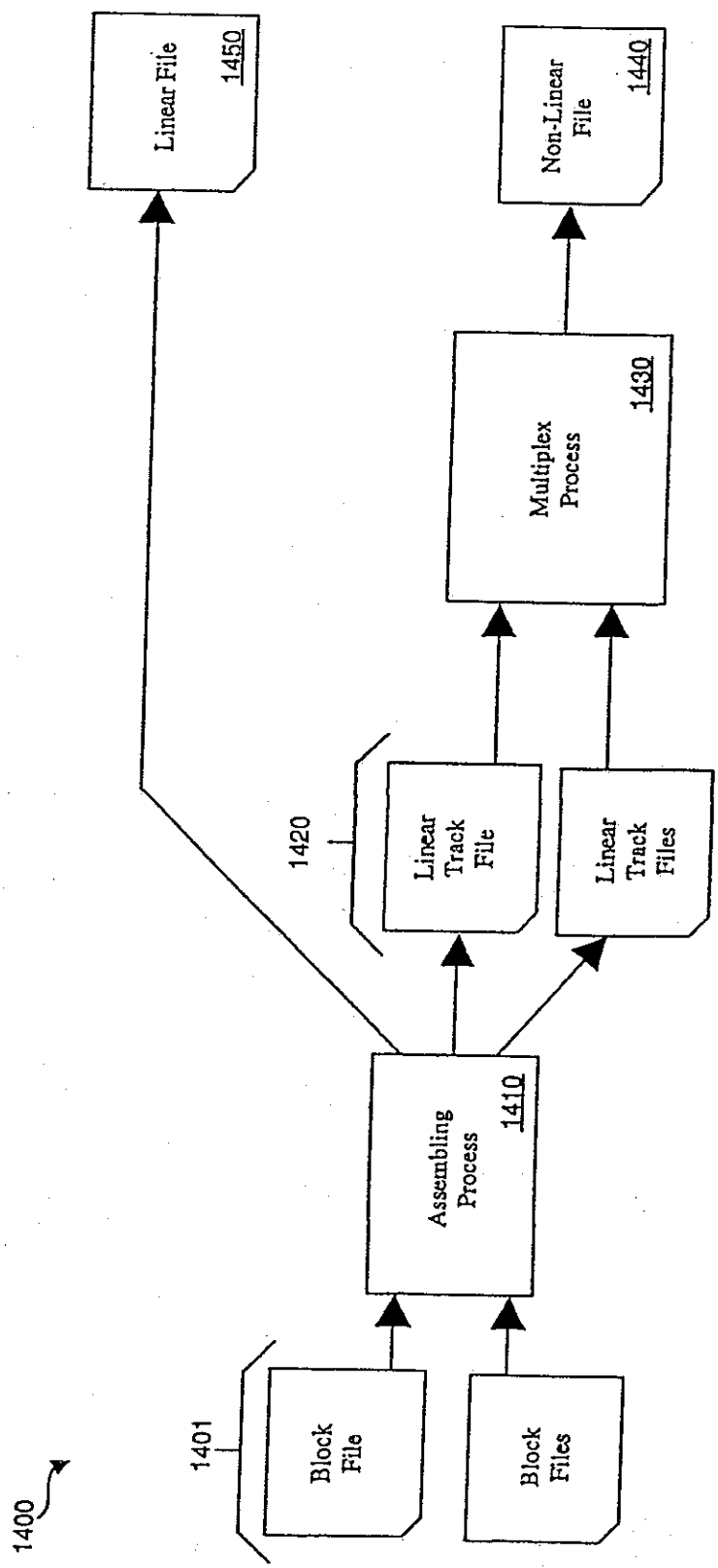

1400
1401
Block File
Block Files
Assembling Process
1410
1420
Linear Track File
Linear Track Files
Multiplex Process
1430
Non-Linear File
1440
Linear File
1450

Figure 14A

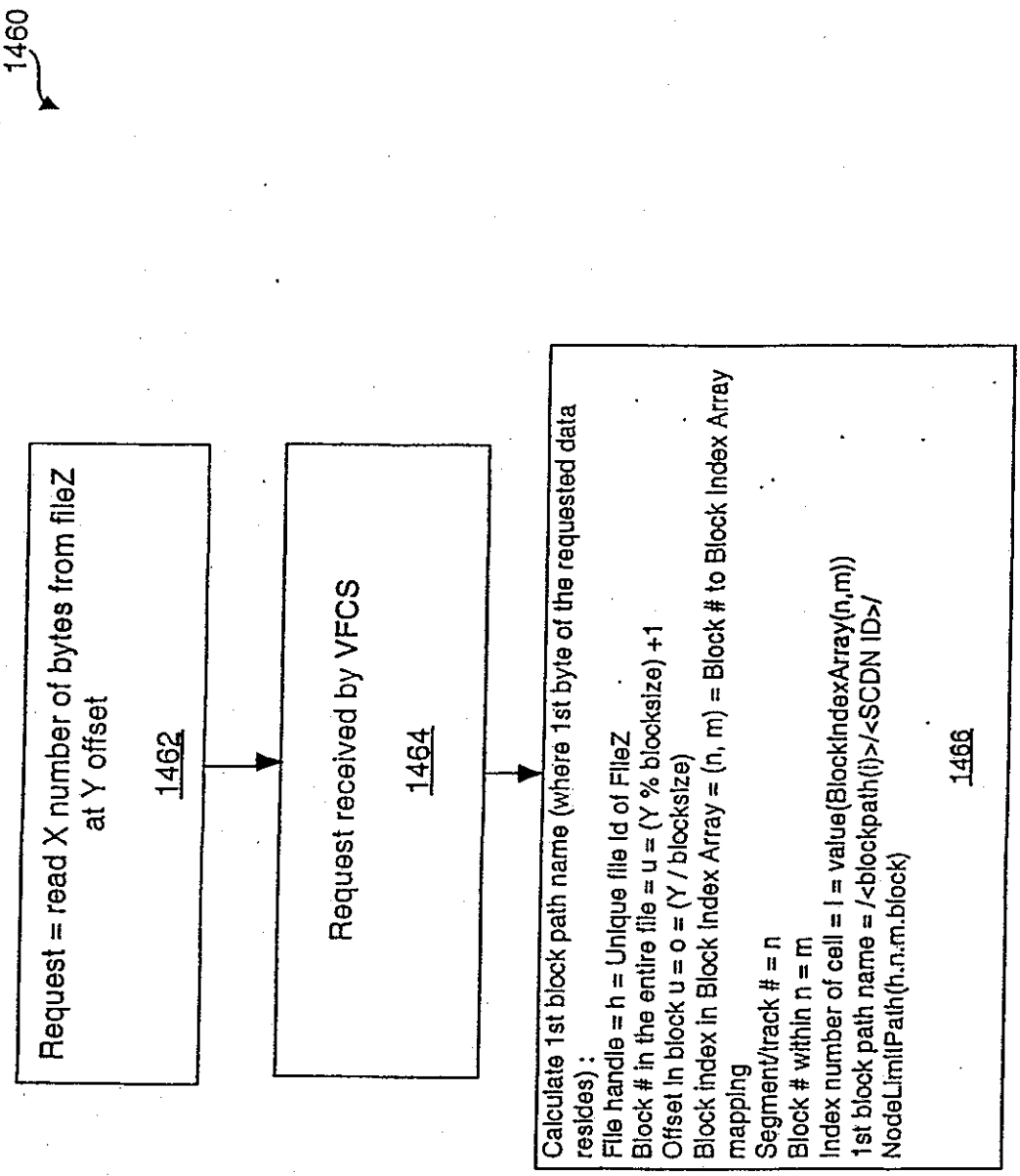

1460
Request = read X number of bytes from fileZ at Y offset
1462
Request received by VFCS
1464
Calculate 1st block path name (where 1st byte of the requested data resides) :
File handle = h = Unique file Id of FileZ
Block # in the entire file = u = (Y % blocksize) +1
Offset in block u = o = (Y / blocksize)
Block index in Block Index Array = (n, m) = Block # to Block Index Array mapping
Segment/track # = n
Block # within n = m
Index number of cell = l = value(BlockIndexArray(n,m))
1st block path name = /<blockpath(l)>/<SCDN ID>/
NodeLimilPath(h.n.m.block)
1466

Figure 14B

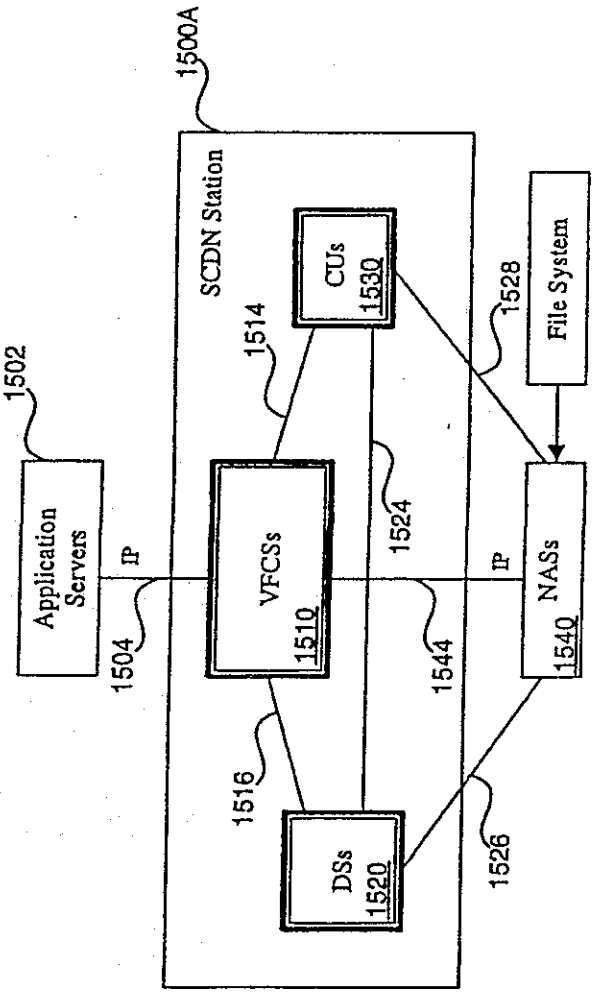

1500A
SCDN Station
CUs
1530
1514
1528
File System
1502
Application
Servers
IP
VFCSs
1510
1524
IP
NASs
1540
1504
1544
1516
DSs
1520
1526

Figure 15A

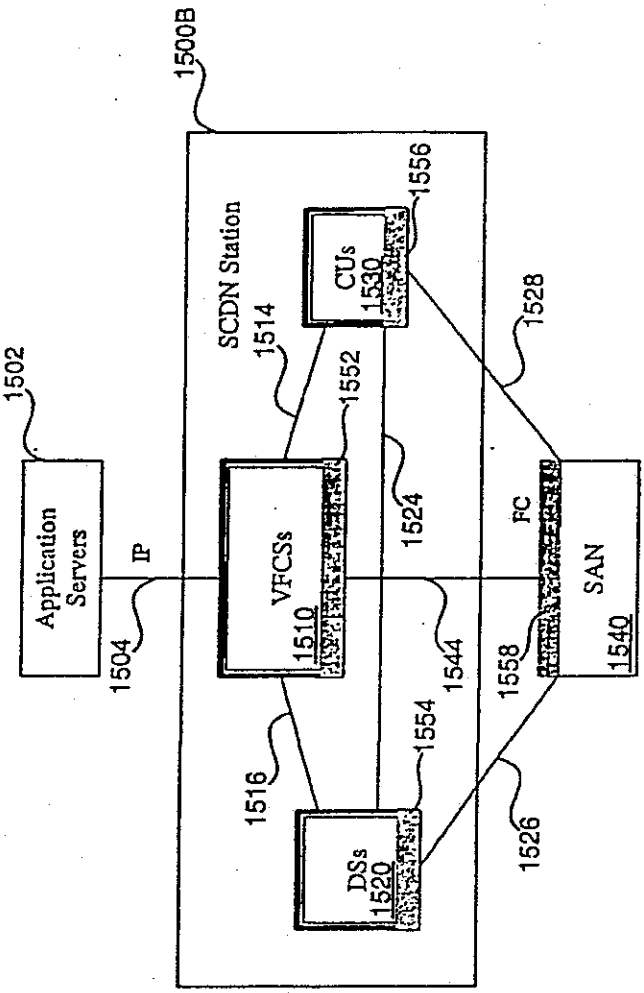

1500B
SCDN Station
1514
1556
CUs
1530
1528
1502
1552
Application Servers
IP
VFCSs
1510
1524
FC
SAN
1540
1504
1544
1558
1554
1516
DSs
1520
1526

Figure 15B

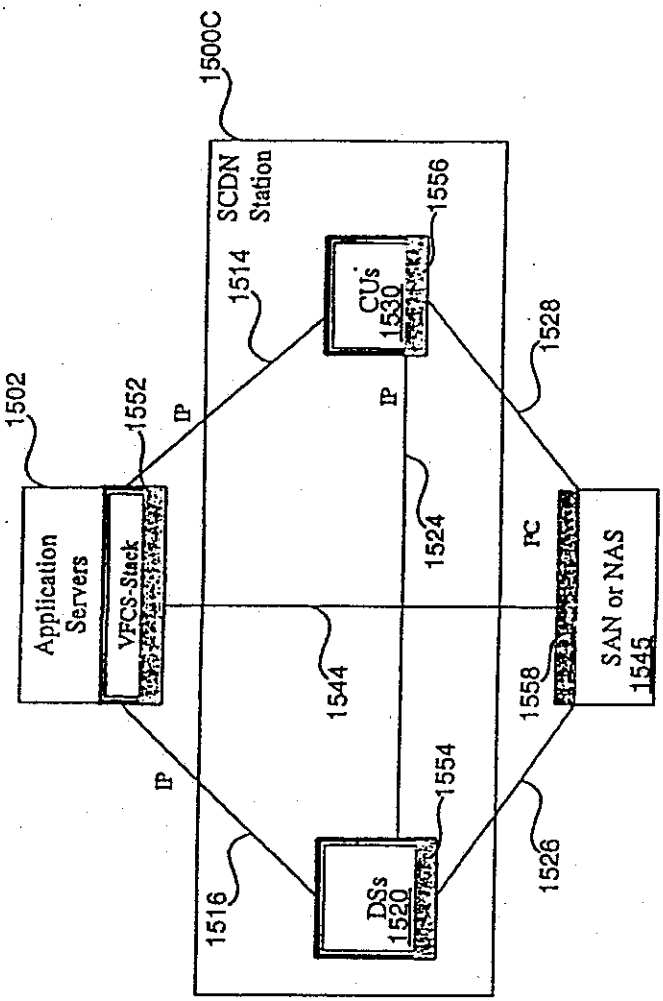

1500C
SCDN Station
1556
1514
CUs 1530
1528
1502
1552
IP
IP
Application Servers
VFCS-Stack
1524
FC
SAN or NAS 1545
1544
1558
1554
IP
1526
1516
DSs 1520

Figure 15C

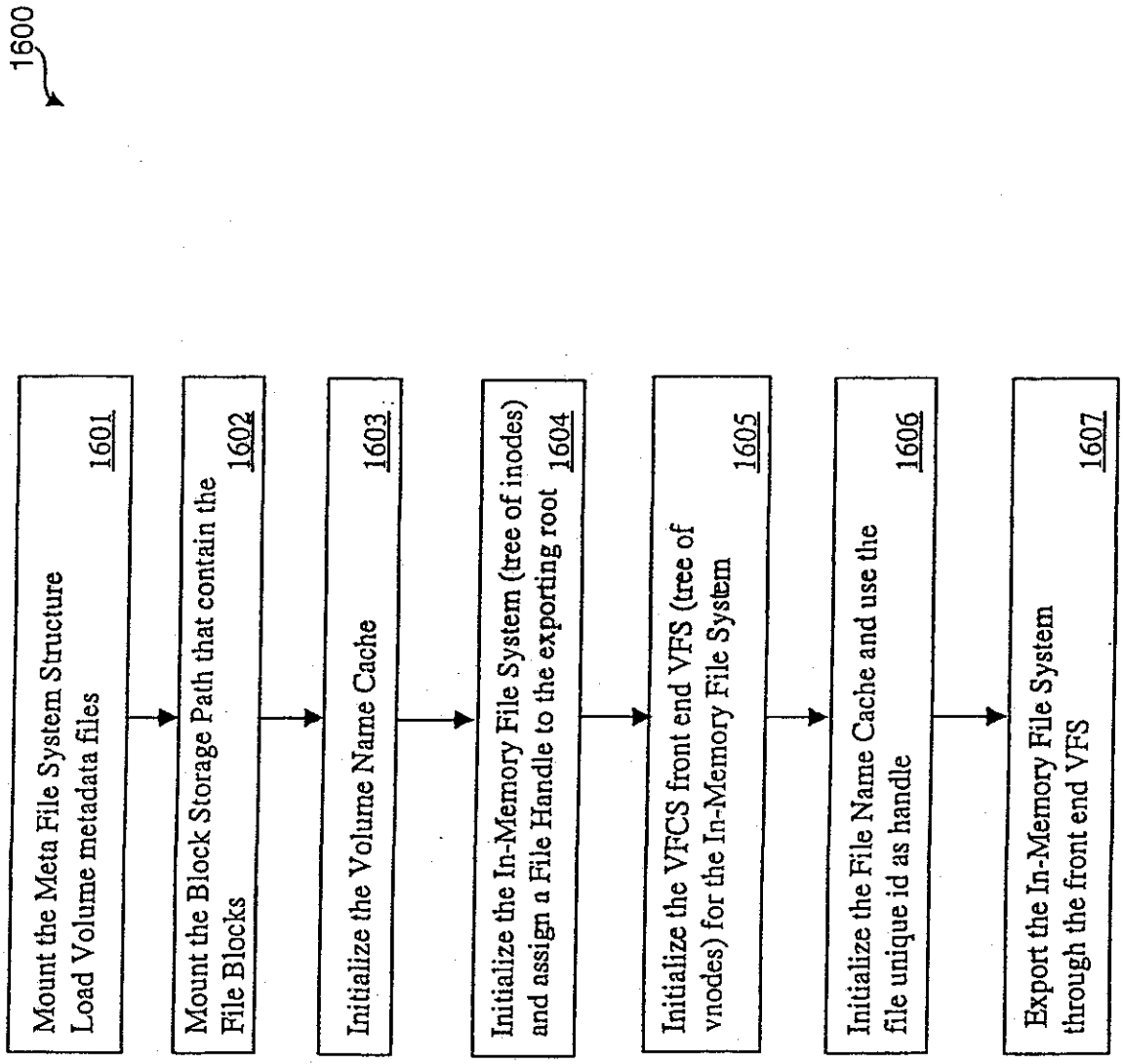

1600
Mount the Meta File System Structure Load Volume metadata files 1601
Mount the Block Storage Path that contain the File Blocks 1602
Initialize the Volume Name Cache 1603
Initialize the In-Memory File System (tree of inodes) and assign a File Handle to the exporting root 1604
Initialize the VFCS front end VFS (tree of vnodes) for the In-Memory File System 1605
Initialize the File Name Cache and use the file unique id as handle 1606
Export the In-Memory File System through the front end VFS 1607

Figure 16

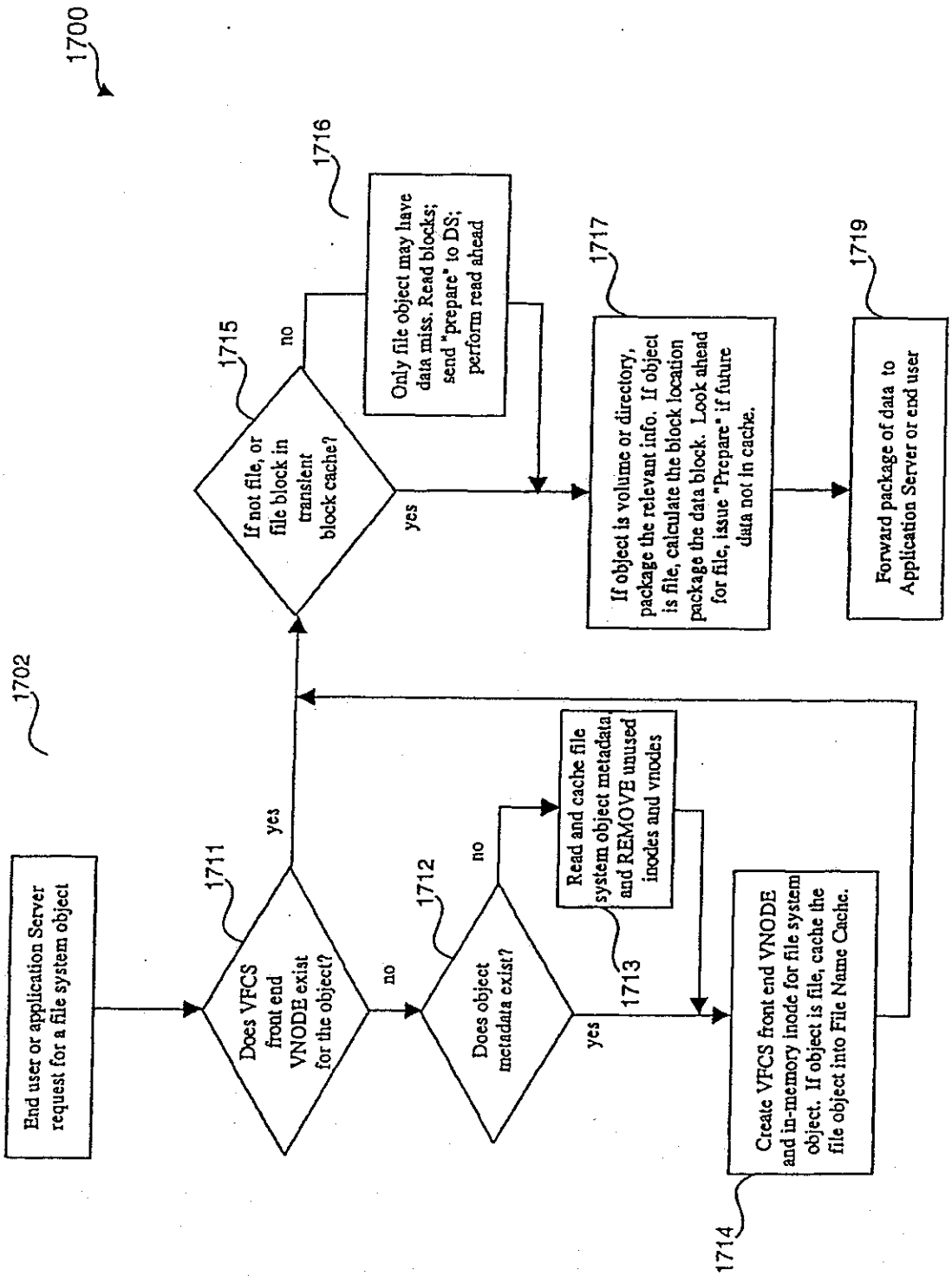

1700
End user or application Server request for a file system object
1702
Does VFCS front end VNODE exist for the object?
1711
yes
no
Does object metadata exist?
1712
no
yes
Read and cache file system object metadata and REMOVE unused inodes and vnodes
1713
Create VFCS front end VNODE and in-memory inode for file system object. If object is file, cache the file object into File Name Cache.
1714
If not file, or file block in transient block cache?
1715
no
yes
Only file object may have data miss. Read blocks; send "prepare" to DS; perform read ahead
1716
If object is volume or directory, package the relevant info. If object is file, calculate the block location package the data block. Look ahead for file, issue "Prepare" if future data not in cache.
1717
Forward package of data to Application Server or end user
1719

Figure 17A

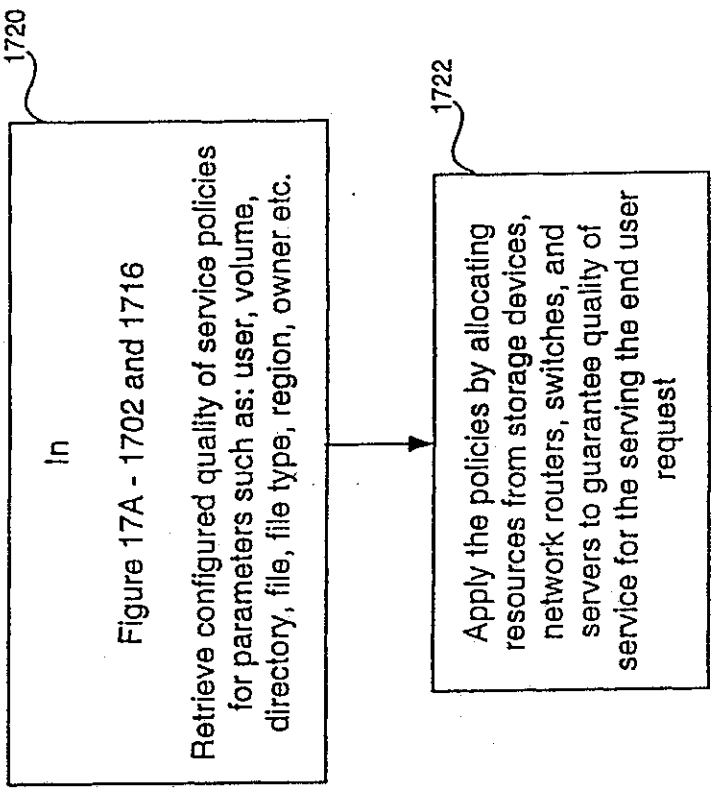

1720
In
Figure 17A - 1702 and 1716
Retrieve configured quality of service policies for parameters such as: user, volume, directory, file, file type, region, owner etc.
1722
Apply the policies by allocating resources from storage devices, network routers, switches, and servers to guarantee quality of service for the serving the end user request

Figure 17B

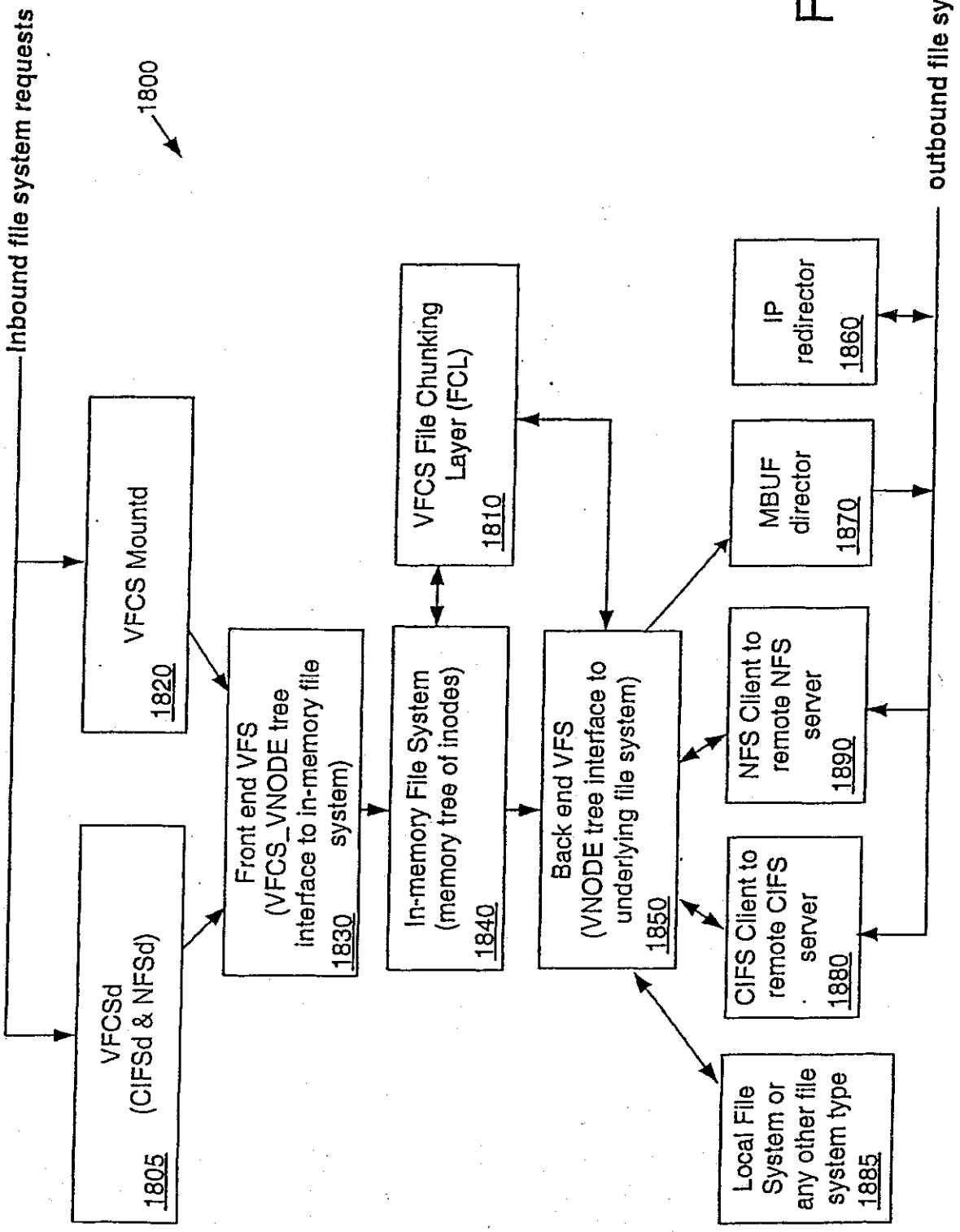

1800
Inbound file system requests
outbound file system response
VFCSd
(CIFSd & NFSd)
1805
VFCS Mountd
1820
Front end VFS
(VFCS_VNODE tree
interface to in-memory file
system)
1830
In-memory File System
(memory tree of inodes)
1840
VFCS File Chunking
Layer (FCL)
1810
Back end VFS
(VNODE tree interface to
underlying file system)
1850
Local File
System or
any other file
system type
1885
CIFS Client to
remote CIFS
server
1880
NFS Client to
remote NFS
server
1890
MBUF
director
1870
IP
redirector
1860

Figure 18

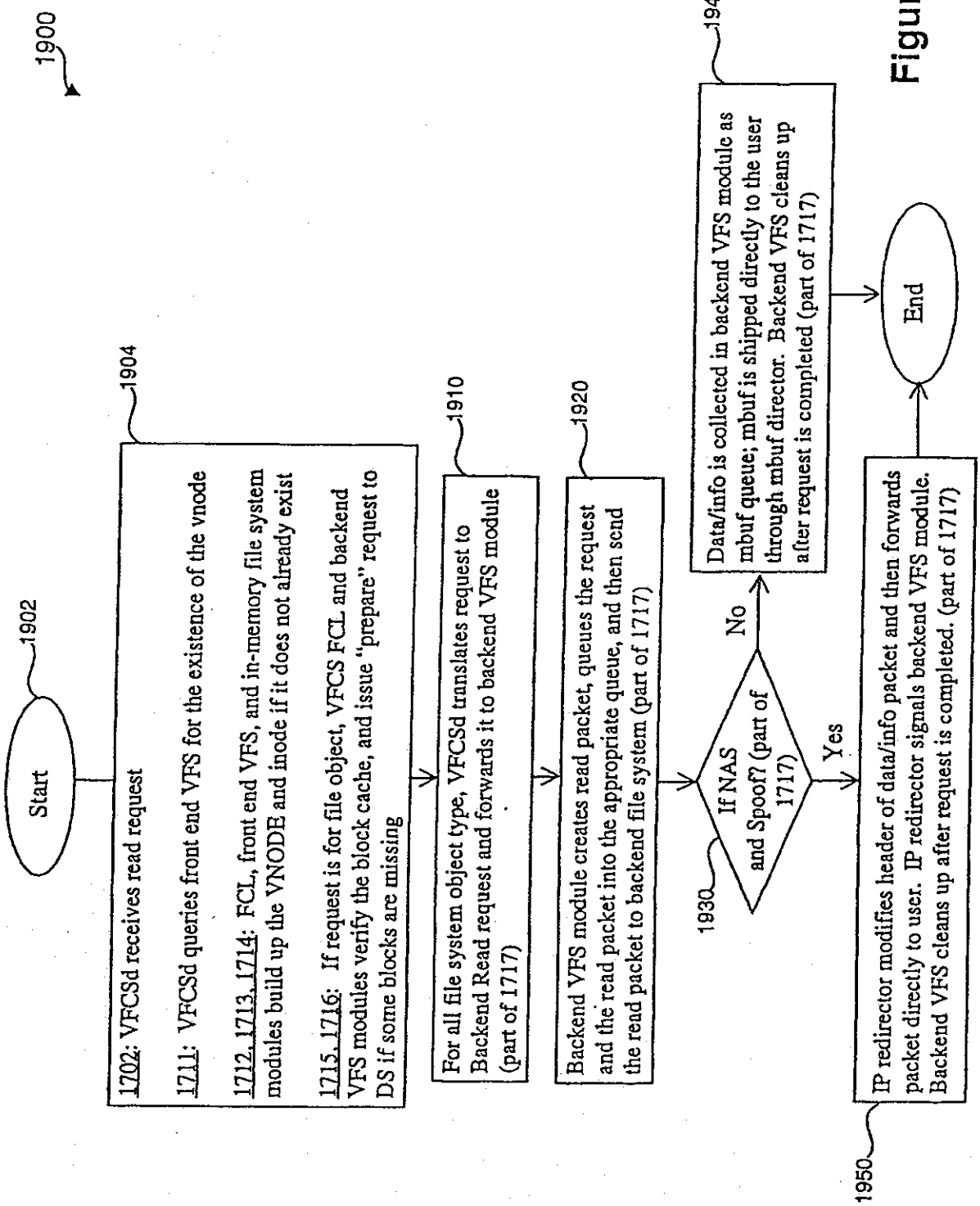

1900
Start
1902
1702: VFCSd receives read request
1711: VFCSd queries front end VFS for the existence of the vnode
1712, 1713, 1714: FCL, front end VFS, and in-memory file system modules build up the VNODE and inode if it does not already exist
1715, 1716: If request is for file object, VFCS FCL and backend VFS modules verify the block cache, and issue "prepare" request to DS if some blocks are missing
1904
For all file system object type, VFCSd translates request to Backend Read request and forwards it to backend VFS module (part of 1717)
1910
Backend VFS module creates read packet, queues the request and the read packet into the appropriate queue, and then send the read packet to backend file system (part of 1717)
1920
If NAS and Spoof? (part of 1717)
1930
No
Yes
Data/info is collected in backend VFS module as mbuf queue; mbuf is shipped directly to the user through mbuf director. Backend VFS cleans up after request is completed (part of 1717)
1940
IP redirector modifies header of data/info packet and then forwards packet directly to user. IP redirector signals backend VFS module. Backend VFS cleans up after request is completed. (part of 1717)
1950
End

Figure 19

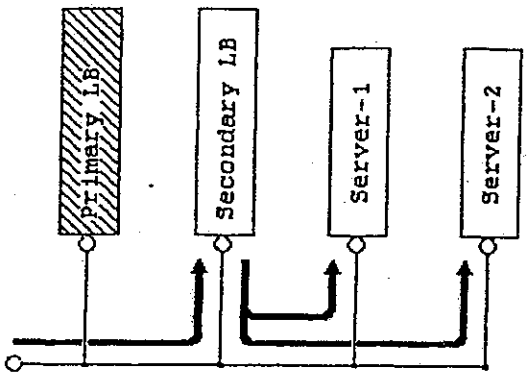


Figure 22

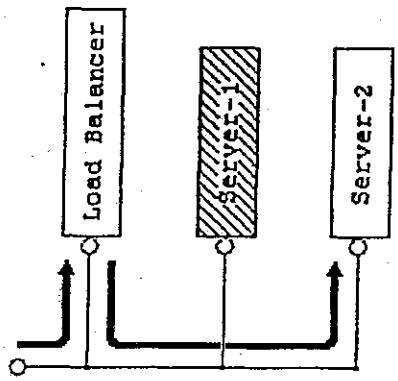


Figure 21

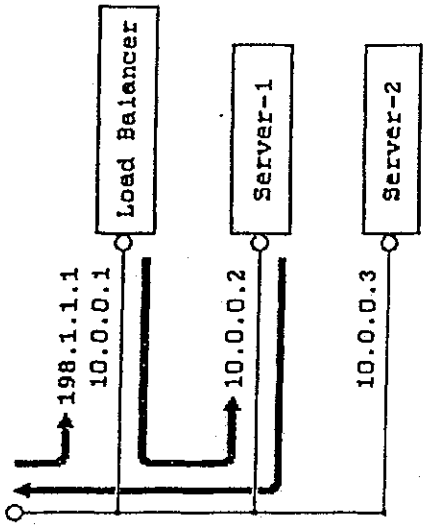


Figure 24

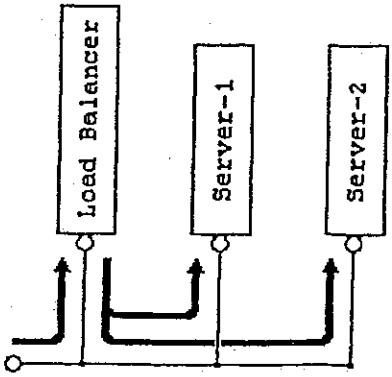


Figure 20

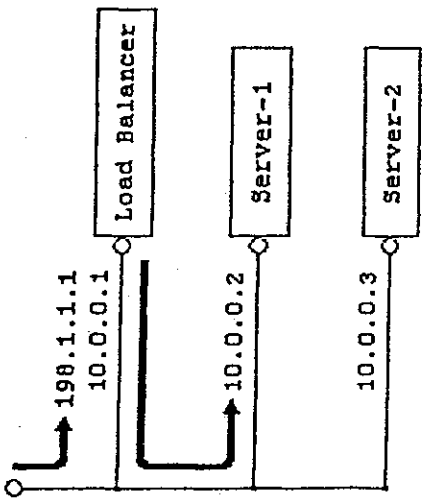


Figure 23

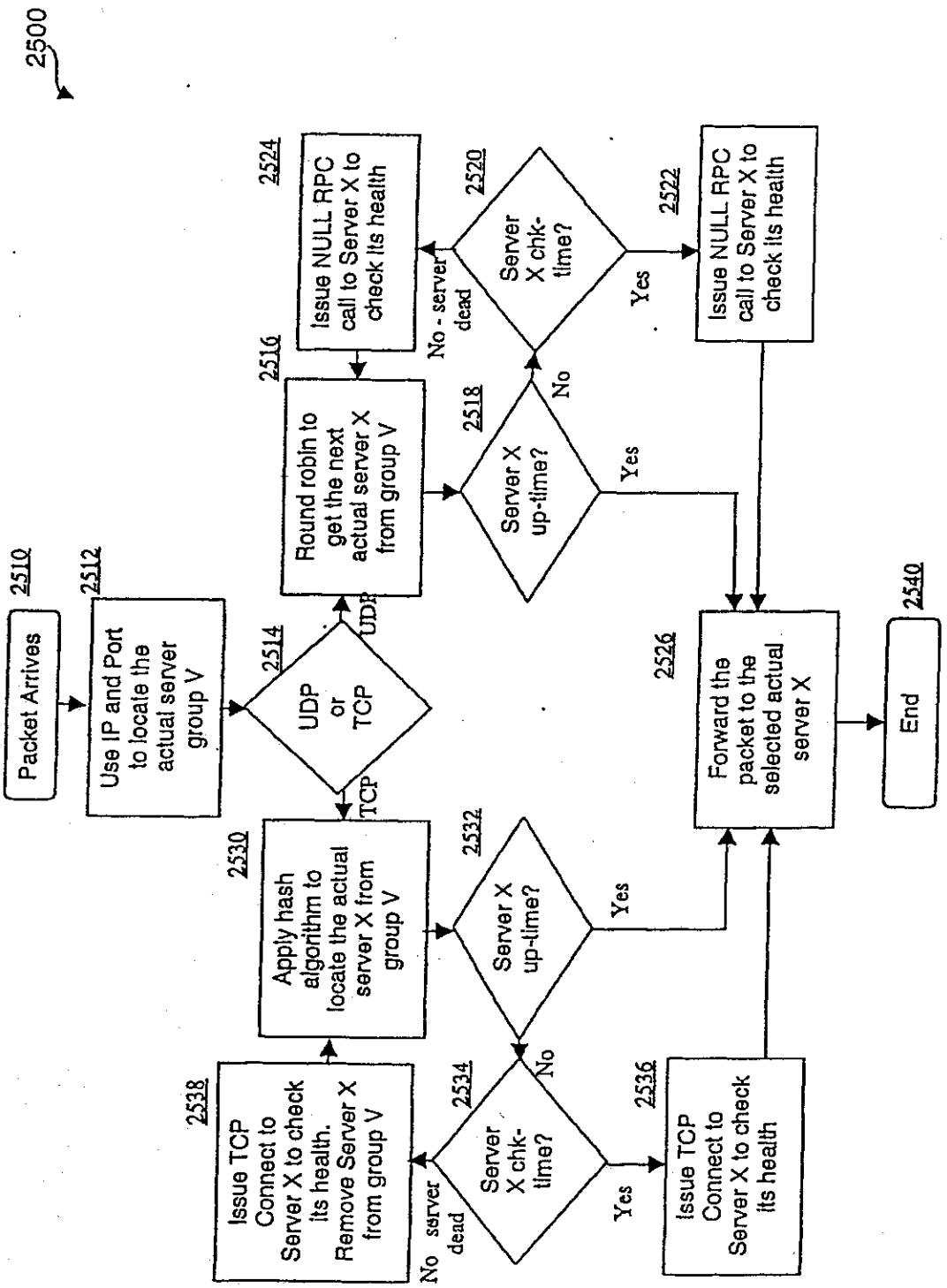

2500
Packet Arrives
2510
Use IP and Port to locate the actual server group V
2512
UDP or TCP
2514
UDP
TCP
Round robin to get the next actual server X from group V
2516
Server X up-time?
2518
No
Yes
Issue NULL RPC call to Server X to check its health
2524
Server X chk-time?
2520
No - server dead
Yes
Issue NULL RPC call to Server X to check its health
2522
Apply hash algorithm to locate the actual server X from group V
2530
Server X up-time?
2532
Yes
No
Forward the packet to the selected actual server X
2526
End
2540
Issue TCP Connect to Server X to check its health. Remove Server X from group V
2538
Server X chk-time?
2534
No server dead
Yes
Issue TCP Connect to Server X to check its health
2536

Figure 25

25 APR 2005

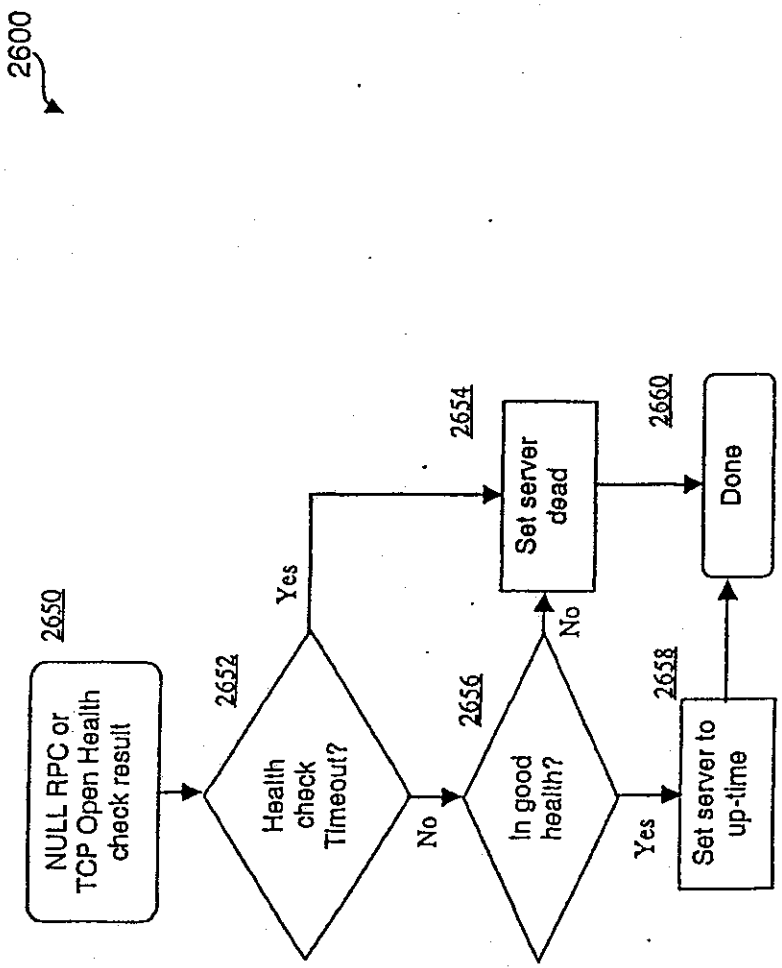


Figure 26

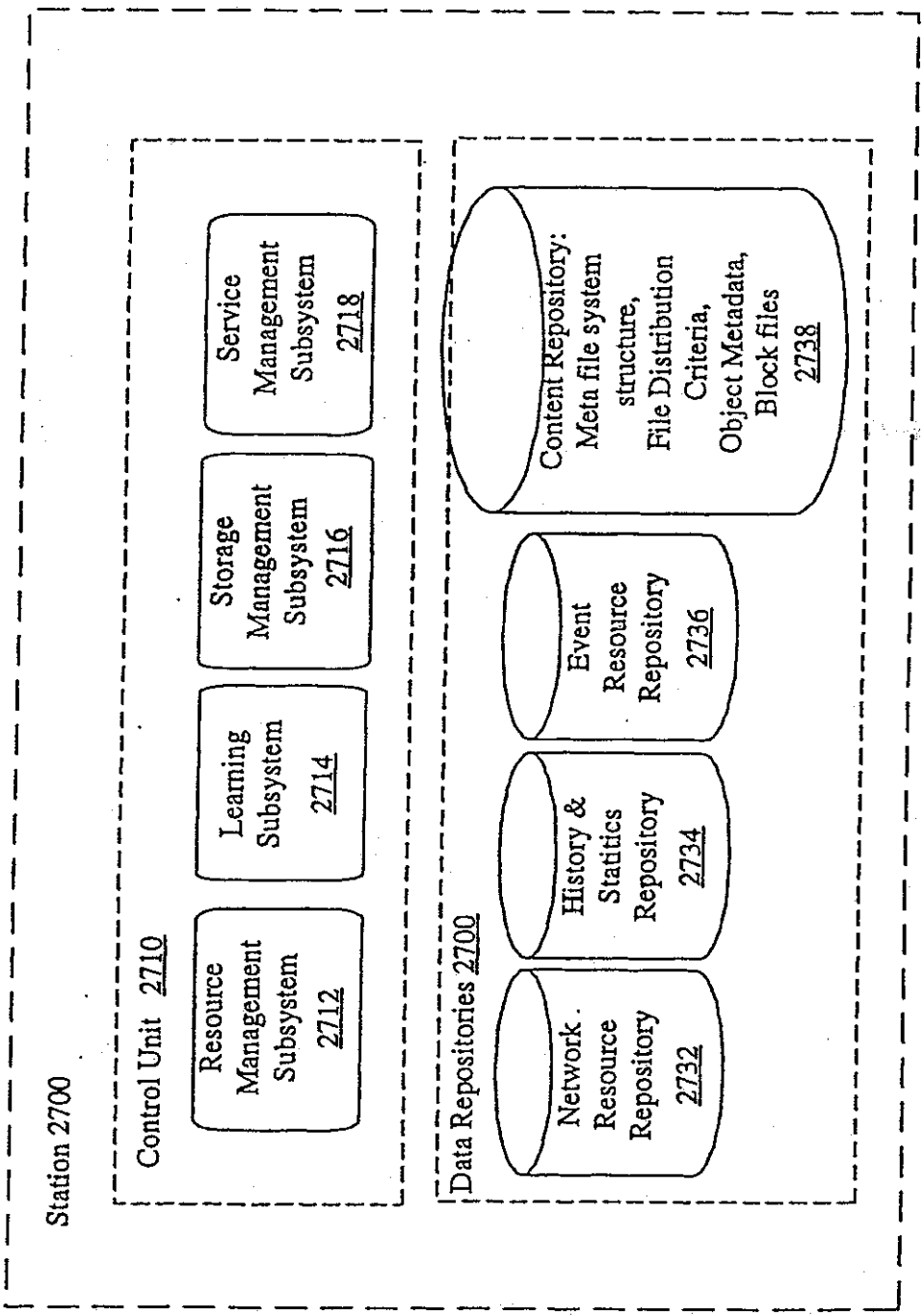

Station 2700
Control Unit 2710
Resource Management Subsystem 2712
Learning Subsystem 2714
Storage Management Subsystem 2716
Service Management Subsystem 2718
Data Repositories 2700
Network Resource Repository 2732
History & Statitics Repository 2734
Event Resource Repository 2736
Content Repository: Meta file system structure, File Distribution Criteria, Object Metadata, Block files 2738

Figure 27

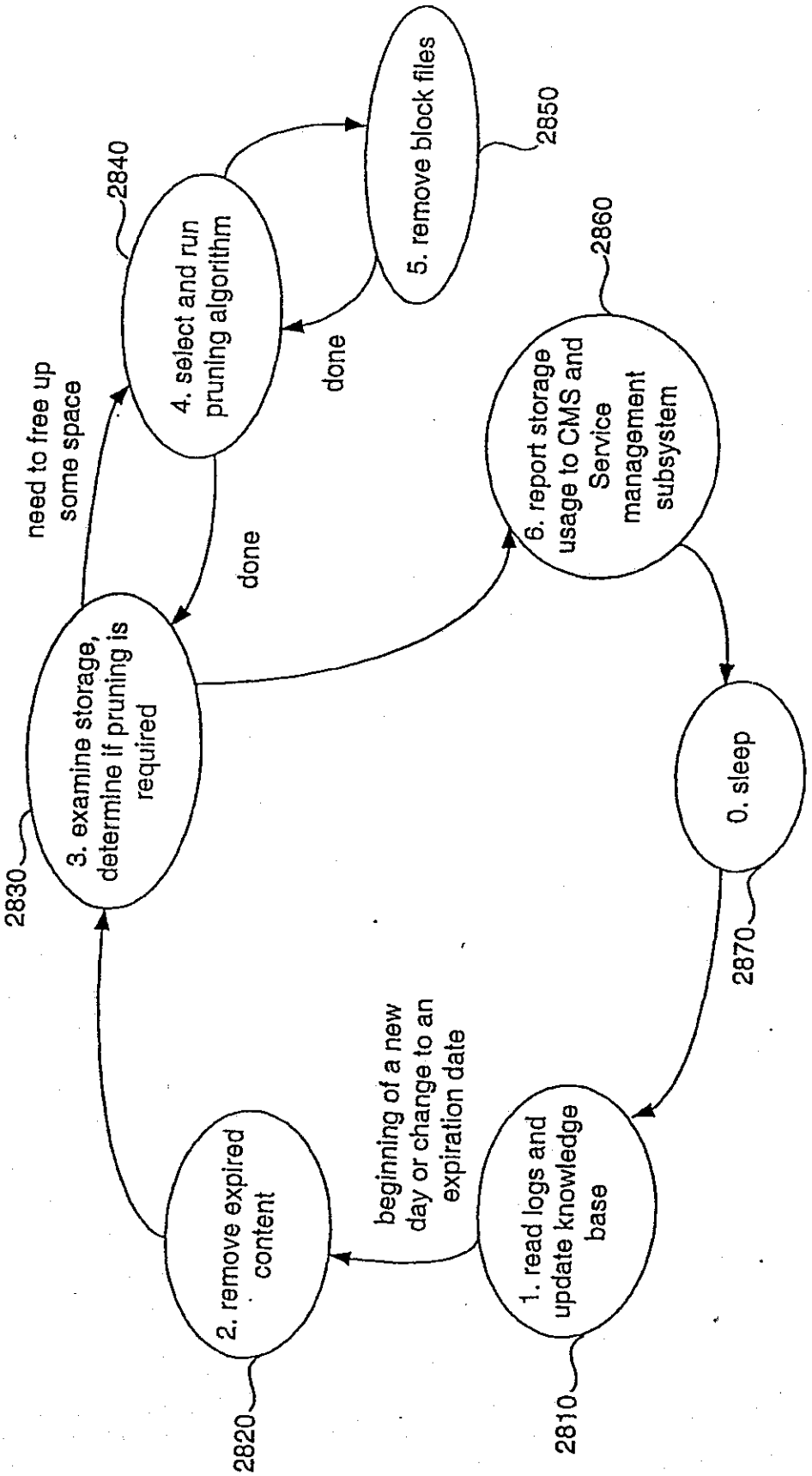

1. read logs and update knowledge base
2810
2. remove expired content
2820
3. examine storage, determine if pruning is required
2830
4. select and run pruning algorithm
2840
5. remove block files
2850
6. report storage usage to CMS and Service management subsystem
2860
0. sleep
2870
beginning of a new day or change to an expiration date
need to free up some space
done
done

Figure 28

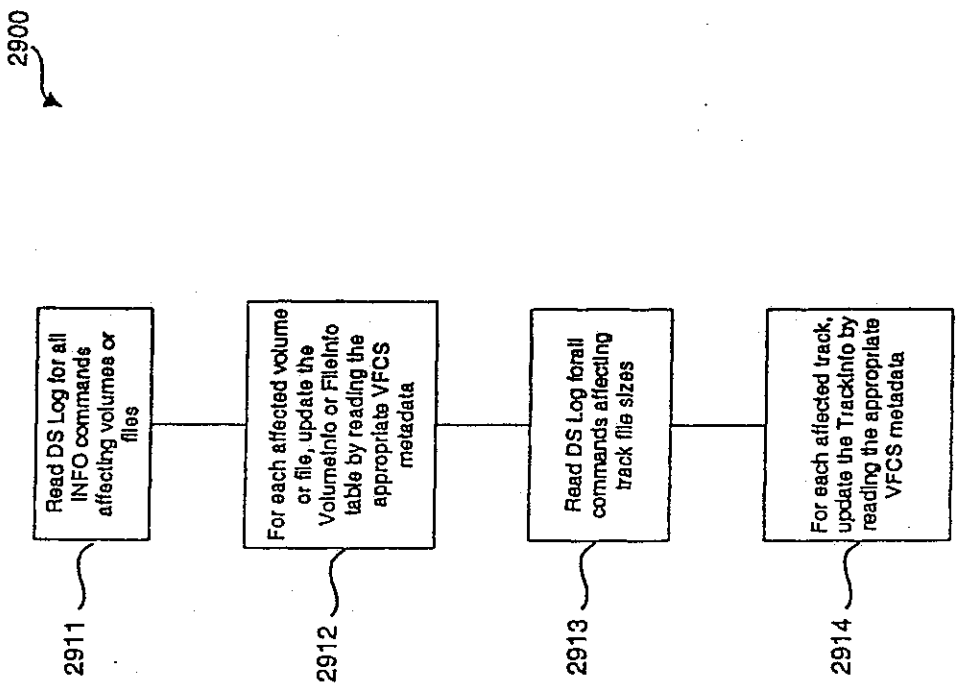

2900
Read DS Log for all INFO commands affecting volumes or files
2911
For each affected volume or file, update the VolumeInfo or FileInfo table by reading the appropriate VFCS metadata
2912
Read DS Log forall commands affecting track file sizes
2913
For each affected track, update the TrackInfo by reading the appropriate VFCS metadata
2914

Figure 29A

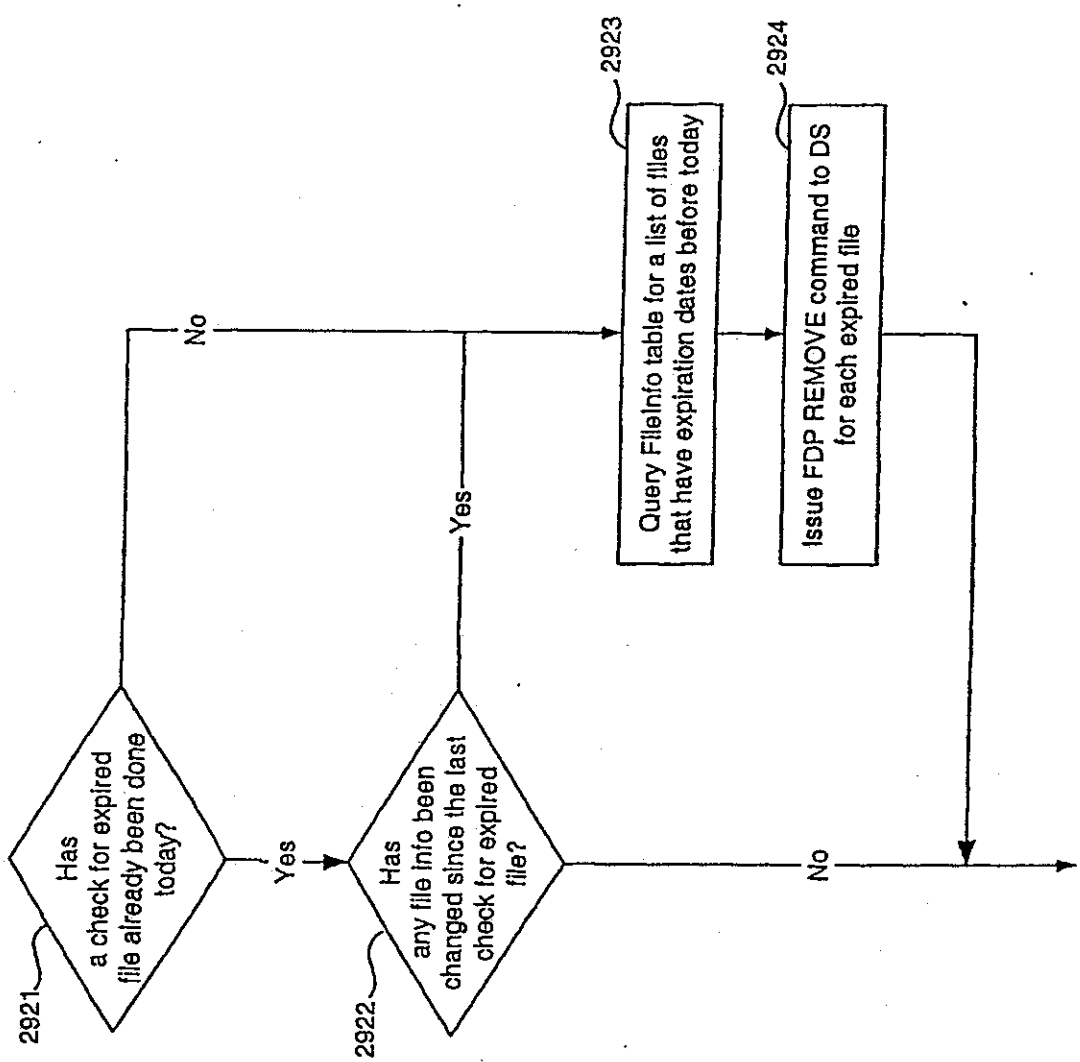

Has a check for expired file already been done today?
2921
No
Yes
Has any file info been changed since the last check for expired file?
2922
Yes
No
2923
Query FileInfo table for a list of files that have expiration dates before today
2924
Issue FDP REMOVE command to DS for each expired file

Figure 29B

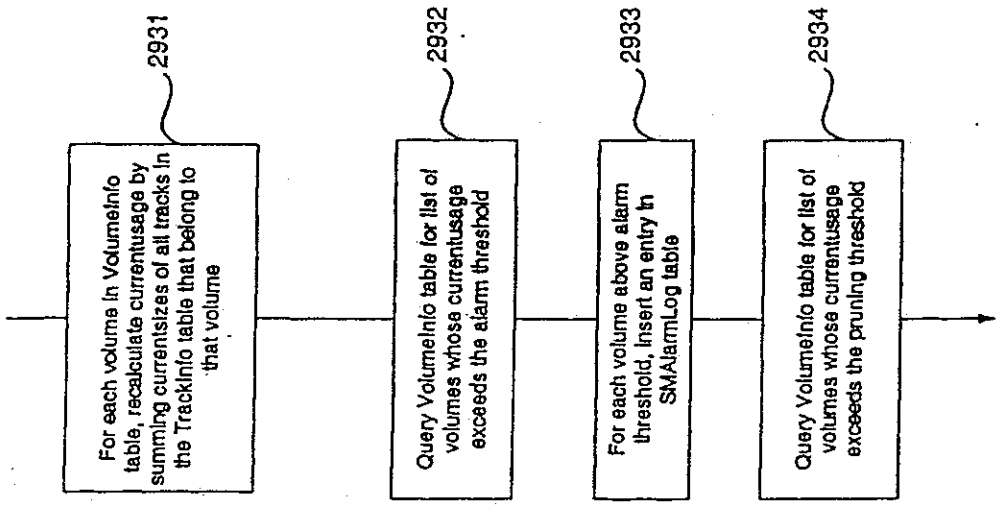

2931
For each volume in VolumeInfo table, recalculate currentusage by summing currentsizes of all tracks in the TrackInfo table that belong to that volume
2932
Query VolumeInfo table for list of volumes whose currentusage exceeds the alarm threshold
2933
For each volume above alarm threshold, insert an entry in SMAlarmLog table
2934
Query VolumeInfo table for list of volumes whose currentusage exceeds the pruning threshold

Figure 29C

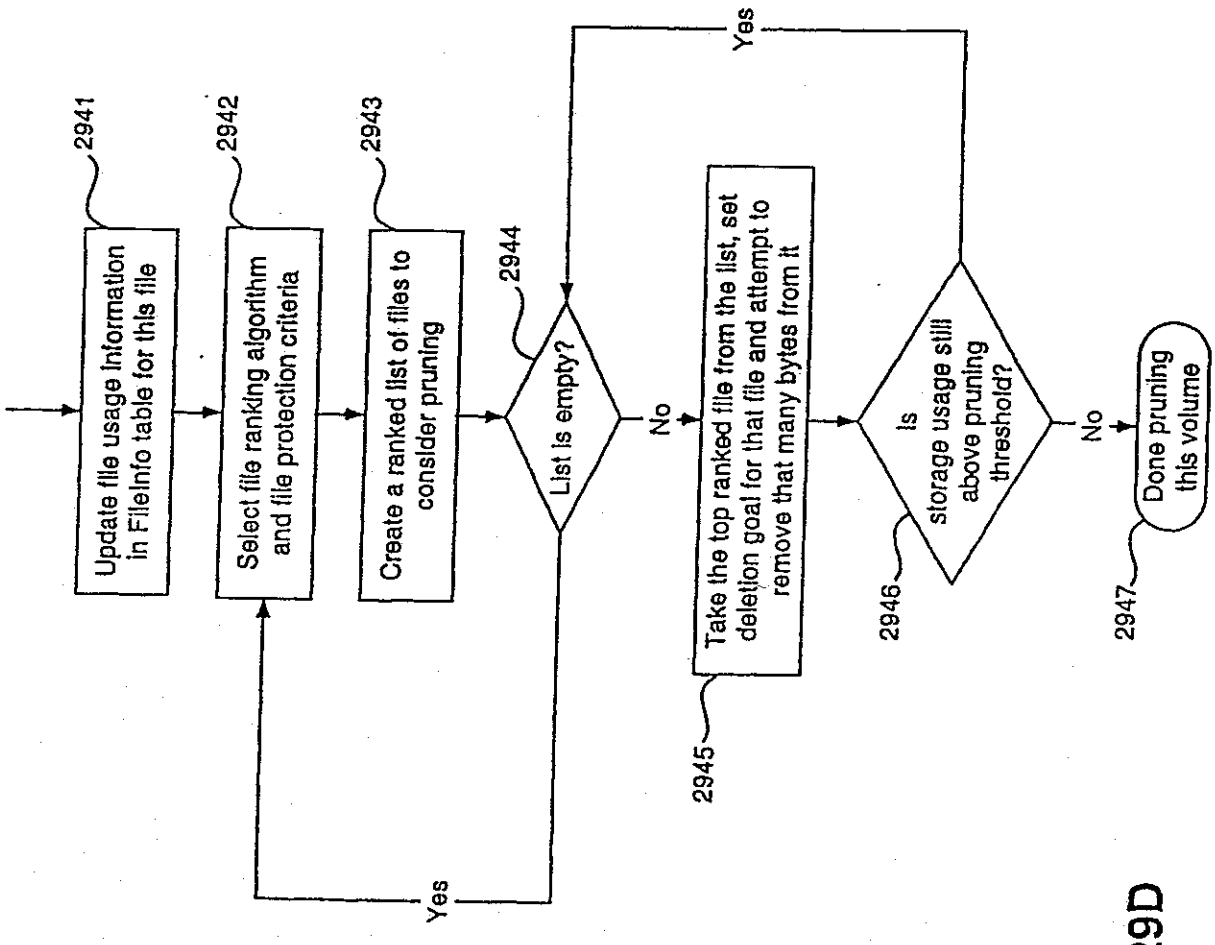

2941
Update file usage information in FileInfo table for this file
2942
Select file ranking algorithm and file protection criteria
2943
Create a ranked list of files to consider pruning
2944
List is empty?
Yes
No
2945
Take the top ranked file from the list, set deletion goal for that file and attempt to remove that many bytes from it
2946
Is storage usage still above pruning threshold?
Yes
No
2947
Done pruning this volume

Figure 29D

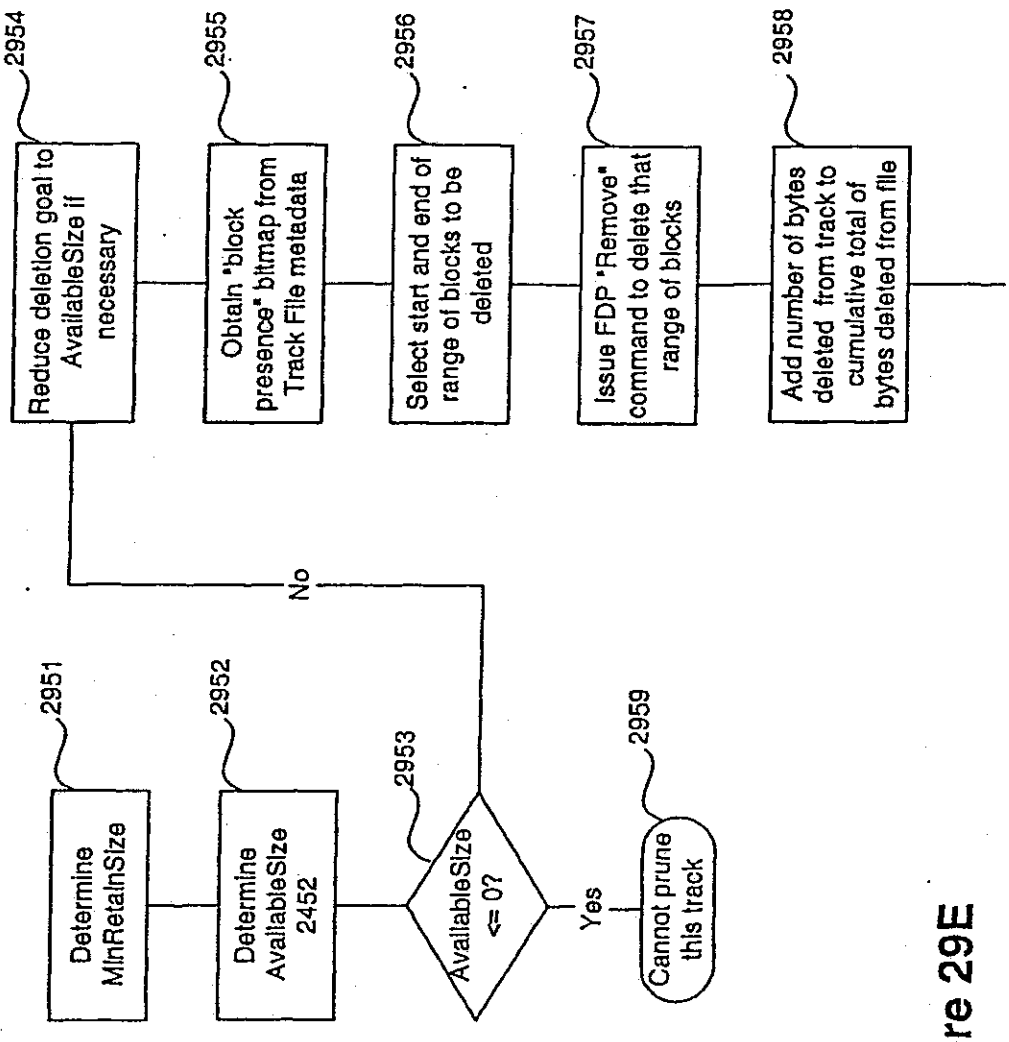

2954
Reduce deletion goal to AvailableSize if necessary
2955
Obtain "block presence" bitmap from Track File metadata
2956
Select start and end of range of blocks to be deleted
2957
Issue FDP "Remove" command to delete that range of blocks
2958
Add number of bytes deleted from track to cumulative total of bytes deleted from file
No
2951
Determine MinRetainSize
2952
Determine AvailableSize 2452
2953
AvailableSize <= 0?
Yes
2959
Cannot prune this track

Figure 29E

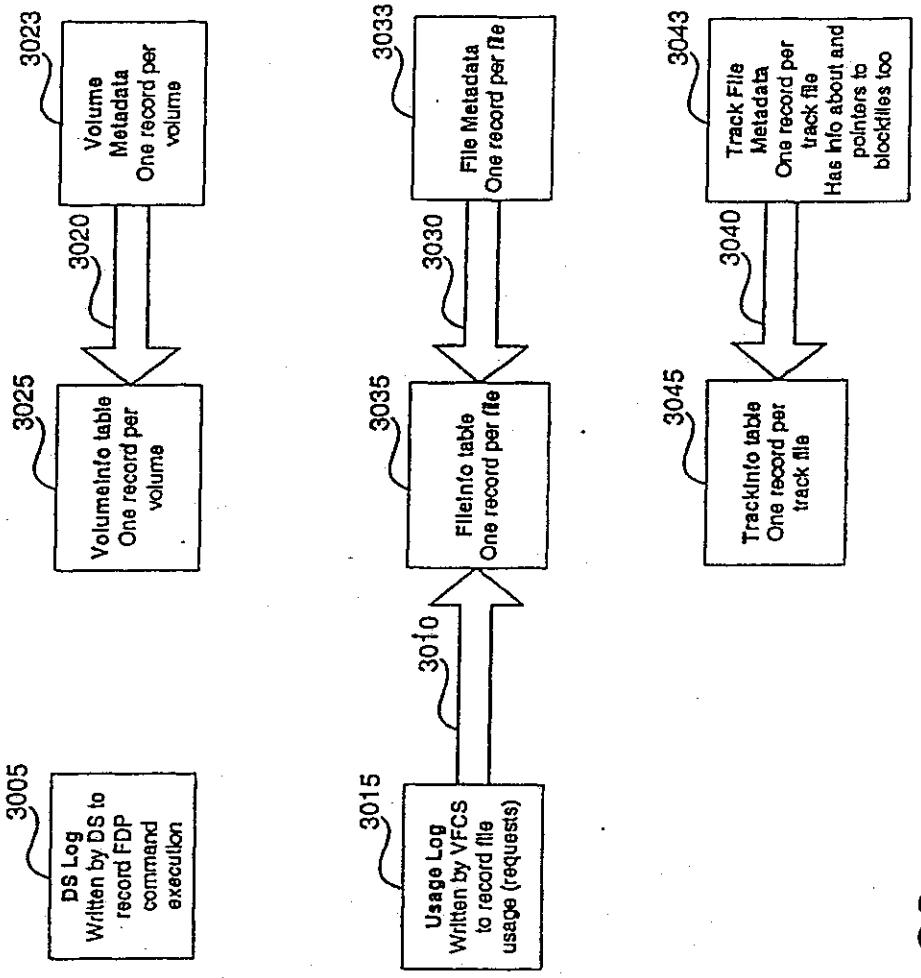

3023
Volume Metadata One record per volume
3020
3025
VolumeInfo table One record per volume
3033
File Metadata One record per file
3030
3035
FileInfo table One record per file
3043
Track File Metadata One record per track file Has info about and pointers to blockfiles too
3040
3045
TrackInfo table One record per track file
3005
DS Log Written by DS to record FDP command execution
3010
3015
Usage Log Written by VFCS to record file usage (requests)

Figure 30

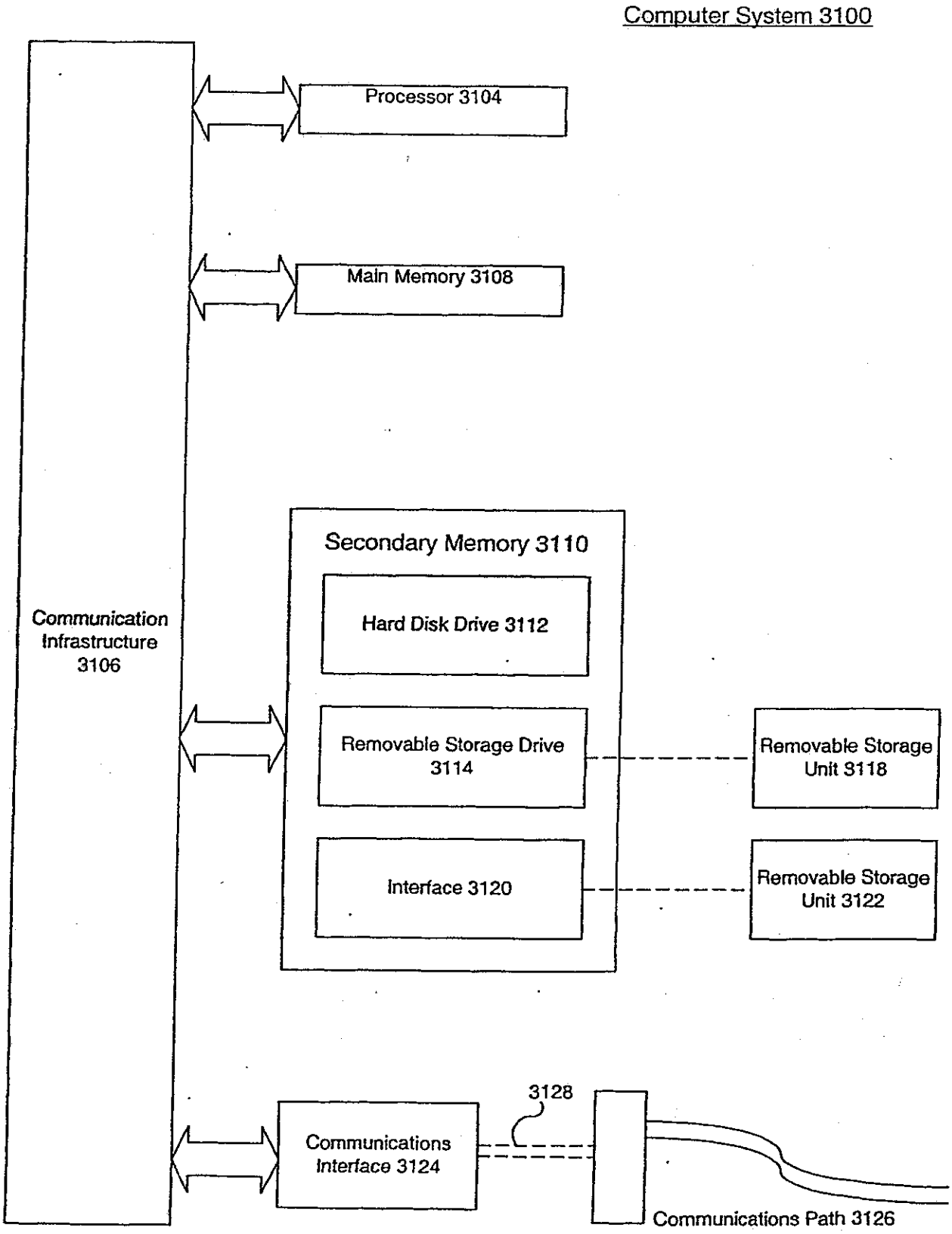

Computer System 3100
Processor 3104
Main Memory 3108
Communication Infrastructure 3106
Secondary Memory 3110
Hard Disk Drive 3112
Removable Storage Drive 3114
Removable Storage Unit 3118
Interface 3120
Removable Storage Unit 3122
3128
Communications Interface 3124
Communications Path 3126

FIG. 31