Derived Virtual Devices: A Secure Distributed File System Mechanism[1]

Rodney Van Meter, Steve Hotz, Gregory Finn
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{rdv,hotz,finn}@isi.edu
310-822-1511

Abstract

This paper presents the design of *derived virtual devices* (DVDs). DVDs are the mechanism used by the Netstation Project to provide secure shared access to network-attached peripherals distributed in an untrusted network environment. DVDs improve Input/Output efficiency by allowing user processes to perform I/O operations directly from devices without intermediate transfer through the controlling operating system kernel. The security enforced at the device through the DVD mechanism includes resource boundary checking, user authentication, and restricted operations, e.g., read-only access. To illustrate the application of DVDs, we present the interactions between a network-attached disk and a file system designed to exploit the DVD abstraction. We further discuss third-party transfer as a mechanism intended to provide for efficient data transfer in a typical NAP environment. We show how DVDs facilitate third-party transfer, and provide the security required in a more open network environment.

1. Introduction

A network attached peripheral (NAP) is a device that communicates with the external world via a network interface, rather than a bus. System buses limit the sharing of devices and do not scale well in bandwidth, distance or number of devices. Communication via a local area network (LAN) provides flexibility in system design and avoids the problems of shared-bus communication, while allowing us to exploit the ever-increasing aggregate bandwidth provided by high-speed networks. These advantages are changing the way computer system architectures are defined [1], and we see NAPs becoming a significant component of new storage systems [2,3] and new multimedia architectures [4,5].

[1] This research was sponsored by the Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

However, components of a system built around a LAN cannot depend on the tight coupling and simplifying assumptions provided by a bus-based architecture. The boundary between the "inside" and "outside" of a system grows fuzzier, and the inherent level of trust that can be assumed among components of the system must decrease. Many current NAP system designs simply treat the LAN like a different type of bus, but do not address this added "open-ness" dimension and the consequent security issues. This may prove problematic unless the type and sources of network traffic are limited to prevent misuse of the NAPs.

The Netstation Project is explicitly addressing the problems inherent in using NAPs in an open network environment. A Netstation is a heterogeneous distributed system comprised of NAPs brought together as a single system operating across a network. One of our primary goals is to support multiple Netstations made up of components connected via a single, shared LAN. The requirement to allow individual devices to be shared by multiple Netstation systems results in additional complexity. Moreover, we have chosen to support IP connectivity of these devices, to allow Netstations to be configured across LAN boundaries. Each of these goals introduces issues of safety and security.

This paper presents derived virtual devices (DVDs) as the mechanism used by the Netstation Project to provide secure shared access to network-attached peripherals distributed in an untrusted network en- vironment. The security enforced at the device through the DVD mechanism includes resource boundary checking, user authentication, and operational restriction, e.g. read-only access. Yet, DVDs enable efficiency by allowing user processes to perform I/O transfers directly from devices without intermediate transfer of data through the controlling operating system kernel. DVDs also support nested or recursive granting of access to the device, allowing file and window systems to run recursively.

The remainder of this paper is organized as follows. Section 2 presents an overview of the Netstation Project and its architecture to provide context for the discussion of DVDs. Section 3 describes the DVD abstraction, command interfaces for management and access, and security mechanisms in detail. In section 4, we illustrate the use of DVDs by presenting the interactions between a network-attached disk and a file system designed to exploit the DVD abstraction. Section 5 shows how DVDs can facilitate third-party transfer for efficient data transfer between NAPs. Sections 7 through 9 discuss related work, the current state of our implementation, and our conclusions.

2. Netstation Environment

The Netstation Project [6] evolved from research on the Atomic LAN [7], a 640Mbps point-to-point switched LAN developed at ISI from parallel computing chips designed by Chuck Seitz and his group at Caltech[2]. The idea behind Netstation is to substitute a gigabit network in place of a workstation bus, similar to the efforts of the DAN [4] and ViewStation [5] groups. Devices such as disks, cameras, displays, and low-bandwidth input concentrators are connected to processing nodes via the LAN. Figure 1 shows a typical hardware configuration.

[2] The Atomic LAN has become a commercial product of Myricom known as Myrinet.

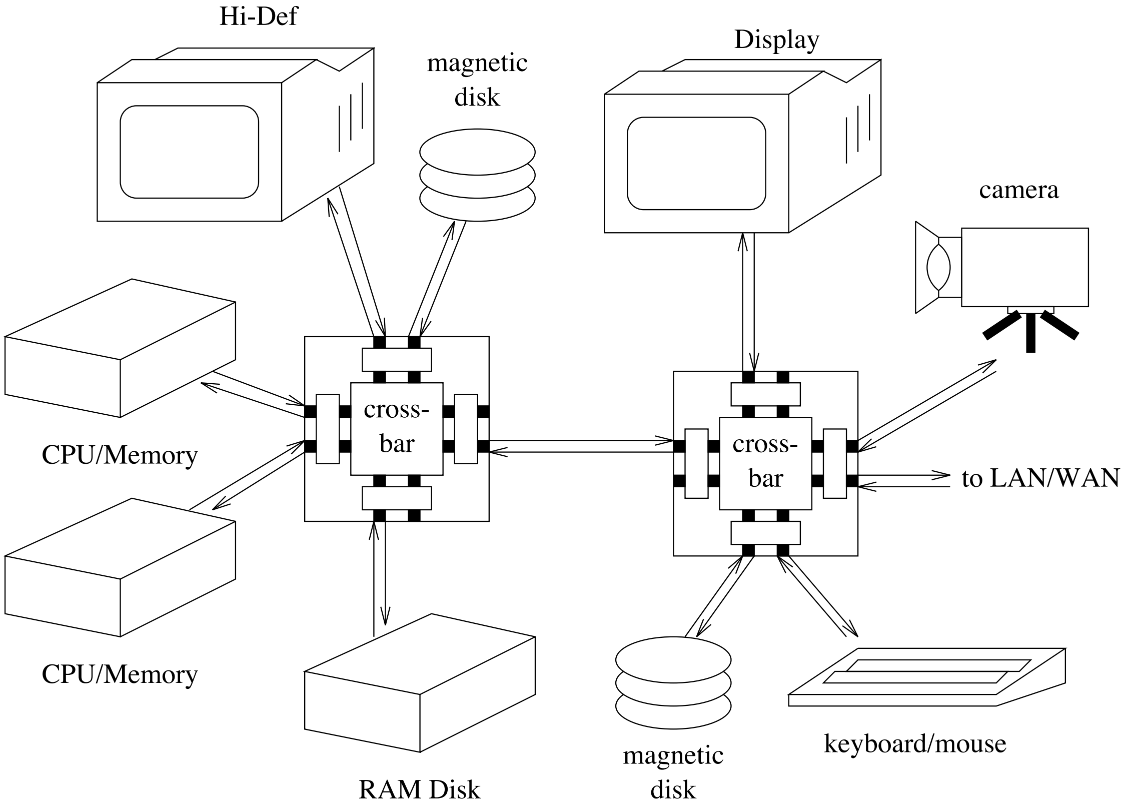

Figure 1: Netstation Architecture

The primary advantage of a Netstation is allowing high-bandwidth devices to communicate directly, alleviating the shared-bus bottleneck. An example application would be the transfer of video data directly from an incoming network port to a network-attached display device, without passing through the main processor or the cpu/memory bus. A second potential advantage is the flexibility afforded by dynamically configuring workstation components from a shared pool of resources.

The Netstation architecture includes (a) two related device abstractions (Network Virtual Devices and Derived Virtual Devices) which provide for the required system functionality, and (b) the management information and protocols needed to access and control the networked devices. The Network Virtual Device abstraction is used for higher level functions such as resource location and naming. Derived Virtual Devices are lower-level abstractions composed of two components: (1) some portion of an NVD resource (an object), and (2) an execution context which provides a functional interface (methods). A concise description of Netstation architectural components and their interactions is given below.

2.1 Network Virtual Devices

A **Network Virtual Device (NVD)** is a named physical device resource that is attached to the network. An NVD is the object granularity for device naming, resource location, and management within a Netstation system. One or more NVDs may be housed in a chassis along with a network media interface and sufficient processing power to provide and manage the interface that is presented on the network. Each NVD is managed individually (i.e. it has a name and a description in a device management database), even though multiple devices may reside at the same network interface[3].

We have chosen to use Internet Domain Names [8] to identify NVDs on the network. The DNS can then provide the mapping from device name to the address of the network interface where the device resides. Given the Internet domain name (and address) of the device, a client that desires to use the device sends requests to a well-known NVD management port.

The functions of system configuration and resource location are built on top of the NVD abstraction. In the simplest case, resource location can simply be obtaining a pre-configured device name. To configure a file system to use a particular NVD disk, the domain name of the NVD would be sufficient to identify the resource. For example, "sd0a.disk1.isi.edu" would replace "/dev/sd0a" in a file system mount table. In the more complex case of dynamically finding available resources, space, the resource location library routines return NVD domain names. This scheme is modular and flexible as different resource location mechanisms can be used depending on the needs of the particular system that requires device resources.

An important point to note is that data-related commands (e.g. READ) cannot be issued to an NVD. Instead, NVDs accept control commands that create and manage a set of abstract devices intended to support data I/O. These abstract devices are referred to as Derived Virtual Devices.

2.2 Derived Virtual Devices

A **Derived Virtual Device (DVD)** is an abstraction of a physical device that is comprised of (a) all or some part of an NVD's resources, and (b) a set of functions that provide access to, and control of, the device resources. DVDs are created (and destroyed) dynamically, and each is accessed through a port number that is unique for the lifetime of the DVD. This port number also serves as the identifier for the DVD resource; the Netstation system does not maintain a persistent DVD identifier similar to NVD domain names[4].

DVDs enforce the bounds checking and operational restrictions required for safe shared access, providing lower-level functionality than the naming and management functions based at the NVD level.

DVDs can be derived from the resources of an NVD, or from the resources of a parent DVD. In the former case, the default information maintained about the NVD is sufficient to

[3] Requests sent to an interface will contain the name of the desired device in order to multiplex between different devices available via the interface.

[4] Services built on top of DVDs may retain persistent DVD information including an identifier for some portion of its resources.

specify the derived device mapping. In the latter case, the user/owner of the parent DVD may be offering a more complex, "value added" abstraction based on its DVD (e.g. a file system based on a DVD representing a set of blocks from a disk NVD). Creating a DVD in this case requires that the owner explicitly specify the portions of its DVD to be shared, and the required mapping into the derived DVD.

Section 3 discusses DVD functions in more detail.

2.3 Device Command and Access

We chose RPC as the communication abstraction since it models the request-response nature of bus-based device interactions. Use of RPC implies that the NVD presents one (or more) procedural interface(s) to client applications. We distinguish between the following two types of RPCs:

- **DVD Management Procedures (DMPs)** are sent to NVDs to control and manage device access through the creation and maintenance of DVDs (e.g. `create_DVD()` and `install_DVD_map()`).

- **DVD Command Sets (DCSs)** provide an execution context for each DVD which allows safe shared access to device resources (e.g. `readblock()` and `writepixel()`).

Authenticated RPC is used to avoid unauthorized device access, where the type/level of authentication can be configured locally and varies according to device type and RPC procedure.

2.4 NVD Management

Netstation systems are tied together with a local database that defines the available NVDs and DVDs. This database is known as **the Network Virtual Device Manager (NVDM)**. The NVDM contains information as follows:

- NVD entries comprised of attribute-value pairs that describe characteristics of the available devices (e.g. NVDname: sd0a.disk1.isi.edu, NVDtype: disk, NVDblock cnt: 65536).

- DVD entries that associate (1) an NVD resource, (2) a subset of the DVD Command Set for the NVD resource, (3) an access control list to specify users allowed to create the described DVD, and (4) an indication of the required level of user authentication.

Multiple DVD entries may exist for each NVD to grant different access privileges to each system user. An NVD definition language defines the permissible NVD attribute-value pairs, and includes an enumeration of the DVD command set universe.

Users of Netstation devices consult this database to locate devices that can meet required specifications. Each NVD must consult this database to obtain configuration and access control information for the DVDs it will support.

3. Derived Virtual Devices

A derived virtual device (DVD) is an abstraction of a physical device that can be viewed as a set of resources and an *execution context* at the device. The execution context enforces the desired constraints associated with the device[5]. Clients of a device see a virtual device, which provides a set of services such as nonvolatile storage of blocks (a disk drive DVD) or display of pixels (a frame buffer DVD). This virtual device is mapped to real physical resources by processing resources at the NVD (i.e. a device controller).

DVDs are created by any entity with access to device resources. We use the term *derived* to indicate that the device is constructed from an already existing grouping of resources. The original resource is referred to as the *parent*, and the new DVD as the *child*. The parent resource can either be an NVD or another DVD. In either case, the access rights granted to a child must be a subset of the parent's access rights; this constrains both (a) the set of resources accessible, and (b) the operations that may be performed on the device. Note that NVDs are strictly a set of resources, and do not have associated data-access procedures. Hence, in the case of a parent NVD, the constraints placed on the Device Command Set must be obtained from the configuration information maintained by the local Netstation management database (i.e. the NVDM).

The owner of a virtual device grants access to others by creating a mapping within the set of resources it owns, sending that mapping to the virtual device to create a new virtual device, granting client access to the DVD, and informing its client of how to communicate with the new virtual device.

The new, secondary client is then allowed to communicate directly with the device (via the new DVD), without the intervention of the granting server. The key to ensuring that the secondary client does not overstep its newly-acquired authority to execute commands at the device is that *the device enforces the constraints* of the new DVD. These restrictions are implemented by creating a customized set of procedures, parameterized with the particular DVD limitations and lacking the restricted operations.

DVDs can be nested; any client that has access to a DVD can create a child of that DVD. Although the focus is the use of DVDs for mapping files, once the client has access to the DVD it may assign any meaning to the blocks it chooses. Because a DVD presents the same interface as its parent, it is possible to run systems in a recursive fashion; file systems can be built on DVDs created by other instances of the file system, as in stackable filing [10], or window systems can be run on virtual displays that are actually windows on larger virtual displays, much like Plan 9's 8-1/2 [11]. For example, nesting is also used when the file server grants access to a user process, which can then grant access to other devices to facilitate third-party transfer, as described in section 5.

[5] DVDs are related to the concept of virtual store as defined in the Open Storage Systems Interconnection (OSSI) model [9]. Differences are noted in section 7.

Section 2.3 introduced the two types of DVD RPCs: DVD Management Procedures and DVD Command Sets. The following sections discuss these interfaces in more detail.

3.1 DVD Management Procedures

We have defined a protocol which describes the full functionality of a DVD. It is a set of commands used for controlling DVDs which we collectively call the DVD Management Procedures (DMP). These are the commands sent to NVDs which manipulate the DVDs themselves, rather than perform actual I/O operations.

The `create_DVD()` command is the most critical of these procedures. To fully specify DVD creation requires all of the following information:

- An ID of the user that is to be granted access to the new DVD.
- A set of resources. This includes an identifiable resource (either an NVD or existing DVD), and a specification of the resource subset to be accessible by the new DVD.
- A (possibly trivial) mapping from the new DVD address space (e.g. block numbers) into the physical resources.
- The subset of DVD Command Procedures that the user is permitted to use (e.g. cannot use write function).
- Ranges for parameters values for each DVD RPC to enforce constraints.
- Authentication level/type required for each DVD RPC.

All of this information is required so that the DVD creator can establish access constraints on the DVD.

Other commands allow the creator to modify the operating environment of the DVD. For example, it must be possible to dynamically increase the size of a child DVD which represents a file mapping (`install_DVD_map()`). This is superior to the simpler alternative approach where the DVD must be destroyed and recreated, forcing the client to reconnect.

The creator must also be able to determine (normally at child DVD destruction time) certain information about the usage of the child DVD. For example, it may be necessary to receive a list of the blocks that were written to the child DVD, a feature necessary for effective implementation of write before read (described in section 4.4).

Examples of semantic constraints that DVD creators must be able to specify include address remapping and limits and access control features such as read only, write before read and append only (for tape), and restrictions on management operations such as modification of NVD owner lists.

The semantic flexibility provided means that it is possible to define new commands, which might be useful for compression, encryption, storage allocation, parity computation for distributed RAID [12,13,14], and "composite" virtual devices (striping for disks or tapes, treating multiple displays as a single large display, etc.).

Our prototype DVD creation mechanism is based on Scheme [15]. The DVD creator downloads Scheme code at create time, specifying a Scheme function to be executed before and after the execution of each command at the NVD, to adjust argument values (block addresses, etc.) and determine if permissions would be violated by executing the command. Note that use of such a language in a non-prototype environment would raise security concerns that must be addressed. In principle, any of the currently proposed "safe" languages (Java, Safe-Tcl, Penguin, etc.) could be used; for ease of implementation we chose Scheme.

3.2 DVD Command Sets

DVDs provide a set of "data-related" or "I/O-related" commands that can be executed. We refer to these as the DVD Command Sets (DCSs). The interface provided is of course device-specific, and the same device may in fact present several levels of interface. A disk drive, for example, may present a lower-level block-oriented interface, such as the Small Computer Systems Interface (SCSI) or Intelligent Peripherals Interface (IPI), or a higher-level file-oriented interface such as NFS. A display may present a simplified pixel-oriented interface, or a high-level interface that includes windowing functionality, font management, etc., as X Windows does. In general, Netstation devices provide lower-level, device-oriented interfaces. The choice of interface for RPCs executed at the DVD for data I/O is, to a certain extent, orthogonal to the DMP.

As an example, the RPCs appropriate for a disk drive patterned on a SCSI interface include:

- data operations: `READ, WRITE, ERASE, COPY, VERIFY`
- block management: `FORMAT UNIT, REASSIGN BLOCKS, READ DEFECT DATA, READ CAPACITY,` error level control, etc.
- buffer management: write caching policy, replacement algorithm, full/empty ratios for initiating data transfer, etc.
- physical control: `TEST UNIT READY, START/STOP UNIT` (spin up and spin down, eject), and `PREVENT/ALLOW MEDIUM REMOVAL` (for removable drives), etc.

In the normal SCSI model, `READ` returns data to the original requestor, and third-party copy is a complex variant of the `COPY` command. In Netstation, DVDs simplify addressing of data blocks, allowing commands such as `READ` to simply and nearly transparently become third-party transfers. Third-party transfers are discussed in Section 5.

3.3 Security

The havoc that can be wreaked on a disk drive by misuse of commands such as `FORMAT` greatly exceeds that of `TEST UNIT READY`. Thus, the level of authentication and privilege required to execute commands differs.

The level of security required to execute specific RPCs is established at DVD creation. Two factors, the level of authentication and the level of integrity, can be specified independently for each of the two parts of an RPC, the RPC control block and the data. The two parts are controlled separately because they transit the network separately, and may even have different destinations, as in third-party transfer. The large size of most data segments, compared to the RPC control block, also makes it desirable to allow data to be transferred without compute-intensive operations, such as encryption.

Several levels of authentication are provided. Execution of some commands requires no authentication. Others may use known weak methods such as host source address, which has the two major flaws of being spoofable and not unequivocally identifying *who* at a particular node issued the RPC. The examples in this paper assume that the Kerberos authentication system is used, which we expect to be a common mode of operation.

The integrity of the data transferred can also be selected. Some RPC control or data blocks may be protected only by the network's built-in mechanisms, such as checksumming, which can protect the data against accidental corruption in the network but not malicious tampering, while others require that the integrity of data be assured (perhaps via a one-way hash), a common choice for the RPC command block. Still others may require that all data be protected from modification. Management functions (such as the creation of new child DVDs) typically require the highest possible protection.

4. A DVD File System

DVDs can be used as an enabling technology in file systems. We refer to our file manager as STORM (STORage Manager). In this section, we detail several system operations, including booting a device, booting the file system itself, reading a file, and extending a file for writing.

Note that transport-level network overhead is not included in these diagrams. As these messages are typically sent reliably, additional packets for connection setup and control may be required. However, these message sequences do include some infrequent operations such as acquisition of an authentication key; such sequences will typically not have to be executed for every operation.

4.1 Booting a Device

When a Netstation device boots, it must configure itself, including determining who is allowed to access it. Some of this information must be retrieved from the device's NVDM. The device's built-in configuration must be adequate to allow it to find and communicate securely with its NVDM. The information the device starts with (stored in nonvolatile RAM or otherwise statically configured) includes the identifier of its NVDM and a secret key it shares with Kerberos. Because this secret key will unequivocally authenticate the Kerberos server, which will authenticate the NVDM, it is not necessary to know the locations of the

Kerberos server and the NVDM; the locations may determined dynamically, perhaps by a multicast on the local network. The steps involved are, taking a disk as an example (see figure 2):

1. The disk authenticates itself to Kerberos.

2. The disk receives a Kerberos ticket to access the Ticket Granting Server (TGS).

3. The disk requests a ticket to access its NVDM.

4. The disk receives the ticket.

5. The disk requests its DVD configuration and Access Control List (ACL) from its NVDM.

6. The NVDM sends configuration info to disk.

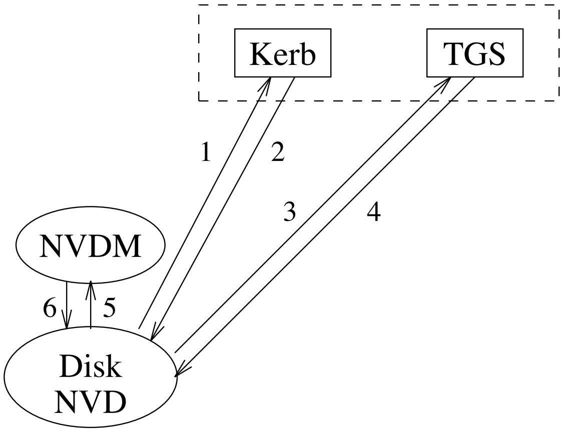

Figure 2: Booting a Disk

In our simple example, the disk receives an ACL indicating that STORM is the only user allowed, and it has unlimited access to the entire disk.

4.2 Booting STORM

Booting STORM requires the following steps (see figure 3):

<u>**STORM authenticates, asks for DVD**</u>

1. STORM must authenticate itself to Kerberos, the authentication server.

2. STORM receives a Kerberos ticket to access the Ticket Granting Server (TGS).

3. STORM requests a ticket to access the disk.

4. TGS sends STORM the ticket, which contains, among other information, a session key for STORM and the disk to share.

5. STORM requests access to the disk NVD. This is a `create_DVD()` request. In the simple case of the disk containing only a file system managed by STORM, this request will be for unlimited access to the entire disk.

6. The disk checks the permissions, creates the DVD, and returns the DVD identifier to STORM. Setup is now complete, and STORM is free to access the disk NVD, subject to the constraints imposed by the DVD definition.

Data transfer

7. STORM requests a read of the file system superblock.

8. Data is returned.

9. STORM requests a read of the block containing the file system root directory's inode.

10. Data is returned.

11. STORM requests a read of the block(s) containing the root directory.

12. Data is returned.

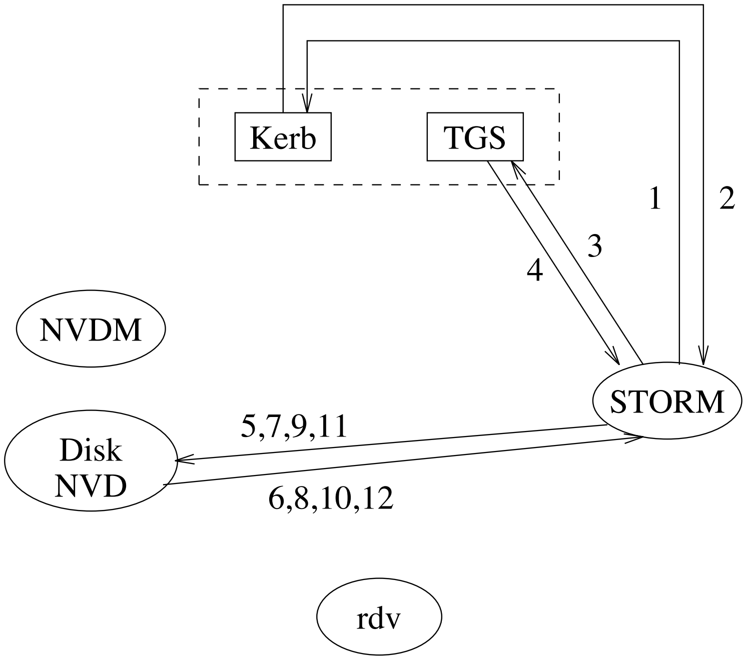

Figure 3: Booting STORM

Note that near the end of the sequence, once the DVD has been established, data requests and responses are processed with a minimum of messages. This is typical of DVD operations; a large number of control messages are used to establish safe conditions for high-speed data transfer. This will be most effective when large amounts of data are to be transferred or many requests executed.

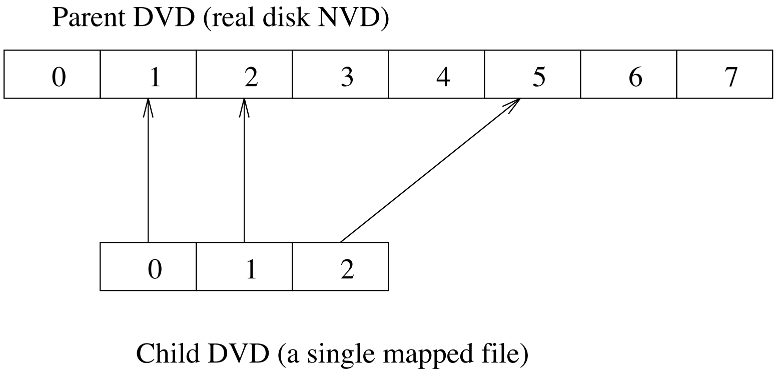

Figure 4: Using a New DVD for a File Mapping

4.3 Reading a File

When an application program opens an existing file, the request is transmitted to STORM. STORM, as the owner of the DVD holding the entire file system, creates a child DVD (with an access list specifying the new user) that includes only the blocks that are part of the file, and returns a DVD identifier (port number) to the new DVD. Figure 4 shows a newly-created DVD that maps a simple three-block file.

Figure 5 shows the steps involved in opening a file through STORM:

1-4. rdv authenticates himself to Kerberos and acquires a ticket to access STORM. This is analogous to steps 1-4 of booting STORM.

<u>Establish DVD for rdv</u>

5. rdv sends a file open request to STORM.

6. STORM determines that the best way to handle this request is to create a DVD for rdv at the disk drive, so it sends a `create_DVD()` command to the disk NVD. This command, detailed in section 3.1, contains information about who the DVD is for as well as what access is being permitted.

7. The disk ACKs the DVD create with the appropriate information.

8. STORM bundles the DVD identifier into a package and sends it to rdv. STORM may have to include extra information for the file system library code being

executed by rdv, such as what operations require the cooperation of STORM, how to handle partial blocks, what to do about EOF, etc. DVD setup is now complete.

rdv gets a ticket

9. rdv, who has not previously accessed the disk, requests a ticket for this purpose. If subsequent file opens access the same disk, this step will not have to be executed.
10. TGS returns the ticket.

Data transfer

11. rdv sends his first data request to the disk NVD.

12. The disk NVD responds with the data.

13. rdv sends his second data request to the disk NVD.

14. The disk NVD responds with the data.

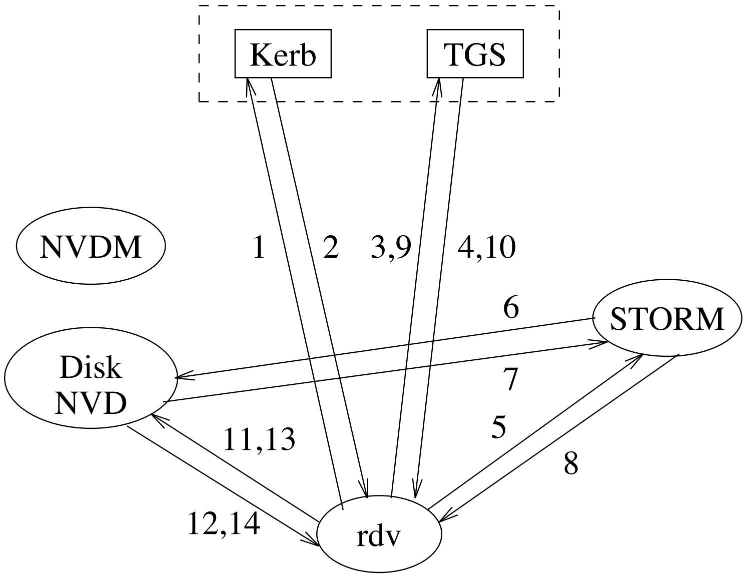

Figure 5: Opening a STORM File

The use of DVD file systems is most efficient for applications in which the data transfer phase is the primary performance bottleneck. The process of opening a file should happen only rarely compared to the number of read/write operations to be performed on the file. If that is not the case (e.g., an application that opens many small files), a normal file system RPC is likely to be more efficient. A storage manager can maintain file-size information to recognize small file requests. Then, rather than establishing a DVD, it can retrieve data and forwards it to the client similar to a conventional NFS interaction.

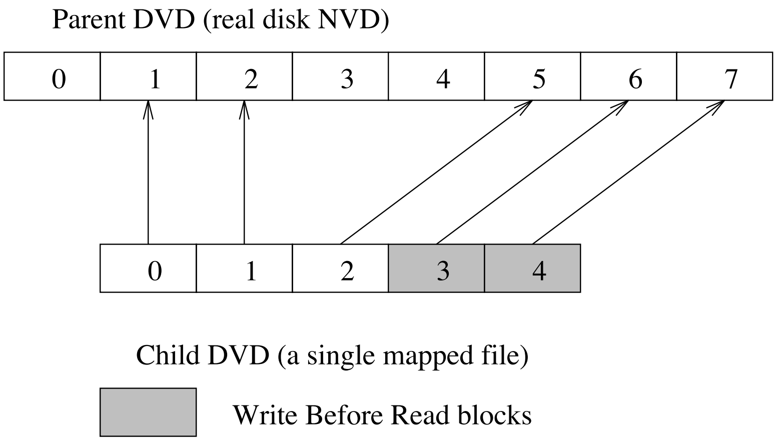

Figure 6: Write Before Read DVD for a File Mapping

4.4 Write Before Read

An optimization we have developed in conjunction with DVDs is *write before read* (WBR). It allows servers to grant access to resources containing sensitive data, without disclosing that data, and without requiring explicit, expensive erase operations.

In a traditional kernel-based system, new blocks are allocated to a user's file when writes are made to the file, or, depending on the FS implementation, when the file's size is extended but not all of the blocks are written. These unwritten blocks cannot be read until they have been written, because they may have once been allocated to a different (now deleted) file containing someone else's private data. This constraint is enforced by the kernel and file system. The safest solution is of course to erase the blocks explicitly before granting access, however this has a large negative performance impact. Thus, the concept of write before read comes into play.

The DVD abstraction allows STORM to create a DVD representing a file, and tailor the DVD Command Set so the `WRITE` procedure is parameterized to enforce the WBR restriction. STORM simply provides (as an optional argument to a create DVD RPC) a list of blocks to be written before they are read.

Figure 6 shows the same file from figure 4, extended two blocks, presumably as a result of a client request to read or write past the end of the physical file allocation. The server that lengthened the file (the owner of the child DVD) marked the two new blocks as WBR, since it knows that those blocks may contain data from having previously been used as part of another file.

When a child DVD is destroyed, the write-first list must be reconciled with its parent. This is executed at the parent DVD. The owner of the parent DVD can request notification of the destruction of the child DVD, and along with it an accounting of blocks that remain unwritten, data which it can use when creating its next child DVD.

5. DVD Third-Party Transfer

Third-party transfer is a mechanism that specifies movement of data, where the party requesting the transfer is neither the source nor the destination of the data. This is a common mechanism in NAP systems that provides support for efficient data transfer between devices, without a copy through the controlling entity.

In this section, we show how DVDs support third-party transfer by presenting an example transfer from a disk drive to a display DVD.

In a Netstation, third-party transfer differs from a primary transfer only in that the locus of control is different; the mechanics of the transfers are the same. It does, however, result in an increase in the number of messages transferred across the network.

One DVD can transfer data to another. This can be done by creating two DVDs, one for the source and one for the destination, that each linearize the area to be transferred, creating a *virtual mapping window* in a fashion similar to the Parallel Transport Protocol (PTP) [16]. As with PTP, the mapping to create the virtual mapping window is done at the storage server, rather than at the device. However, using DVDs, this mapping is then communicated to the devices in the form of the creation of child DVDs. The mapping is then enforced by the child DVDs themselves.

Using a DVD as the destination has the advantage that improper behavior by the source of the data cannot corrupt data at the sink. Giving the source device unlimited access to the destination can allow overwriting or erasing of data if the source misbehaves due to programming errors, concurrency conflicts, or malicious misuse of the source. An important point is that the destination device does not have to trust the source, only the storage server from whom it receives the mapping it enforces. This helps limit the damage that can be caused by security breaches at the devices, though the storage server itself remains the ultimate key to overall system security.

5.1 "Push" Transfer

Figure 7 shows the operations necessary to initiate one type of third-party transfer, "pushing" data from a disk drive to a display. This figures assumes (1) rdv has already opened the file on the disk as detailed in section 4.3, and (2) the display has already booted and retrieved configuration information as explained in section 4.1). From this point, the steps in establishing the connection are:

1-2. rdv gets a ticket to talk to the display.

3. rdv sends a `create_DVD()` request to the display, requesting write access to the whole screen for himself.

4. The display ACKs the create with the appropriate information.

5. rdv sends a `create_DVD()` command to the display, giving the disk NVD write access to a rectangular region of the screen, and mapping it so that (0,0) for that DVD maps to the upper left hand corner of the rectangle. This simplifies the disk's access to the display.

6. The display ACKs the create with the appropriate information.

7. rdv sends a third party copy command to the disk DVD he has access to, requesting that the disk drive send data to the display. This first request includes the ticket and DVD identification information the disk needs to access the display, but that information does not need to be transferred for subsequent requests.

8-9. The disk has not accessed the display, so it gets a ticket from TGS.

10. The disk sends data to the display.

11. The display ACKs the command to the disk.

12. The disk ACKs the command to rdv.

Note that subsequent requests can execute much more quickly, since the DVD state is preserved; this eliminates steps 3, 4, 5, and 6. Note also that interactions with Kerberos and TGS may be eliminated for subsequent setups if valid tickets are still held. Caching the ticket eliminates steps 1, 2, 8, and 9, leaving a eight-step process instead of twelve. Additional requests from rdv for data transfer result in four messages:

13. rdv requests the disk drive to transfer data.

14. The disk drive sends the data to the display.

15. The display ACKs the command to the disk drive.

16. The disk drive ACKs the command to rdv.

This is the bare minimum of messages possible. STORM has not had to be involved at all in this child DVD create or the individual I/O operations, because rdv is granting access to resources he already has access to.

Note that this is asymmetric; the disk drive has access to the display, but not vice-versa, because no DVD allowing the display to access the disk drive has been set up.

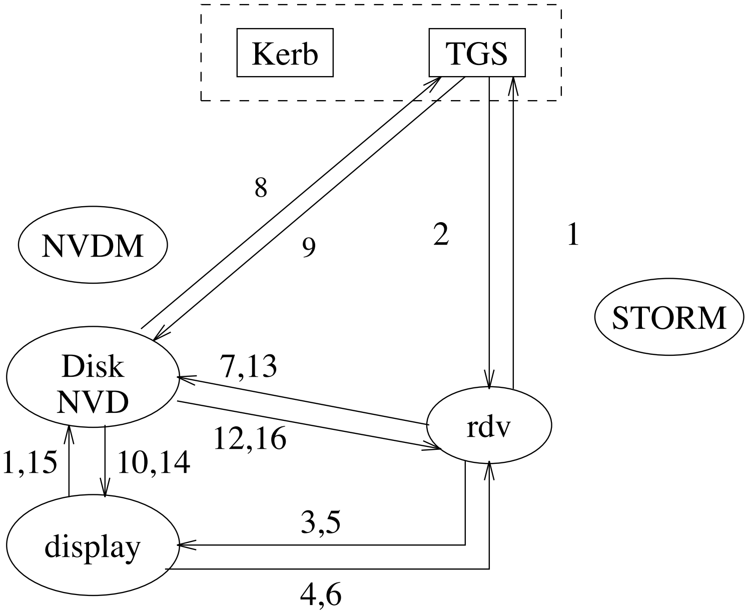

Figure 7: Third Party "Push" Transfer

5.2 "Pull" Transfer

The previous example was shown as a "push" transfer, with the data source initiating the transfer. An equivalent transfer can be set up in the opposite direction, with the display sending `READ` commands to the disk drive rather than the disk drive sending `WRITE` commands to the display. This we refer to as a "pull" transfer.

The choice of whether to use a push or pull transfer can be made based on the relative capabilities of the two nodes. If rdv's latency to the two devices is significantly different, the choice can be made to reduce the total time for the four messages necessary for each transfer. A push transfer would be appropriate if rdv has low latency to the disk drive and high latency to the display, and a pull would be the correct choice in the opposite case.

6. Implementation Issues

The client of a DVD (for example, a user process) accesses the DVD as if it were a regular block-oriented device. Library code would implement `read()` and `write()` transparently to application code, thus preserving the investment in software development. A relink may be required, however.

This library code will run entirely in user mode; once the mapping of the file has been done and the DVD created by the device's owner, no further communication with the owner (the storage manager) is required. Reads and writes are done via user-level RPCs directly to the DVD. If the network code runs in the user's context as well (as is done on some high-performance systems), file I/O may be executed entirely without leaving the context of the application. This can be especially useful on systems that provide low latencies on RPCs.

The library read and write code maintains some structures similar to those normally handled by the kernel file system, such as the EOF marker, which must be returned to the true file system at file close (or process termination) time.

The ability to execute file I/O without the intervention of another process or kernel may be especially useful on distributed systems or on multicomputers, where the file system manager may not be local to the client's node. This allows separation of the operations for actually executing I/O from those for managing disk space, directory structures, etc., which may be centralized or distributed without regard to where the I/O must be conducted.

This limited-functionality library will also result in less memory use at the compute nodes. On massively parallel processors (MPPs), for example, the memory savings of not running the full file system code locally on each of a thousand nodes can result in savings of tens to hundreds of megabytes of RAM.

When the process attempts to write past the end of the existing block allocation, the library code recognizes this, and communicates a request to the file manager to extend the file. The file manager then allocates additional data blocks and communicates an updated file mapping to the child DVD. Should the application (either deliberately or through a mistake in the library code) attempt to read or write past the end of the child DVD, an error is returned. See section 4.4 for more details.

7. Related Work

The projects most similar to the basic concept of Netstation are MIT's ViewStation [5] and Cambridge's Desk Area Network (DAN) [4]. Both projects are ATM-specific, and concentrate more on local-area traffic, with careful distinctions between the "inside" and "outside" of the system, whereas Netstation is fully Internet-accessible and has no system boundary.

As already discussed, DVDs have much in common with *virtual stores* from the IEEE Open Systems Storage Interconnect model [9,17]. DVDs differ from virtual stores in several respects. DVDs do not support composite devices, while a virtual store may represent striping across more than one disk, for example. However, DVDs provide more semantic flexibility, in that the owner of a device (or DVD) is allowed to grant any arbitrary subset of its own capabilities (including management functions) to its children when creating DVDs, while virtual store is limited to a data mapping of storage devices. Moreover, although the focus of this paper is on DVD use in file stores, DVDs are more general and may be used for other network attached peripherals such as displays.

Numerous projects have proposed giving the disk node more autonomy, as part of parallel file systems [18,19] or to execute their own space allocation [20,21]. DVDs, with their flexibility and programmability, provide a platform which could be used for similar purposes.

8. Status and Future Work

Much remains to be implemented before Netstation can be considered complete. The network-attached display hardware is complete, and programming of it nearly so. An implementation of the X Window System using the display, with a prototype implementation of the DVD definition mechanism, is complete. A prototype software version of the keyboard device is under way. Design of the hardware for the camera is under way.

STORM itself, and the user library that accesses it, are in the early stages of implementation. Early goals for the implementation include the ability to use third-party transfer to move data to and from the display, via DVDs. The details of the API for file-related and non-file I/O are still in development. The Kerberos authentication system has not yet been incorporated into the system.

Future research includes defining a composition function so that multiple devices can behave as a single virtual device. As mentioned above, DVDs are typically derived from a single device, however it is desirable to provide a higher-level abstraction to create composite devices.

9. Conclusion

We have shown the design of a device abstraction, derived virtual devices, which provides the efficiency of low-level device access while maintaining many of the protections of higher-level abstractions such as files. We have described a file system design based on DVDs which supports third-party transfers from device to device and allows direct access to the devices by clients at all levels. Derived virtual devices also recurse to allow clients to safely grant access to subsets of their resources to their clients.

References

[1] Rodney Van Meter. A brief survey of current work on network attached peripherals (extended abstract). ACM Operating Systems Review, pages 63-70, January 1996. Full version available on the web at http://www.isi.edu/~rdv/nap-research/index.html.

[2] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In Proc. Fourteenth IEEE Symposium on Mass Storage Systems, pages 27-44. IEEE, September 1995.

[3] Randy H. Katz. High-performance network and channel based storage. Proc. IEEE, 90(8):1238-1261, August 1992.

[4] P. Barham, M. Hayter, D. McAuley, and I. Pratt. Devices on the desk area network. J. Selected Areas in Communications, 13(4):722-732, May 1995.

[5] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet desk area network: Architecture, implementation and experience. J. Selected Areas in Communications, 13:710-721, May 1995.

[6] Greg Finn. An integration of network communication with workstation architecture. ACM Computer Communication Review, October 1991. Available on line at ftp://venera.isi.edu/atomic-doc/ATOMIC.Netstation.ps or http://www.isi.edu/netstation.

[7] R. Felderman, A. DeSchon, D. Cohen, and G. Finn. ATOMIC: A high speed local communication architecture. J. High Speed Networks, 3(1):1-29, 1994.

[8] P. V. Mockapetris. Domain names - concepts and facilities. RFC 1034, USC Information Sciences Institute, November 1987.

[9] IEEE P1244. Reference Model for Open Storage Systems Interconnection - Mass Storage System Reference Model Version 5, September 1994.

[10] John Heidemann and Gerald Popek. Performance of cache coherence in stackable filing. In Proceedingd of the 15th Symposium on Operating Systems Principles, pages 110-127. ACM, December 1995.

[11] Rob Pike. 8-1/2, the plan 9 window system. In Proc. Summer 1991 USENIX Conf., Nashville, June 1991.

[12] John H. Hartman and John K. Ousterhout. The zebra striped network file system. ACM Transactions on Computer Systems, 13(3):274-310, August 1995.

[13] Pei Cao, Swee Boo Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP parallel RAID architecture. In Proc. 20th Annual International Symposium on Computer Architecture, pages 52-63, May 1993.

[14] Darrell D. E. Long, Bruce R. Montague, and Luis-Felipe Cabrera. Swift/RAID: A distributed RAID system. Computing Systems, 7(3):333-359, 1994.

[15] William Clinger and Jonathan Rees (editors). Revised^4 Report on the Algorithmic Language Scheme, November 1991.

[16] Lawrence Berdahl. Parallel transport protocol proposal. Lawrence Livermore National Labs, January 3, 1995. Draft. ftp://svr4.nersc.gov/pub/Pio-1-3-95.ps.

[17] IEEE P1244. Virtual Storage Architecture Guide, March 1995.

[18] Peter C. Dibble and Michael L. Scott. Beyond striping: The Bridge multiprocessor file system. Computer Architecture News, 19(5), September 1989.

[19] David Kotz and Nils Nieuwejaar. Flexibility and performance of parallel file systems. ACM Operating Systems Review, 30(2):63-73, April 1996.

[20] Robert M. English and Alexander A. Stepanov. Loge: A self-organizing disk controller. In Proc. Winter '92 USENIX, pages 237-251, January 1992.

[21] Garth Gibson. Secure distributed and parallel file systems based on network-attached autonomous disk drives. White paper, September 1995.