NVD Research Issues & Preliminary Models

Gregory G. Finn, Steven Hotz, Rod Van Meter
USC/Information Sciences Institute
March 1995

Version of: 9/1/95

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.

UNIVERSITY OF SOUTHERN CALIFORNIA
INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way Marina Del Rey, CA 90292

1. Overview

Microprocessors are now often used in peripheral devices. Recent advances in multicomputer research have produced chip-sized gigabit network interfaces that operate across spans from a centimeter to 100s of meters [1][2]. Since a gigabit local-area network (LAN) has roughly the same channel capacity as a system bus, it is now reasonable to consider the substitution of a gigabit network in place of a system bus.

In conventional computer architectures the main processor communicates with its major peripheral devices via a system bus. In a *netstation architecture* the communication function of the system bus is replaced by a high-speed internetwork [3]. Peripheral devices communicate with the hosts that control them via the network. Netstation architecture blurs the physical boundary of a computer system. It also blurs the definition of what a host is or is not.

The advantages of utilizing a gigabit network rather than a bus are improved scaling and performance combined with vastly greater accessibility. Networks provide symmetric communication and support simultaneous transfers. Data can be exchanged between any two hosts directly. By relying upon Internet protocols, devices attached to the network are directly accessible to any other host on the internetwork.

To access and control devices across a network requires the development of commonly understood network protocols and interfaces. The client/server model can be extended to apply here. The focus of netstation research is the design and implementation of efficient communication and control mechanisms between the device-client, that controls a device, and the device-server, that presents via the internet an interface to the physical device.

2. Problem Outline

The network interface in the past was not seen as a principal part of the system architecture. The network channel was treated as a slow-speed peripheral device. However, with the advent of gigabit networks, the network is no longer a slow-speed device and it operates at speeds that approximate main-system memory bandwidth. To achieve reasonable performance, a gigabit network interface must be tightly coupled to main-system memory.

In a netstation architecture the principal peripheral device is the network interface. The system bus is largely replaced as a key communications medium by the internetwork. To achieve this, peripheral devices such as disks, displays, keyboard with mouse, are made autonomous nodes on the internet. The operating system or application process that accesses and controls peripheral devices does so via messages that are sent across reliable transport protocol connections.

This architectural shift toward the network as the dominant peripheral communications medium requires both an adaptation of existing kernel device-control methods where practical and creation of new control methods that reflect the physical and administrative differences of a message-based distributed-system architecture. Questions of system configuration, naming, resource discovery and access control must be addressed.

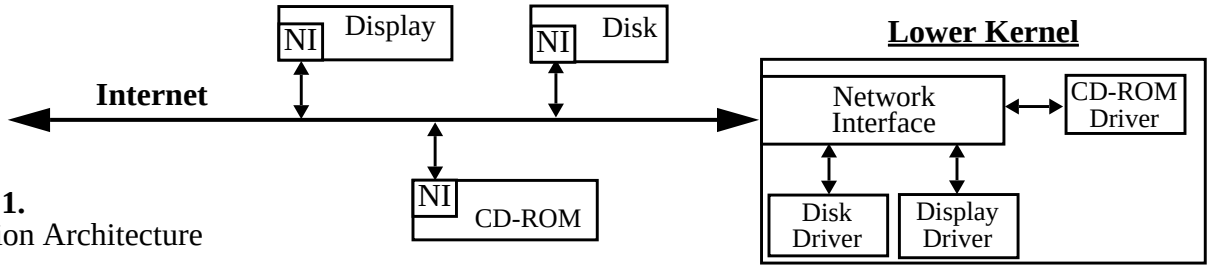

Figure 1.
Netstation Architecture

UNIVERSITY OF SOUTHERN CALIFORNIA
INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way, Marina Del Rey, CA 90292 9/1/95

2.1 Unit of Transfer

The fundamental unit of data transfer across a bus is a word and the fundamental bus data-transfer operations are load and store. Transfer latency across a bus is a few nanoseconds and the transfer overhead a few tens of nanoseconds typically. The fundamental unit of data transfer across the internet is a packet. The packet is a grossly larger unit of communication than a word. Transfer latency across the internet varies from several microseconds to tens of milliseconds, and the typical transfer overhead at the source and destination is currently at least several microseconds.

It is possible to access and control devices that are non-resident with little or no change in device-control software by using distributed memory. However, the performance obtained using distributed memory in an internet setting would be poor, since the vast majority of hosts provide little or no support for it. A message-passing model of distributed computation is better adapted to the internet operating environment.

Though message passing is better suited to the internet environment, differences in overhead and latency restrict the types of devices that are suitable for network distribution. Transfer latency across gigabit networks is limited by path length. Little can be done to significantly improve that. For some device types, that latency will limit the maximum practical separation between the client and device. Unlike transfer latency, reducing the per-packet overhead is not an intractable problem. In the context of high-speed networking it is being actively studied and remains a fruitful area of research.

2.2 Means of Transfer

In the context discussed here, we assume that data and commands that pass between a device and its controlling application occur via messages that are sent across the internet. The reliability and fidelity of the command transfer across the internet must closely approximate that obtained crossing a system bus. Commands sent to the device sub-system must exhibit execute once (and only once) semantics. A straightforward way to accomplish this is for the application to issue commands using remote procedure calls (RPCs) that are carried via a reliable transport-layer protocol.

The rate that commands can be issued is limited by that request/response time. Reducing the packet-related overheads between the controlling application and the netstation node that it is using is particularly important. TCP is not well-adapted to the task of transferring RPCs to an application. It provides a byte-stream to the source and destination applications, rather than a sequence of objects framed by the source. The delay-bandwidth product across a high-speed LAN is also often less than even a small RPC packet. A specialized transport protocol that is designed to carry RPCs may reduce packet-related overhead by allowing RPCs to be individually framed.

The External Data Representation and Remote Procedure Call standards as documented by RFC 1014 [4] and RFC 1050 [5] are sufficient standards that allow creating the RPCs used to control devices. The RPC standard in RFC 1050 is not transport layer-specific. Implementations exist for more than one transport protocol.

Industry is moving towards device-command formats that are suitable to the networking domain. An example of this is the SCSI-3 standard for command of devices that is being developed by ANSI/ISO [6]. SCSI-3 defines the syntax framework of commands sent to target devices for execution, their semantics, certain aspects of error reporting and recovery, and the behavior of queueing at the target device for environments where multiple requests may be outstanding. Methods for access ordering are covered, but access control, ownership and authentication issues are not discussed.

The SCSI-3 committee has considered the transporting of commands over a variety of different media, including high-bandwidth point-to-point channels and networks. As a result, SCSI-3 commands adopt a

client/server model that should allow a straightforward implementation via RPCs. In particular, a proposed draft of a Generic Packetized Protocol (SCSI-GPP) [7] has been defined that resides between the SCSI-3 device driver as client, the network, and the SCSI-3 device itself as a server or target for those commands.

2.3 Network Virtual Device

In a netstation architecture, device sub-systems reside within nodes on the internet. A node exports network service interfaces to the device sub-systems that it contains. These interfaces are called *network virtual device*s (NVDs) to distinguish their external representation of a sub-system from its actual physical interface within the node.

An NVD is associated with named context in its node. Each context supports one or more RPC *program*s and a set of *procedure*s associated with each program. These contexts will generally be leght-weight, entered and exited as the result of up-calls and down-calls from the transport-layer. Once a connection to a specific NVD is opened, the connection is associated with that NVD's context. RPCs that arrive over the connection are executed in that context.

The relationship of an NVD to its external users parallels that of a server to its clients. A workstation as the client can use an NVD to emulate a conventional peripheral device. After the network interface is brought up and the necessary network services are spawned, the workstation kernel opens a connection to the NVD and attaches it. Since the number of simultaneous connections allowed can be large, the number of peripheral devices that a host or process can control via NVDs is not rigidly bound.

A particular device sub-system or group of sub-systems within a node may present more than one NVD to the network. This allows presentation of differing access methods to, or views of, sub-systems that are contained within the node. New NVDs can also be spawned within a node in response to changing conditions. If the controlling owner of an NVD wishes to make that NVD visible to third party, the owner could spawn a new connection to the same NVD or create a new NVD that has somewhat different properties. For example, the controlling application of a read/write disk NVD could spawn a connection for the third party to an NVD that is restricted to read-only access.

2.3.1 Aggregation by Function

Different physical devices that perform a similar function may present the same NVD interface. Although intrinsically different devices, CRT, LCD and field-effect displays perform a similar function. A single Display NVD interface could be defined that allows each to be controlled via the same family of RPC calls. Ideally, it should be possible to write a display application, such as a window-system server, without the need to incorporate specialized support for the several display technologies that it may be called upon to access, the device-dependencies being captured within the NVD and not exported.

2.4 Access Control

Peripheral devices have control-register and buffer memory locations that are well-known to the device-control software in non-distributed architectures. Direct control of devices on a bus is explicitly limited to the other devices or processors physically attached to that bus. By way of contrast, a netstation node is controlled via a network indirectly. Since a node is one among millions on the internetwork, a method must exist that allows a node to detect and reject unauthorized commands.

While a strong means of authentication is desirable for all commands exchanged between a device node and its controlling application, the costs associated with authentication remain high. Unless specialized

acceleration hardware is available at both ends of the connection, the cost of authentication will preclude reaching the needed level of performance.

The performance requirement places a severe restriction on the level of access control that can be provided. Since device-control communication is by nature bursty and can demand high bandwidth, if encryption or digest/authentication algorithms are to be employed their effective throughput should at least match the communications requirements. Recent experience suggests that this will be a difficult goal to meet for digest algorithms such as MD4 and MD5 [8].

A weak compromise position between no access control and authentication can be achieved by restricting access to a device node to hosts that are members of a set of addresses that are defined by a system administrator. This has the advantage of very low overhead and adequately protects a device node from casual attack, but this mechanism is subject to packet spoofing attack.

2.5 Configuration and Resource Discovery

In a conventional architecture the sub-systems that make up a computing system are located physically close to one another. System configuration is largely static and done at boot time. In a netstation architecture close proximity is not required and the nodes that make up a system are logically independent. The configuring sub-systems to form an effective computing system is not a static operation. The desired sub-system services must be defined by application requirement, nodes that provide the services identified, allocated and released when no longer needed. This bears a strong resemblance to issues addressed in the design of distributed operating systems.

Services must be locatable along with the nodes within which they reside. To allow a straightforward and relatively static configuration procedure, one approach is to identify each node with a unique name that can be resolved to an internet address. The address can be used to connect to the node and access its node's property and attribute list. That list would provide information concerning the sub-systems, services present in the node and their status. If a system configuration is static, the names of the component nodes and their sub-systems could be defined in a configuration file that is accessed at boot time.

A more sophisticated configuration capability is created by providing domain servers running on a well-known port that maintain a database of the nodes in their domain, their property and attribute lists. Such a server could also manage the nodes in its domain, controlling access to them. By assigning nodes in a domain by default to the management server in the domain, the server becomes a well-known network rendezvous point for determining which nodes are available, the sub-systems and services that they provide, and for noting to which system or task they are assigned currently.

3. Netstation Node and NVD Model

Traditionally, peripheral devices have been controlled directly, by means of bus transactions. Emerging high-speed networks have reached a level of performance that those device-control functions can now be successfully carried out indirectly, by means of RPCs that are sent to the device via reliable transport-layer connections across a network.

A *Netstation Node* is a node on a network. There are two categories of netstation nodes, a *device node* or a *host node*. A host node is distinguished from a device node by its ability to execute user processes. A device node presents a set of interfaces to the devices that it contains. A device node is not a full-fledged internet host and does not necessarily obey RFC-1122 [9]. The interfaces to the sub-systems of a device node are expressed via ports presented by a reliable transport-layer protocol. Communication to and from a node passes through a *node processor* which executes the transport-layer protocol.

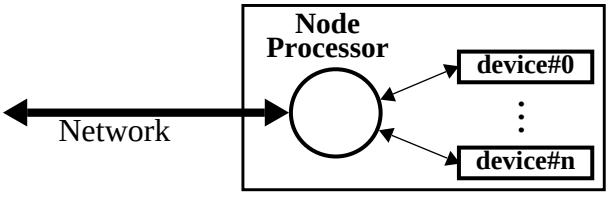

Figure 2. Netstation Device Node

A device node contains a non-empty set of NVDs and the sub-systems that they represent. The NVDs within a node are interdependent to the extent that the transport-layer connections are managed by the node processor that controls the internetwork interface. Each NVD constitutes a named conext within which are executed the RPCs that are directed to it over a connection. Additional physical dependencies between the sub-systems within a node may also exist. For example, a JPEG NVD may require for its operation access to the node's Display NVD frame buffer.

Because of the physical dependency that each NVD has upon the node processor, none of the NVDs within a node can be considered truly independent. Only the node requires an individual internet address. Any question of mobility is assumed to be the province of the network-layer protocol.

The relationship between the interfaces presented and the device sub-systems contained within a device node NVD is not necessarily 1-to-1. Sub-systems may be aggregated by an NVD context to appear externally as a single sub-system. For example, a string of disk drives may appear as a single large drive. A single sub-system may also be presented via multiple NVDs. This technique can be used to present differing execution contexts for a sub-system that correspond to the differing ways that the sub-system is used. For example, a bitmapped display device that has video decompression capability may be presented as either a basic bitmapped display NVD or as a video monitor NVD.

3.1 NVD Access and Context

Access to a specific named NVD is arranged by opening a transport-layer connection to it, which associates that connection with that NVD's execution context and reserves any dependent resources. The transport-layer packets that are received by the node are passed into the associated NVD context and executed. RPC program numbers provide further demultiplexing within that context. The result of RPC execution will generally be a device command. Replies are created and sent back out the same connection.

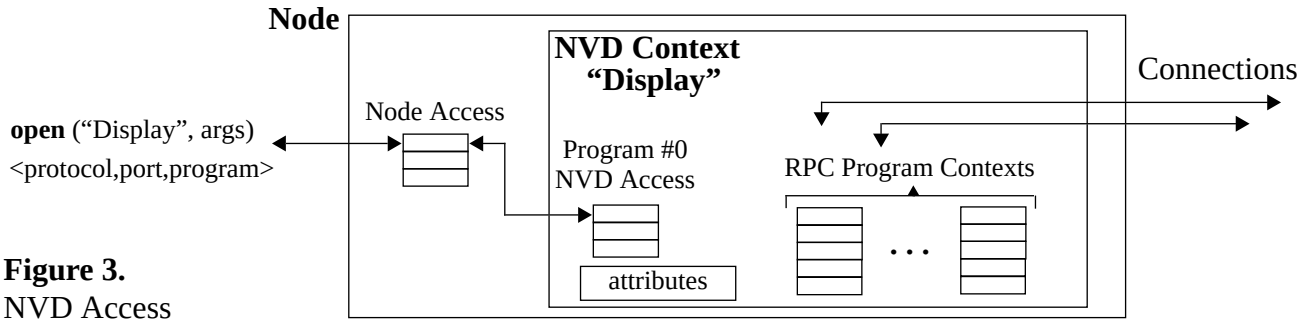

Figure 3.
NVD Access

Figure 5 portrays the relationship between an NVD and its device node. Access to an NVD is achieved by opening a connection to the node over a well-known port and issuing an **open**() RPC to program zero. The **open**() arguments specify the desired NVD by name, in this case "Display". If a Display NVD exists in the node, the **open**() RPC may be passed into the Display NVD context for NVD-related execution. The result of the **open**() is passed back to the caller over its connection.

If access is allowed, the protocol, listening port of a transport-layer connection and an RPC program number will be returned. Connections are opened between the node processor and the NVD's owner. Over that connection, the NVD will expect RPCs for the specified program. Each NVD that is being actively used will have at least one transport-layer connection associated with it.

When the node processor receives a transport-layer payload, it determines to which connection the payload belongs. Once the connection and its associated connection-control block (CCB) has been detemined, the NVD is known. The RPCs contained in the payload are then passed to the correct NVD and program context for execution.

An NVD may implement more than one program context. The intent is that each separate device sub-system within an NVD when opened have a connection uniquely associated with its command stream. The assignment of RPC program numbers to sub-systems may be arbitrary as long as program zero is reserved for NVD-related access and status RPCs. Since RPCs specify both a program number and procedure number in each call, a straightforward way to provide separate program execution contexts is to maintain a separate entry point table for the procedures that are implemented within each program. The assignment of both program and procedure numbers is application specific.

3.1.1 Program Multiplexing onto a Single Connection

Although it is possible to mix RPCs that specify different progams within a single stream, the mixing of different command streams onto a single connection may join streams of distinctly differing quality-of-service or resource reservation characteristics. This is discouraged. The exception to this is NVD access and status RPCs that use program zero. These are mixed into the command stream of the distinguished owner connection to an NVD.

3.2 NVD Naming

An attributes data structure is associated with each NVD. NVDs could be defined and identified within a node by their particular attributes. But a list of attributes may not be unique within a node. This is resolved by requiring that each NVD have a name attribute that is unique within the node. The node processor or a particular NVD within the node may be given a domain name that is associated with the internet address of the node. The Domain Name System (DNS) provides the internetwork name-to-address mapping service [10][11].

In the example of Figure 5, assume that the node is given the domain name: Parrot.isi.edu. The NVDs are assigned names: Audio-Out, JPEG and Display. These NVD sub-system names need not be part of the local DNS database, since they themselves are not nodes on a network. Once the internet address of Parrot.isi.edu is known, access to a specific NVD can be made over a connection to Parrot's access port.

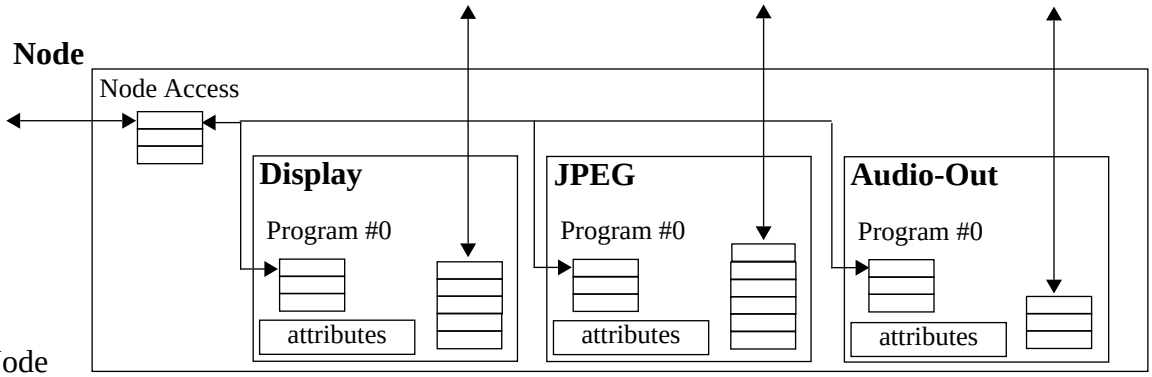

Figure 4.
Complex Node

3.3 Node Clustering, Composition and Internal Dependency

A node may contain multiple sub-systems internally. This may be done to spread the cost of the network interface, because one sub-system requires another sub-system's resources for its operation or because the sub-systems logically cluster together for the purposes of typical application. In Figure 5, the node presents three quasi-independent NVDs: (1) a display sub-system, (2) an audio output sub-system, and (3) a JPEG decompression sub-system whose output is viewed in the display sub-system.

Within this node, the JPEG NVD is physically dependent upon the Display NVD. It makes little sense to allow an application access to the JPEG sub-system, but deny it access to the Display. However, the Audio-Out sub-system is independent. Allowing access to it does not imply use of Display or JPEG sub-system resources within the node. Dependency relationships should be made known in an NVD's attributes explicitly, so that an NVD can be grouped together with the other NVDs upon which it is dependent.

The dependency of one NVD on another suggests a *superior/inferior* relationship. The granting of ownership access to a superior NVD should in general *reserve* all of its inferiors for the same owner. Similarly, releasing ownwership of a superior should imply the release of its inferiors. In the case of a physical dependency, such as that of the JPEG on the Display, the granting of isolated ownership access to the JPEG sub-system makes little or no sense.

NVDs within a node may also be clustered into logically dependent superior/inferior relationships that reflect the needs of a particular application. In Figure 5, a Multimedia-Out NVD is defined that is the superior of the Display, JPEG and Audio-Out NVDs. Once ownership to the Multimedia-Out NVD is granted, the dependency relations would reserve for the owner the three dependent NVDs.

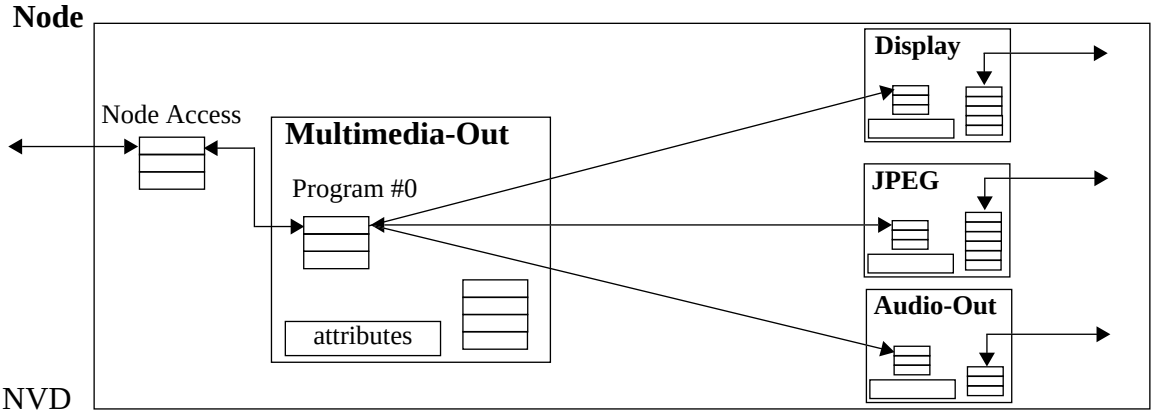

Figure 5.
Composite NVD

3.3.1 Intra-Node Dependency and Execution Context

In Figure 5, access RPCs are shown to pass progressively downwards, from the node program zero context into the program zero context of the NVD named where it is executed. Passing the thread of control from the superior context level to progressively inferior contexts allows access dependencies and similar issues to be dealt within the layer in which they apply.

When a superior reserves an NVD, the program zero context of the reserved NVD becomes inferior to that of the superior. Any subsequent access or status RPCs that are sent to the inferior are passed first to the program zero context of the superior. This arrangement gives a superior explicit control over the granting of access and opening of connections to an inferior.

RPCs sent over the access connection to program zero are interpreted in order to obey their context-related constraints. Other program contexts are typically associated with device sub-system command

streams. In the interest of efficiency, the connections associated with these streams are explicitly bound when opened to a particular NVD and program context. As illustrated by Figure 5, although the Multimedia-Out NVD controls access to its inferior NVDs, the RPCs that control the inferiors arrive over connections that execute directly within them.

3.3.2 Virtual NVDs

The lowest layer NVDs in a device node are expected to interface to a device sub-system. Their RPC program assignments are known internally, are static and are expected to be known externally, by any application that wishes to use the NVD. In some circumstances it may be desirable to define a superior NVD that can interpret the command streams of one or more inferior programs. This is achieved by the superior implementing program contexts that roughly match the contexts of its inferiors, opening the command streams in its own context for the inferior program contexts and trapping the RPCs of the inferiors' programs. Because the result of an **open**() RPC returns to the application the RPC program number to associate with a connection, there need be no conflict with staticly allocated program contexts in the inferiors.

The level of indirection that this provides can allow a superior NVD to adjust the behavior of its inferior programs to meet a more global interface requirement of an application. For example, the specific device command requirements of a particular Display NVD program within a specific node could be hidden underneath a globally understood interface that is provided by the superior NVD.

3.3.3 Extra-Node Dependency

At a higher level of abstraction, nodes may depend upon one another for purposes of executing an application. No provisions for dependencies of that type are made in the NVD model. The application should attempt to allocate the set of NVDs in separate nodes that it needs. If it cannot obtain all the resources that are required for execution, it must release the subset that it had already acquired to avoid potential mutual deadlock with other applications.

3.4 NVD Ownership

When properly initialized, each device node maintains an *access* connection that listens on a well-known <protocol, port, program = 0>. This connection is a rendezvous point for status and access requests for all NVDs within the node. Access and status RPCs that arrive over the access connection contain the name of the NVD for which they are destined.

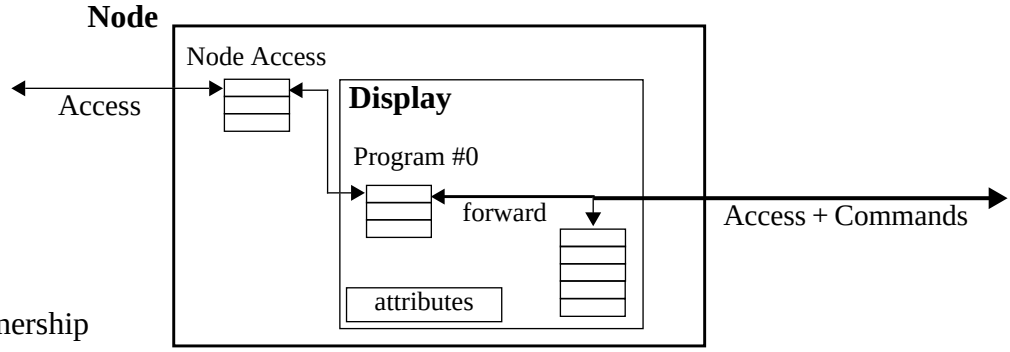

Figure 6.
NVD Access and Ownership

The initial connection to an unopened NVD becomes by default the *owner* connection. Upon receipt of a program zero RPC, the node processor executes the corresponding RPC entry routine in the NVD. One action that can be taken is to forward the RPC over the NVD's owner connection, as depicted in

Figure 6. This allows the owner to interpret any access requests that are sent to its NVD. Another action may be to open a connection to a NVD manager server and forward the RPC to it for resolution.

4. References

[1] Cohen, D., Felderman, R., Finn, G., DeSchon, A..
ATOMIC: A High-Speed Local Communication Architecture
Journal of High-Speed Networks, 1994, No. 1, pp. 1-28.

[2] Boden, N., Cohen, D., et al.
Myrinet: A Gigabit-per-Second Local Area Network
IEEE Micro, Vol. 15, No. 1, February 1995.

[3] Finn, G.
An Integration of Network Communication with Workstation Architecture
ACM Computer Communication Review, Vol. 23, No. 3, October 1991.

[4] RFC 1050
RPC: Remote Procedure Call Protocol Specification Version 2
Sun Microsystems, Inc., June 1988.

[5] RFC 1014
XDR: External Data Representation Standard
Sun Microsystems, Inc., June 1987.

[6] ANSI X3T10-994D, revision 16
Information technology -- SCSI-3 Architecture Model
October 1994, unapproved draft.

[7] Technical Report TR X3-991D
Small Computer Systems Interface -- 3: Generic Packetized protocol (SCSI-GPP)
A draft proposed X3 technical report..

[8] Touch, J.
Report on MD5 Performance
Internet Draft
USC/Information Sciences Institute, Dec. 1994.

[9] Braden, R. (Editor)
RFC 1122
Requirements for Internet Hosts -- Communications Layers
USC/Information Sciences Institute, October 1989.

[10] Mockapetris, P.
RFC 1034
Domain Names - Concepts and Facilities
USC/Information Sciences Institute, Nov. 1987.

[11] Mockapetris, P.
RFC 1035
Domain Names - Implementation and Specification
USC/Information Sciences Institute, Nov. 1987.