NDMP Design Document

Reza Sabdar

Revision 1.6
August, 2007

Abstract

This document specifies the design and architecture of the NDMP server agent for ON Nevada.

Table of Contents

1 Introduction

1.1 Purpose of Document

The document provides the information regarding the design and architecture of the NDMP server on Solaris. The reader will be able to get a high level understanding of technical and developmental issues of NDMP server on Solaris.

1.2 Document's intended audience

The intended audience includes the PSARC review team, the OpenSolaris user community, our Quality Engineering staff and our Technical Writing staff.

1.3 Document References

- NDMP V4 specification
  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/NDMPV4.pdf
- NDMP CLI design documentation
  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/ndmpadm_design.pdf
- NDMP RBAC description
  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/NDMP/PSARC/ndmp_rbac.txt
- NDMP V4 functional specification:
  http://www.ndmp.org/download/sdk_v4/draft-skardal-ndmp4-04.txt
- Sun's NDMP wiki page:
  http://audi-bur.east/wiki/index.php/NDMP
- Man page of ndmpd (1):
  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/NDMP/PSARC/ndmpd.1
- Man page of ndmpdadm (1M):
  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/NDMP/PSARC/ndmpadm.1m

- Man page of ndmpdstat (1M):

  http://audi-bur.east/audi/pb/docs/specs/seastone/idaho/NDMP/PSARC/ndmpstat.1m
- NDMP org website

  http://www.ndmp.org

1.4 Definitions of important terms, acronyms, or abbreviations

Term	Definition
NDMP	Network Data Management Protocol
DMA	Data Management Agent
DAR	Direct Access Recovery
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
API	Application Program Interface
CLI	Command Line Interface
DDS	Dynamic Drive Sharing
EOF	End Of File
EOM	End Of Media
FC	Fibre Channel
GUI	Graphical User Interface
IPC	Inter-Process Communication
LBR	Local Backup/Restore
CRAM-MD5	Challenge Response Authentication Mechanism - Message Digest algorithm 5
NAS	Network Attached Storage
OS	Operating System
SCSI	Small Computer System Interface
SDK	Software Development Kit
ACL	Access Control List
SMF	Service Management Facility
TLM	Tape Library Module

1.5 Summary/Abstract of document contents

This document describes the functional and architectural details and interfaces of the Solaris NDMP server implementation. The purpose of the NDMP server on Solaris is to allow remote, network-based backup applications to control the backup and restore of an NDMP compliant server(s) without installing third party software on Solaris. Specifically, this document describes the Solaris user interface, the architecture, the design, the technical challenges and resolutions and the deliverables for the project.

2 Project Overview

This project integrates NDMP server functionality into Solaris. NDMP will enable Solaris servers to participate in backup and restore operations with NDMP-compliant Data Management Applications (i.e. backup application) and one or more NDMP Servers and backup devices. This project will support NDMP protocol version 2, 3, and 4

2.1 What is NDMP?

The Network Data Management Protocol (NDMP) is an open protocol for enterprise-wide network backup and restore. The purpose of NDMP is to enable backup/restore applications such as EMC/Legato® Enterprise Backup to control the backup and restore operations on NDMP-compliant servers without having to install software to each server. An NDMP Server is one that provides NDMP data, tape, or scsi services. Several different configurations are possible and they are described later.

Once NDMP server software has been integrated, a Solaris system can be backup up using any NDMP-compliant Data Management Application. Today, this list includes the aforementioned EMC® Lagato EBS® (OEM'ed by Sun), Veritas® Netbackup®, BakBone® NetVault®, IBM/Tivoli® storage manager, and numerous others.

The primary advantage of NDMP is that it provides standardizes interfaces that can be incorporated into a vendor's OS for use by commercially available 3rd Party backup applications. Another advantage of NDMP is that is separates data transfer and control operations thus allowing a backup operation to be controlled over the network while the data will be transmitted only if necessary to reach the backup subsystem. On systems with a locally attached tape device the data does not cross the wire even though the backup application may be running elsewhere on the network.

In addition to backup and restore operations, NDMP provides a set of commands to control tape operations as well as SCSI service that allows low-level SCSI commands to be passed to remotely attached such as SCSI or Fiber Channel media changers.

2.2 User experience

NDMP will be delivered as two packages in the Solaris distribution: SUNWndmpu and SUNWndmpr. The NDMP service can be enabled using the standard SMF interfaces svcs(1M) and svcadm(1M). Once running, the NDMP daemon can be administered using a single command: ndmpadm (1M). ndmpadm will allow a user to query the availability and status of backup devices on the system, get and set NDMP variables, and display status and manage active NDMP sessions.

Beyond this, NDMP operation is completely controlled by a NDMP-compliant Data Management Application.

2.3 Definitions

NDMP Host/Server: The host computer system that executes the NDMP Server application. Data is backed up from the NDMP host to either a local tape drive or to a backup device on a remote NDMP host. The server application typically runs as a daemon service.

Data Management Application(DMA): The Data Management Application is the backup/restore application that controls the NDMP session. In NDMP there is a client-server relationship in which the DMA is the session server; the NDMP services are the clients. In NDMP versions 1, 2, and 3 the term "NDMP client" was used instead of "DMA."

Control Connection: A bi-directional TCP/IP connection that carries XDR-encoded NDMP messages between the DMA and the NDMP Server. NDMP protocol has a veriaty of commands classified in different "interfaces" for the backup services it provides. The commands and their parameters change between the versions.

Data Connection: The connection between the two NDMP Servers that carry the data stream. The data connection in NDMP is either an NDMP interprocess communication mechanism (for local operations) or a TCP/IP connection (remote backup).

Data Service: The NDMP service that transfers data between disk and the Data Connection. Data services provide an abstracted interface to the filesystem or disk of the NDMP server.

Tape and SCSI Service: A NDMP Service that passes low-level SCSI commands to a SCSI device, or manipulating and access to the tape device, typically used by the DMA to manipulate a SCSI or fiber channel attached media changer or a tape device.

DAR or Direct Access Recovery: An optional capability of NDMP Data and Tape Services whereby only relevant portions of tape media are accessed during Restore Operations.

NDMP environment variables: These are a set of name/value pairs which are passed between the NDMP server and DMA on certain occasions. The available variables are passed within NDMP messages and only some messages can include environment variables. A few example of these variables are DIRECT, HIST and FILES.

NDMP configuration parameters: Not to be mistaken with environment variables, these are the parameters set by the administrator to perform the backup. The ndmpadm(1M) tools is used to set/view these variables such as ndmp_version.

Mover: An aspect of the Tape Service that transfers data between the tape device and the Data connection.

2.4 NDMP Configurations

2.4.1 Local backup / restore

This is the simplest configuration where the data and the tape device both reside on the same server. The DMA (Backup Application) uses a TCP/IP connection to the server to manage the backup of the data to the tape device. The backup data remains entirely local to the server.

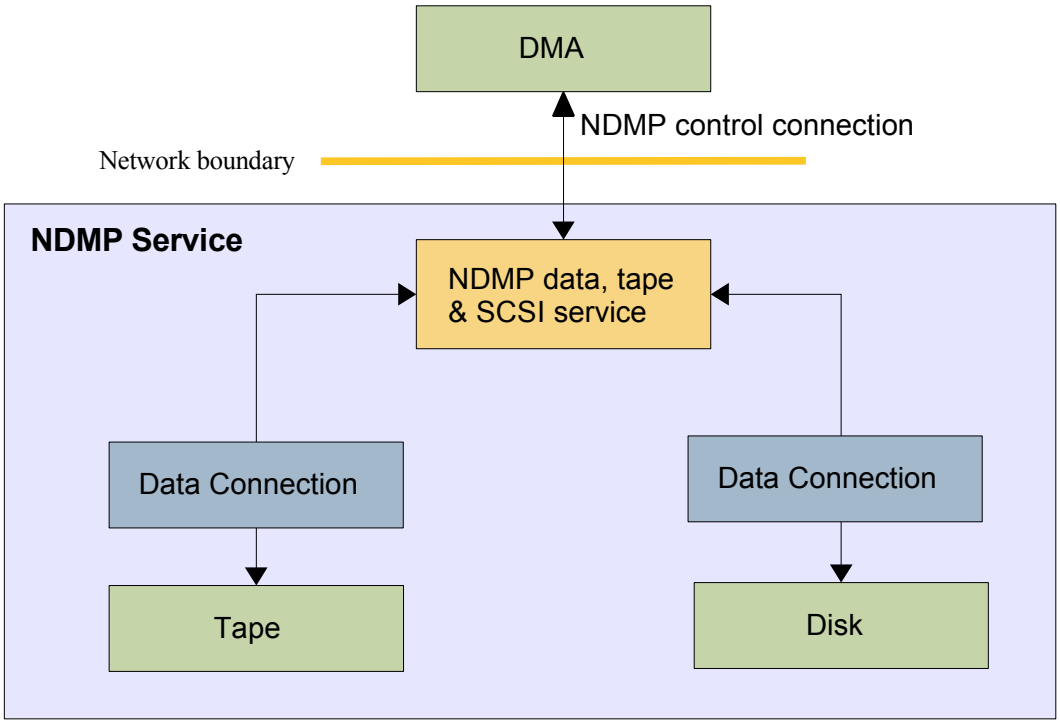

Figure1- Simple Backup / Restore Configuration

Multiple backup sessions can be performed and managed concurrently on a single (or multiple) servers. In this case each session has it's own locally-attached tape backup device.

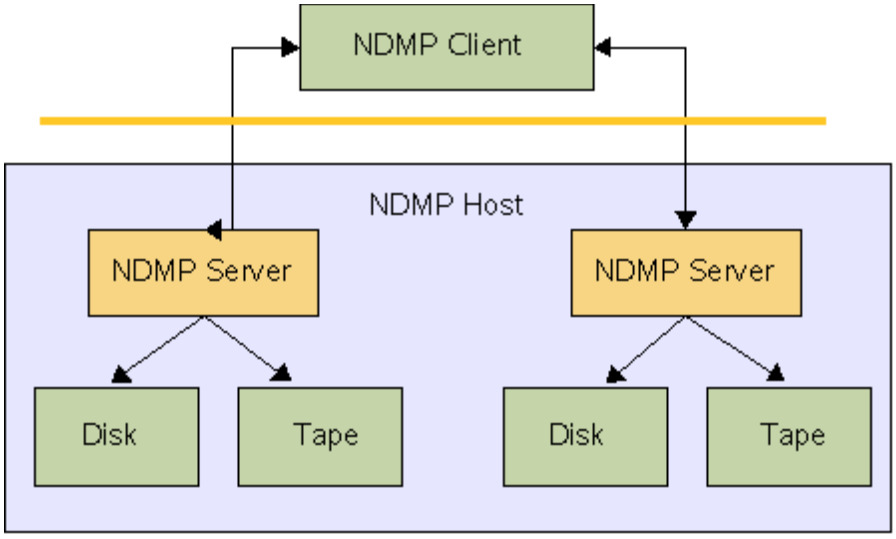

Figure 2 - Multiple NDMP sessions on a single NDMP Host

2.4.2 Remote backup

In this configuration, is also known as three-way backup, the data and the tape device reside on different NDMP servers. The DMA uses TCP/IP connections to both servers to coordinate and manage the backup of the data to the tape device. A separate NDMP TCP/IP data connection between the two servers is used to transfer the data.

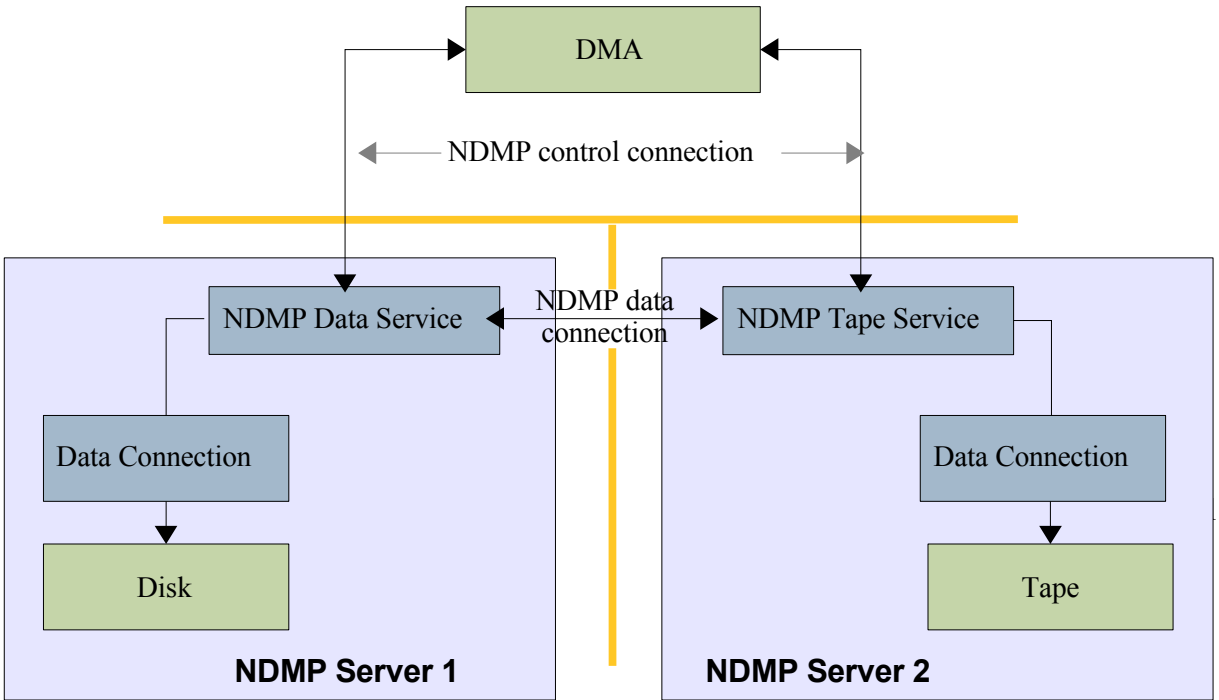

Figure 3 - Standare three-way backup configuration

2.4.3 Incremental level backup

NDMP supports both full and incremental backup. There are two different incremental backup methods: *leveled* and *token-based* incremental backup. The most common incremental backup method supported by 3rd party DMAs is “Level” backup method.

A backup *level* is determined using the value of an environment variable when backup starts. The environment variable is named LEVEL with valid values of ‘0’ to ‘9’. A level of ‘0’ is full backup and subsequent backup level 'n' in the range ‘1’ to ‘9’ is relative to 'n-1'. For example, consider the following events in the system:

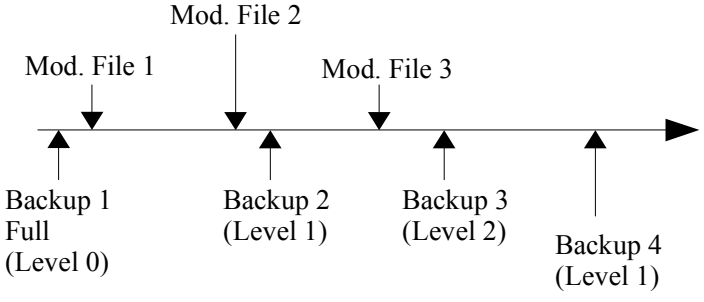

Figure 4 - Incremental backup example

The following table shows what each backup contains and why:

Backup Name	Backup Contents	Reason
Backup1 (L0)	All files	Full backup.
Backup2 (L1)	File1, File2	Both files were modified since Backup1
Backup3 (L2)	Files3	Only File3 was modified after Backup2
Backup4 (L1)	File1, File2 and File3	All the 3 files were modified after Backup1.

The implementation of incremental backup depends on the underlying backup technology being used. The current implementation of NDMP proposed by this ARC case supports both dump- and tar-type backup. In both cases, the incremental backup scheme is similar to the UNIX 'dump' command: information about the files being backed up is stored in a metadata file maintained on the target server. (A future implementation of NDMP will support a ZFS backup type which will have it's own metadata format. NDMP has been implemented with an interface, described in section x, that will be invoked to write backup metadata appropriately for the backup type). The metadata for dump- and tar-type backups is stored in a text file named dumpdates. The dumpdate's text file format is similar to that used by the dump backup utility. Each line consists of three fix-width fields: volume or file system name, backup level, and the date of the backup. Lines beginning with a '#' character will be considered as comment lines and will be ignored. For example:

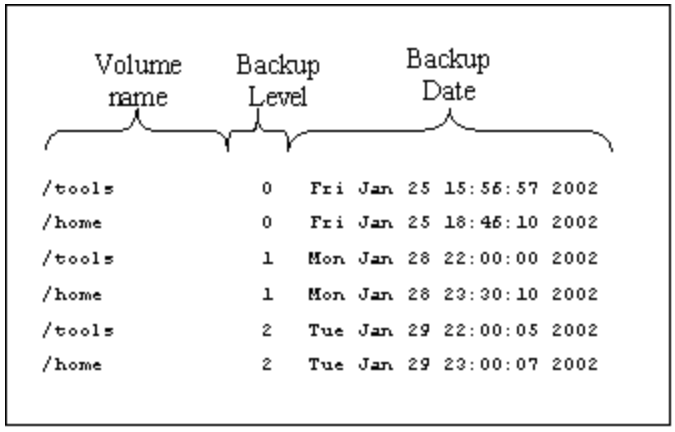

Figure 5 - Dumpdates File Example

When a backup is made, the date and time of the backup of the previous level is obtained from the file. All files in the backup path modified after that date and time are backed up. After the backup is complete, the dump date file is updated if the UPDATE environment variable is set to ‘y’. An environment variable is a variable that is used by NDMP peers to communicate with each other. These variables are discussed in detail in Section 7.1.2. If the entry for the current backup level of the current file system exists in the ‘dumpdates’ file, that entry is updated in the file, otherwise, a new entry is appended to the file.

This method of backup does not let multiple backup sets represent independent full and incremental backup schedules on the same file systems, which may cause limitation for NDMP DMAs. To address this problem, another NDMP environment variable can be used to show the dump name. The variable name is DMP_NAME and can hold the name of the backup. The same name can be used as the dumpdates file of that backup set. If DMP_NAME environment variable is not set the default ‘dumpdates’ file will be used, otherwise, a file with the same name as the value of this environment variable will be used to extract the correct backup date.

The dumpdates file is only used when incremental dump backup is taken. There are other incremental backups possible depending on the type of filesystem being used such as ZFS incremental backup. In that case, dumpdates file can no longer be used and other ZFS-specific methods will be used for finding modified files rather than the file modification time and date. More details on toggling between ZFS-incremental and dump-incremental methods are presented in 2.5.1.

If a dump backup is being performed and the target filesystem is ZFS, however the dumpdates file will not be created and instead, the ZFS user properties will be used. These user defined ZFS properties will present the information necessary to run the next level incremental backup and as they are kept along with the filesystem, they can be retrieved anytime they are needed. Below is the list of variables that present dumpdates data in ZFS filesystem user properties:

ZFS user property name	Comments
dumpdates:L0, dumpdates:full	The date and time of the last level 0 (full) backup. Not defined in case of no backup.
dumpdates:L1..dumpdates:L9	The date and time of the last level 1..9 incremental backup. Not defined in case of no backup.

All the references to 'dumpdates' file throughout this document would be interpreted to 'dumpdates:' module in case ZFS filesystem is being used.

2.4.4 Token-based incremental backup

Token-based backup is the other form of incremental backup that is implemented by some DMA and NAS vendors. In a level-based incremental backup, a level is sent to the server. The server finds the base date for the backup by looking up in the ‘dumpdates’ file using the file system/volume name and the level specified by the DMA. In token-based incremental backup the data server does not keep track of

backup dates and levels, instead a date is sent to the server by the DMA. The date sent to the server by the DMA is similar to the date extracted from dumpdates file in the level method.

The token is a 64-bit integer as shown below. The upper 32 bits represent a zero-based count of the number of sequential incremental backups performed. This counter will be used to limit the number of sequential incremental backups done. The lower 32 bits represent the backup timestamp. It is the number of seconds passed since the UNIX epoch (January 1st, 1970). All the files modified after that time will be backed up.

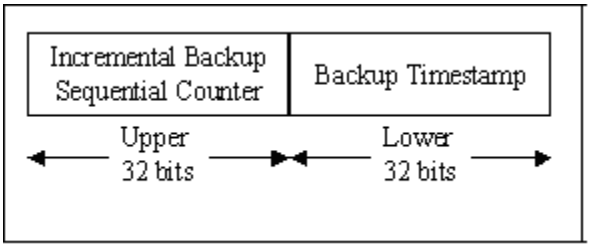

Figure 6 – Token based backup token layout

The token is passed to the server via an environment variable called BASE_DATE. The server validates the counter. If the limit is reached the backup is not performed. Otherwise, the date specified in the environment variable will be used instead of looking up the 'dumpdates' file. The limitation on the number of sequential backup will be specified by "*max.inc.backups"* parameter. The default value for "*max.inc.backups"* will be 7, as the number of days in a week. The value of BASE_DATE is **never** touched by the DMA. DMA either obtains it from the DUMP_DATE environment variable of the data server or sets it to zero for full backup.

Following a successful backup, the data server returns an environment variable named DUMP_DATE to the DMA if the BASE_DATE was set. The counter field of BASE_DATE is incremented and saved in the counter field of DUMP_DATE. The date/time field of it shows the backup startup time of the current backup.

If the BASE_DATE is set, the data server does not read or update the 'dumpdates' file, means that the UPDATE environment variable is ignored. The DMP_NAME can still be valid and can be used to log backups. If the BASE_DATE is zero, the corresponding 'dumpdates' file of the backup will be truncated and then the full backup date will be logged.

If the BASE_DATE is not set by the DMA, the data server reads the 'dumpdates' file and does not return the DUMP_DATE environment variable.

Supporting token-based incremental backup was added to V3 as an extra feature.

2.4.5 Selective Backup

NDMP has environment variable named FILES that can be used when backup type is set to 'tar'. This environment variable shows a list of file/directory names to be backed up. If the DMA supports user-defined environment variables, users can set the value of this environment variable to the path or names

to be backed up.

The value of this environment variable should be a space-separated list of patterns, file or directory names. Paths are not accepted as a value of this variable. If names contain space characters, those space characters should be escaped by a '\' character. Only '*' and '?' characters are allowed as wildcards in the FILES environment variable. For example, a user might be interested in backing up source files such as *.c, *.h and *.s files, in which case the user can set this environment variable as:

```
FILES=*.c *.h *.s
```

Some DMAs allow multiple environment variables with the same name. For example, SyncSort Backup Express® allows multiple instances of FILES environment variable with different values. If this environment variable is not set, all the files and directories in the backup path will be backed up.

2.4.6 Excluding Files/Directories

NDMP can have an exclusion list that contains files or paths that are exempt from backup. Some DMAs have the capability of user-defined NDMP environment variables for passing vendor specific information to the NDMP servers. There is a NDMP environment variable for that reason called EXCLUDE which is a comma-separated list of patterns, files or directory names. The files/directories of the file system that match the entries of the EXCLUDE environment variable will be excluded during backup.

The names and patterns as used with this environment variable must be file names and cannot contain a path. If there is ',' in the file names, it must be escaped by '\' character. Only '*' and '?' characters are allowed as wildcards in the EXCLUDE environment variable. The '*' character matches any number of any character. The '?' character matches one occurrence of any character.

For example, users can exclude the "etc' directory of the volume and the core files by simply setting the EXCLUDE environment variable to "etc,core". Note: there should be no spaces between the entries, otherwise, it will be considered as a part of file names and will not result in the desired filtering.

2.4.7 Local/Remote restore

The restore configurations are the same as backup except that the data flow is in the reverse direction. So the pictures are similar to those shown above.

2.4.8 Direct access restore

Direct access restore or DAR method was added to NDMP protocol in version 3, which makes it possible to restore a small set of a huge backup images in a relatively short time. DAR is a method of locating objects in the backup stream by directly seeking to their position, instead of sequential search. This

reduces recovery time of a small number of files in many tapes. This feature is supported for both v3 and v4 versions in the NDMP server.

For DAR support, the data server needs to keep track of object locations in the backup stream. When a file/directory is being added to the backup stream, its start location is saved by the data server and sent as part of the file history to the DMA. The DMA is responsible for saving this information and returning it to the data server at recovery time.

The location of the object is defined to be the offset in bytes from the beginning of the backup stream. This is opaque to the DMA and it does not do any processing on this information.

Restoring Mid-level Directories in DAR

In DAR, if the object to be restored is a single file and the mid-level directories of its path don't exist at recovery time, the data server creates those directories. So those directories are not restored and their permission and other information might be different from the directories at backup time. This is because the data server does not know the location of each mid-level directory if only the DMA requests the file to be restored, so it cannot restore those directories. There are three solutions for this problem:

- Setting the permission of those directories manually.
- The DMA should include those directories in the recovery list.
- Full recovery.

Recursive vs. Non-recursive DAR

DAR can be done on single file or a directory. For single file the workflow is straightforward. The data server asks the mover to read a part of the backup stream and send it to the server. The server retrieves the information and creates the object based on the information found in the backup stream.

DAR can be done on directories recursively too. In this case, the data server should instruct the mover to seek to the position of the current entry of the recovery list. Read the header and restore it. If the current entry is a directory, the data server should then continue reading the next entry of the backup stream, examine it and see if it is an entry beneath the requested directory. If it is, it should be restored. Otherwise, it should stop recursive DAR and continue with the next selection in the recovery selection list.

DAR NDMP Environment Variables

DAR is used for recovery when the DIRECT environment variable is set to 'y'. The HIST environment variable also must be set to 'y' to force the data server to generate the file history information.

User should be able to disable DAR. Setting the *dar* mode to 'no' using ndmpadm tools can do this. If this parameter is set, it overrides the value of DIRECT environment variable of NDMP. The default value for DAR is 'no'.

2.4.9 D2D copy

Disk-to-disk copy is an NDMP feature that was introduced in Version 3. Using this feature, files from one Solaris server can be copied from one disk to another disk where the destination disk can be local or reside on a different server on the network.

In disk-to-disk copies, two NDMP data servers typically are engaged. One of them acts as the source and the other acts as the destination. The source data server starts backing up the part of the file system that is specified. The source data server has no idea of the destination of the data. It backs up the files as if it does normal disk-to-tape backup, and sends them to the other NDMP server. The destination NDMP data server acts as if it is restoring data from tape and restores the files and directories to the location specified.

Since only the server knows the format of the backup stream the other end of disk-to-disk copy **must** be a similar Solaris or other compatible platform.

The two NDMP data servers should be connected to each other by the messages that the DMA sends to them. If the destination of the disk-to-disk copy is the same host as the source of the operation, a new connection type, named IPC in V3, is used as the data connection type between the two servers. The implementation of the IPC based data connection is vendor specific. Since the data are copied at file system level, all the file information will be moved from the source to the destination.

It should be noticed that this is not a disk-to-disk backup. It is a disk-to-disk copy and means there is no restore phase associated with this operation. The files on the destination will be the same as the files in the source filesystem.

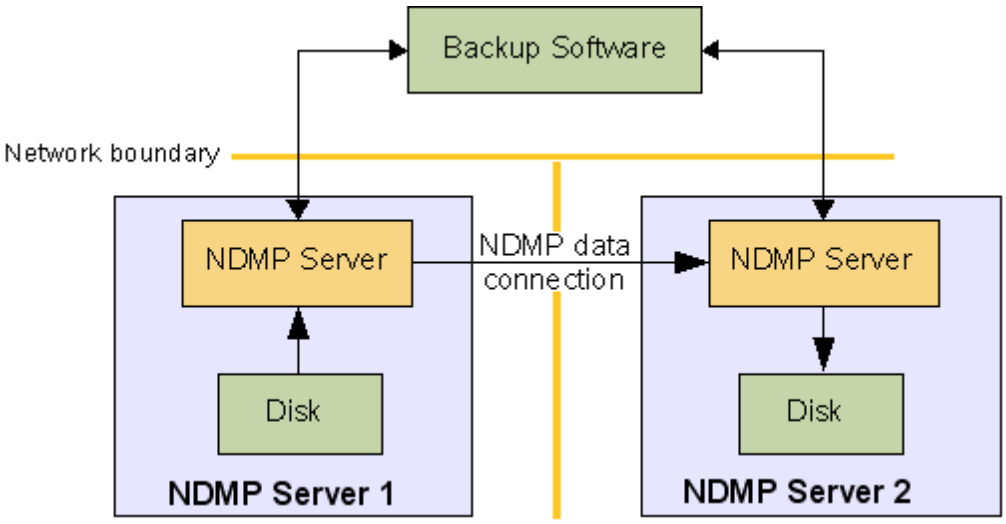

Figure 7 - Disk-to-disk copy

2.4.10 T2T copy

Tape-to-tape copy is another NDMP feature, which supports tape duplication. Two NDMP tape servers are engaged in each tape-to-tape copy operation.

Since the format of the data stream is not important in tape-to-tape copy, the two NDMP servers can be any NDMP implementation (not necessarily from Sun).

One NDMP server acts as the source tape server and the other as the destination tape server. The DMA should connect the two tape servers to each other. When the operation begins the source tape server will read the tape as if the data is going to be restored and sends it to the other tape server. The destination of the data is transparent to the source. The destination tape server reads data as if a backup stream is being written to the tape. The source of the data is transparent to the destination tape server.

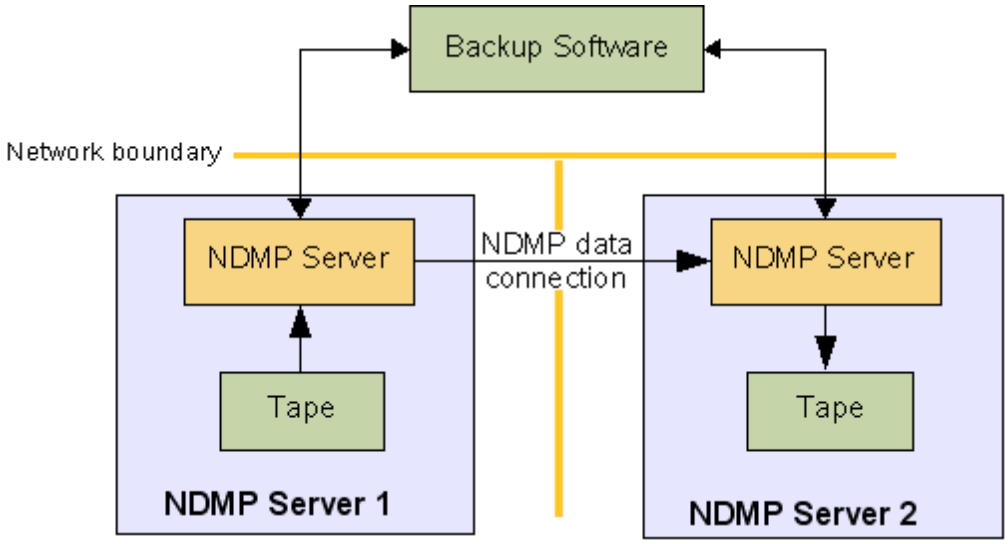

Figure 8 – tape-to-tape copy

2.5 Architectural Design

A simple layout of the architectural design of the NDMP server is shown in the following diagram. The NDMP sever is running on top of the TLM unit which consists of reader/writer threads and a few buffers to pass data to/from the tape or disk. The upper layer is comprised of components based on the NDMP protocol interfaces namely, data, mover, config, connect, SCSI/tape and file history. Each component is

responsible for handling the requests of the corresponding interface properly. The requests from the client are sent through RPC calls which are not shown in this diagram. Other than clients NDMP control connection there are two data connections for data sockets and mover sockets. These sockets are used for performing remote backup/restore or disk to disk and tape to tape copies between the two servers.

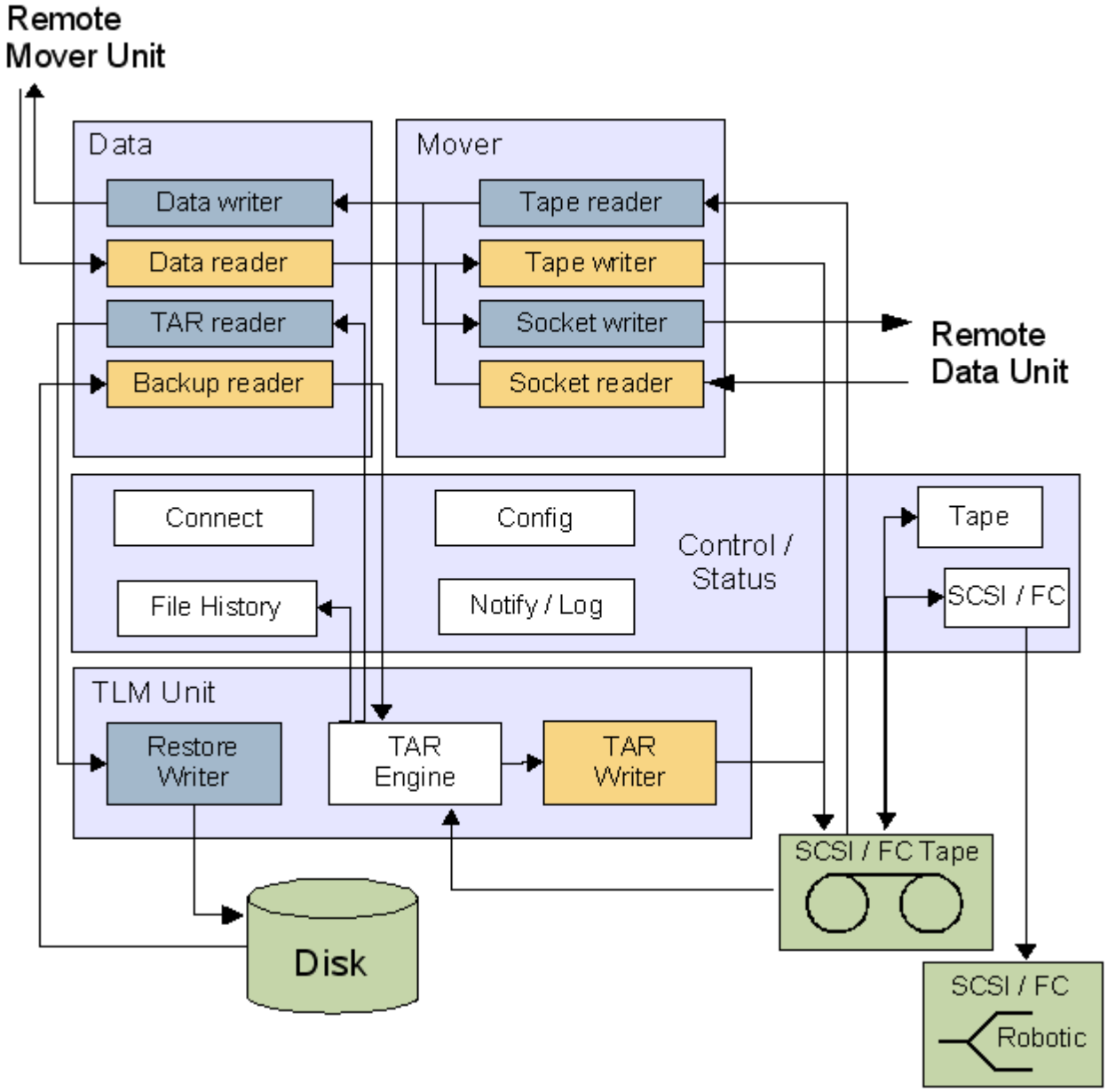

Figure 9 – NDMP architectural layout

2.5.1 TLM unit

TLM or Tape Library Management unit is the component handling the archive image creation, which in

our case is tar(1) format, while taking backups and extracting it during restore. It consists of a tar engine, which is based on Solaris tar implementation, and a couple of reader and writer threads for performing the IO with the tape and simple buffer management for passing data between the threads. TLM also provides other functions for traversing the directory trees, creating bitmap files for incremental backups and some primary functions for SCSI pages read/write operations.

Tar engine

Tar engine provides the archiving support for backup data on tape which is exactly as Solaris' extended USTAR tape format. So if we could locate the tar image on the tape, it would be directly extractable by the tar(1) command. The only problem for this process is that the DMA's usually put some other header and trailer data blocks before and after the tar image that makes it hard to locate. However, NDMP tar image always starts with a magic string “NDMPUTF8MAGIC” which could be used to locate the data by looking for the block that contains this magic string at the first record.

An an example if the position of the data on tape is at the <n>'s filemark we could run the following to extract myhomedir directory to the /vol1/restore path:

```
# mtio -f /dev/rmt/1 rewind
# mtio -f /dev/rmt/1 fsf n; tar xvf /dev/rmt/1 myhomedir -C /vol1/restore/
```

For more details on the tar format and the on-tape image format refer to the tar(1) command man page.

The interface between the tar engine and NDMP is designed with some level of factoring which makes it possible to remove the tar engine and replace it with another customized data archiving method such as cpio(1), ufsdump(1M) and zfs(1M) send. For this reason an *ops vector* is defined which will be the interface between NDMP and TLM unit. The only ops vector currently used is tar and it will be expanded to ZFS in next phase of the project:

Operation	Comments
int (*tm_putfile)()	Puts a single file in the backup stream
int (*tm_putdir)()	Puts a directory tree in the backup stream
int (*tm_putvol)()	Puts the snapshot of a volume or filesystem in the backup stream
int (*tm_getfile)()	Extracts a single file from the backup stream
int (*tm_getdir)()	Extracts a directory tree from the backup stream
int (*tm_getvol)()	Restores a snapshot of a volume or filesystem from the backup stream
int (*tm_putincm)()	Store the incremental backup metadata
int (*tm_getincm)()	Retrieve the incremental backup metadata

As it is shown above, the interface provides both object level and volume level interface to the backup stream. For cpio(1), tar(1) and ufsdump(1M) the file/dir level will be used while for zfs(1), the volume or

filesystem/snapshot level archiving will be required. The last two operations provide the interface slot for changing the underlying data archiving methods for handling incremental backups. As incremental backups are handled differently in traditional dump format and ZFS, this interface provides a flexible slot to replace the 'dumpdates' file with any other metadata pool that can be used for performing incremental backups.

As tar format is currently the only archiving method implemented in this project, we will refer to 'archiving method' as 'tar method' or 'tar engine' throughout this document.

The interface is provided through a structure named `tar_ops`. This interface is among the other 'project private' interfaces between NDMP and TLM unit which is discussed more in detail in chapter 7.

Tar writer thread

This thread is responsible for writing the tar image onto the tape and works closely with other NDMP threads to do that. Refer to the main NDMP architectural diagram at the beginning of this section. The threads shown in orange (light gray) show the backup flow and the blue (dark gray) ones show restore. As shown, the tar writer thread in TLM unit works with ndmpBackupReader thread to perform the backup. The synchronization between these two threads is done using a buffer. The threads involved in performing a local backup is shown below:

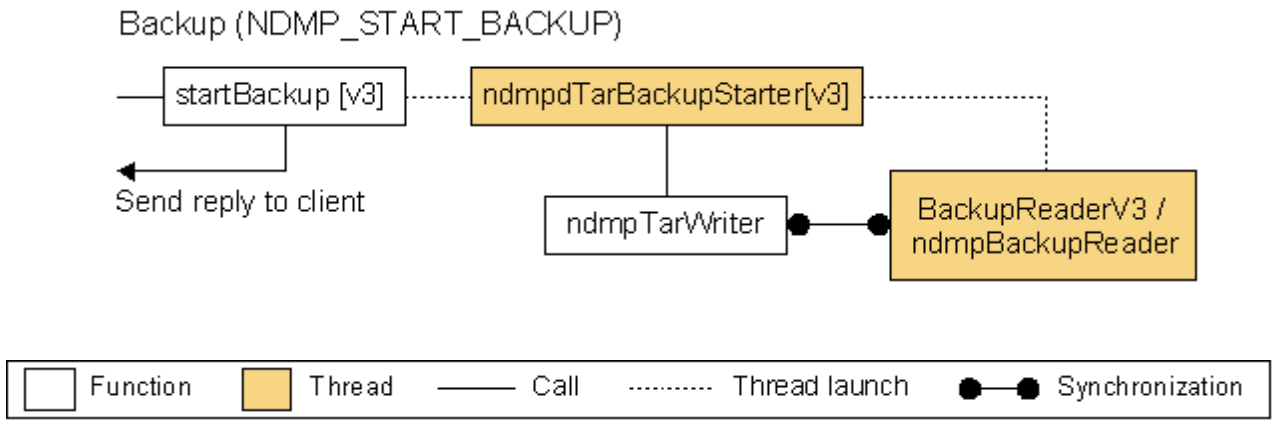

Figure 10 – Local backup threads

The startBackup handler function replies to the request and creates a thread for reading files and writing tar output. Synchronization is done through the simple IPC buffers. The reader thread does the file tree traversing and the writer uses the tar engine to create the output stream.

Restore writer threads

This thread is used when a tape is locally restored. During this process as shown in the architectural diagram, the restore writer thread works with ndmpTarReader thread to restore the tar image read from the media. The threads involved in performing a local restore are shown below:

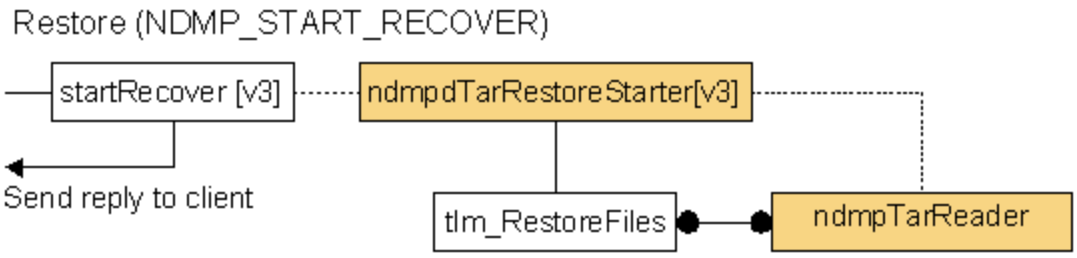

Figure 11 – Local restore threads

The startRecover handler function replies to the request and creates a thread for reading the tape and writing the data to disk. The UNTAR process is done in tlm_RestoreFiles function.

IPC buffers

IPC buffer or TLM_BUFFERS is a set of buffers with counts and locks used between the reader and writer threads for synchronization and as an IPC between these threads. Tape buffers is a rotating set of ten 32KB buffers , which each buffer has size, current location, status and locks information. This buffer is designed to be suitable to keep track of data flow to or from the tape.

2.5.2 NDMP unit

NDMP interface message handlers

NDMP is a RPC-based protocol which consists of different messages. All these messages based on the type and purpose they serve are categorized into the following subsets which is called NDMP interfaces, these interfaces are shown as solid blocks in the architectural diagram above:

- Data interface: for starting backup/restore, provides access to the data on disk
- Mover interface: for accessing the mover to transfer data to/from tape stream.
- Tape interface: for all tape management commands
- SCSI interface: for all SCSI (tape and jukebox) management commands
- Connect interface: for starting/stopping a connection and authentication

- Config interface: for getting the system and server filesystem, tape, SCSI and server information.
- Notify interface: for notifying the client of change of status
- File history interface: for creating the index and catalog for the client.
- Log interface: for sending the logs

All these nine interfaces are implemented in separate components which consist of the handler functions which implement the service.

Data and mover state machines

Other than the handlers, there are two state machines implemented inside the NDMP module which represent the states and the transitions and the sanity checking between these transitions. Both state machines have idle, listen, connected, active and halt states. The listen and connected states are used both for local and remote backups. For local backups the connection is considered to be a local IPC connection where in a remote backup this would be a TCPIP connection. More details of these state machines can be found in the NDMP v4 functional specification.

General worker threads

The basic threads model is depicted below:

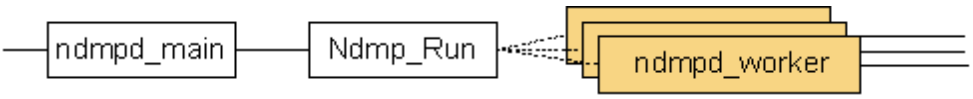

Figure 13 – General worker threads

Ndmp_Run creates a worker thread for each TCP accept call. Each worker thread dispatches the NDMP RPCs on that TCP connection and eventually calls the handler functions. Each handler could be any NDMP message handler including the handlers for NDMP_START_BACKUP and NDMP_START_RECOVER.

Remote backup/restore threads

There are several threads in the Data and Mover components of the NDMP server that handle the remote backup and restore. Data and Mover components each provide a peer for remote incoming/outgoing connections for remote or 3-way backup and restore. As shown in the NDMP architectural diagram, Data component provides the peer filesystem and disk access while the Mover provides the peer tape and media access. In the block diagram, the threads in orange (light gray) indicate the threads handling the backup and the blue (dark gray) ones are threads involving in restore. As can be seen in the diagram each Data thread works with a corresponding Mover thread to perform each of remote backup and restore.

Data reader/writer threads

These threads are part of the data component and serve the read/writes from the remote or local peer for passing the data. The DATA_LISTEN and DATA_CONNECT messages provide the data connection with the peer data server and these threads on each peer would be responsible for reading/writing the data on the socket.

Mover reader/writer threads

Similarly these threads perform the read/writes on the socket and they are as part of the mover component. Socket reader will read the remote data from the peer data server and pass it down to the mover to be written to the tape using the tape writer thread. Similarly the tape reader thread streams the tape data through the mover and pass it on the network socket (or local IPC connection) using the socket writer thread. The MOVER_LISTEN and MOVER_CONNECT commands are used for preparing the initial connection.

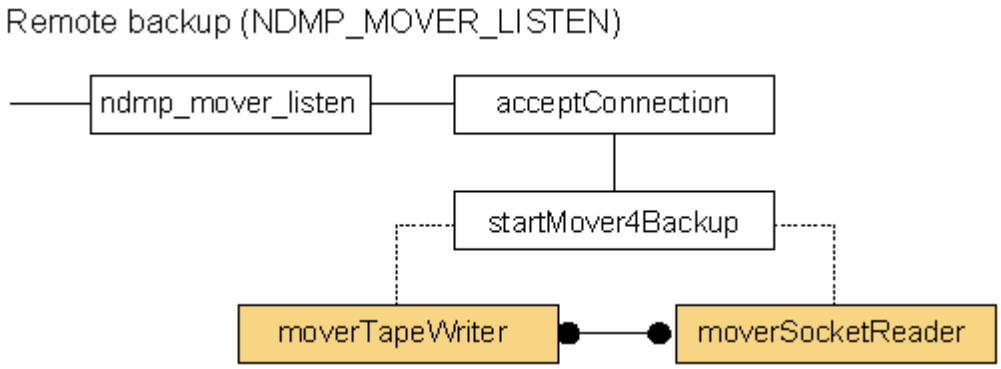

Figure 14 – Remote backup threads

For remote backup, two threads are created: one for reading the tar stream from the network socket and the other for writing the image to the tape. The threads sync through the IPC buffers.

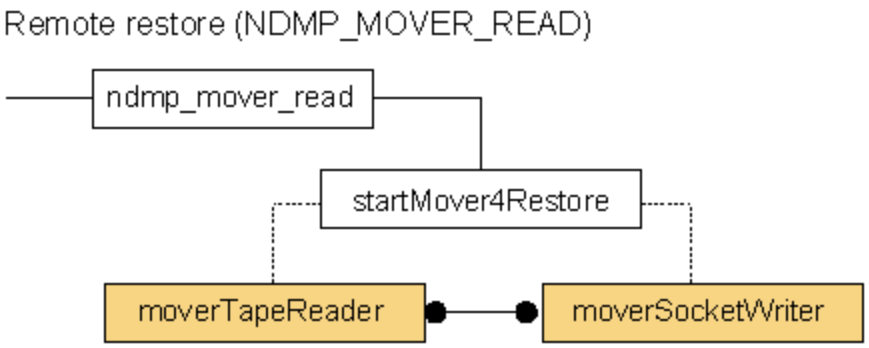

Figure 15 – Remote restore threads

For remote restore, two threads are created: one for reading the tar image from the tape and another for writing the data to the network socket. The threads sync thru the IPC buffers.

3 Deliverables

As a result of this project the NDMP source tree would be delivered into ONNV Nevada gate. The source will go under usr/closed as currently the code will not be part of OpenSolaris even though the effort for open sourcing NDMP is already started and in case of positive results the source tree can be moved to usr/src.

3.1 User Space commands

The NDMP service in Solaris will be provided by a user-space program. The ndmpd shell command would bring up the daemon as the server which accepts remote or local NDMP request for backup on the hosting Solaris server or even a remote Solaris or any other server supporting NDMP protocol. The man page of ndmpd (1) is provided in the reference section.

Although NDMP needs to distinguish some of the features of the underlying filesystem such as snapshot capability, it can still run on a filesystem without any of these features. For example a backup of a ZFS volume would create a snapshot and take a backup from the snapshot for consistency reasons, but in case of UFS, which doesn't have a snapshot capability, the backup will be taken from the live filesystem.

NDMP command also saves the setup and configurations like the configuration parameters using SMF properties and property groups, but the command can still work solely through the command line parameters. So basically there is no dependency or presumption necessary to run ndmpd on Solaris.

This command is integrated as part of the SMF service, which will run at the time of the machine startup and can be managed by svcs(1).

3.1.1 Ndmpd

NDMP command will be delivered as two packages SUNWndmpu and SUNWndmpr which consists of the following:

```
SUNWndmpr:
/usr/lib/ndmp/ndmpd
/usr/sbin/ndmpadm
/usr/sbin/ndmpstat


SUNWndmpu:
/lib/svc/method/svc-ndmp
```

```
/var/svc/mainfest/system/ndmp.xml
```

Ndmpd is the server daemon that runs and listens on a specified port for incoming NDMP connections from the DMA clients. After installing the SUNWndmpr and SUNWndmpu packages, ndmpd service for NDMP V4 can be run by:

```
# ndmpd -q -v4
#
```

Even though the service can be started as command-line, SMF should normally be used for enabling and/or restarting the service:

```
# svcadm enable ndmpd


# svcs | grep ndmp
online         15:45:32 svc:/system/ndmpd:default


# svcs -l ndmpd
fmri        svc:/system/ndmpd:default
name        NDMP Service daemon
enabled     true
state       online
next_state  none
state_time  Wed Apr 04 14:39:19 2007
logfile     /var/svc/log/system-ndmpd:default.log
extract_id  80
dependency  require_all/error svc:/milestone/sysconfig (online)
```

Refer to the ndmpd (1) man page the reference section for more details and examples of using ndmpd.

3.1.2 Ndmpadm

This is a command line program that can set/change the parameters for NDMP service. Meanwhile it can also obtain some useful information about the service which currently is running. This is provided as part of SUNWndmpr package. This command can be used to get the status of current backup:

```
# ndmpadm stat all
        Session Id:     641
        Protocol version:       4
```

```
        Authenticated:          Yes
        EOF:                    No
        Client address:         192.168.80.218:4824
        1 active session.
```

Or to get the value of configuration parameters:

```
# ndmpadm get all
        debug_path=/var/ndmp
        version=4
        dar=no
```

The syntax of this command along with examples are in the ndmpadm (1M) man page which can be found in the reference section.

3.1.3 Ndmpstat

This command line is similar to vmstat(1M) and reports statistics about the progress of NDMP backup or restore in terms of the count of operations and the amount of bytes transferred. It reports the number of threads involved in the current backup operation along with the number of disk/tape IO operations and the number of bytes transferred during each interval. Without options, ndmpstat(1M) displays a one line summary of the backup activities since the system was booted. Below is an example output of the command:

```
# ndmpstat 5
wthr     file      disk      tape        bytes      perf       prcnt
r w b r rd  wr    rd  wr   rd    wr     rd    wr   bk  rs  dsk  tpe  otr
1 0 3 6 50   9   1250  0  32544 4455   42335 3234   5  4   20   40   40
1 0 0 1  1   0    128  0    0   128     64    64    1  0    0   80   20
1 0 0 1  2   0    128  0    0     0     64     0    1  0   80    0   20
1 0 0 1  1   0    128  0    0     0     64     0    1  0   80    0   20
1 0 0 1  3   0    128  0    0     0     64     0    0  0   80    0   20
1 0 0 1  1   0    128  0    0   128     64    64    1  0    0   80   20
^C
```

The syntax of this command along with details of the output and examples are in the ndmpstat(1M) man page which can be found in the reference section.

3.2 Areas of Overlap

Currently there is not any kind of overlap between what NDMP provides with any other process or service on Solaris. One possible overlap could happen if HSM or any other storage management that involves tapes in the process, which could be resolved or prevented by having only one of the services running at a time on Solaris.

3.3 Other changes

There could be other projects (like Anti-virus) that can introduce attributes that need to be backed up. There are some ARC cases for VFS attributes that could cause changes in NDMP. These changes need to be addressed and imported to NDMP. In general, it is better that the interface for getting/setting these attributes be implemented as part of the current generic interfaces to keep the dependencies to minimum.

4 Interfaces

This section describes the NDMP interface dependencies. NDMP is a user-space program which merely accesses the filesystem using the standard system calls and generally, there is no assumption or necessary capability needed in the underlying system for NDMP to run. NDMP relies on MTIO and USCSI library for accessing the tape and SCSI devices which they currently exist and are configurable on the Solaris servers. The main interface of NDMP is the ndmpd and ndmpadm commands which are explained in the following subsection and the man pages (See the reference section for man pages).

4.1 Exported interfaces

The following table is the summary of exported NDMP interfaces with ONNV environment:

Exported Interface	Classification	Comments
ndmpd (1)	Committed	The main daemon program (See man page in reference section)
ndmpadm (1M)	Committed	The CLI for NDMP server administration (See the man page in reference section)
ndmpstat (1M)	Committed	The CLI for reporting the statistics (See the man page in reference section)
SMF(5) properties	Committed	Saves all NDMP properties that control the behavior of daemon and NDMP service. List of exported properties are provided in 4.3
ZFS(1) user properties	Committed	ZFS relevant properties kept per dataset such as incremental backup levels.
RBAC(5) authorization	Committed	Authorization and the rights profile for NDMP service.
Files	Project Private	Log files and temporary files created during backup work. Details provided in 4.7
libndmp	Project Private	Internal library which interfaces with ndmpd(1) and ndmpadm(1M)

4.2 User Interfaces

The ndmpd (1) daemon, ndmpadm(1M) and ndmpstat(1M) CLI tools and their APIs are discussed in sections 7. These are the main exported interface which enables the user to startup the service and configure it through the CLI tools or remotely by setting the SMF properties. Ndmpstat(1M) also provides a report of basic statistics that shows the progress of backup or restore.

4.3 SMF properties

The NDMP properties or parameters that would affect the behavior of the daemon and service are defined here. The stability of these variables are 'Committed'. These variables can be changed either by using ndmpadm(1M) tools (refer to the man page in reference section) or using SMF administration tools like svccfg(1M) for service configuration. The following table defines the properties for NDMP service. The service ID is `svc:/system/ndmpd`.

Property	Default	Type	Description
dar-support	False	Boolean	Indicates if DAR or direct access recovery is supported.
tar-pathnode	False	Boolean	If the nodes leading to a changed file should be backed up during incremental backup for 'tar' backup mode.
dump-pathnode	False	Boolean	If the nodes leading to a changed file should be backed up during incremental backup for 'dump' backup mode.
ignore-ctime	False	Boolean	If it should consider modification time instead of creation time for incremental backup
token-maxseq	9	Integer (0-59)	The maximum sequence number for token-based incremental backup
version	4	Integer (2, 3, 4)	The current NDMP version
debug-path	/var/ndmp	String	The backup working directory
backup-quarantine	False	Boolean	If the quarantined files should be backed up
restore-quarantine	False	Boolean	If the quarantined files should be restored
overwrite-quarantine	False	Boolean	If the quarantined files should be overwritten during restore.
cleartext-username	Empty	String	Administrator user name to be used during clear-text authentication

Property	Default	Type	Description
cleartext-password	Empty	String	Administrator password to be used during clear-text authentication
cram-md5-username	Empty	String	Administrator user name to be used during CRAM-MD5 authentication
cram-md5-password	Empty	String	Administrator password to be used during CRAM-MD5 authentication

4.4 ZFS user properties

The following ZFS user-defined properties will be used for storing dumpdates relevant information inside the ZFS pool. For more information on dumpdates refer to Section 2.4.3:

Property	Default	Range	Description
dumpdates:L0, dumpdates:full	No	Date+time	The date and time when the last full backup has been performed.
dumpdates:L<n>	No	Date+time	The date and time when the last level <n> incremental backup has been performed.

4.5 RBAC Authorization

RBAC details are provided in ndmp_rbac.txt.

4.6 Auditing

The NDMP service will audit authentication events, success or failure, and backup or restore job initiation and termination.

4.7 Zones and Trusted Extensions (TX)

The NDMP service will run only in the global zone. If an attempt is made to start the NDMP service is made in a non-global zone, the service will exit with an SMF error.

The NDMP service does not provide support for labels. If an attempt is made to start the NDMP service on a labeled system, the service will exit with an SMF error.

4.8 Files

The working directory for NDMP daemon is saved in svc:/system/ndmpd/backup.directory. This directory keeps all log and debug files along with state files like dumpdates. The following section lists name of the files and their description along with their content format that are being used in NDMP. The taxonomy of the files are 'Project Private'.

- File name: `dumpdates`
- Description: This is a text file that keeps track of the last level of incremental backup per volume. It keeps the date along with the level for future incremental backups and the dates are referenced in determining for higher level incremental backups. ZFS filesystems uses property data, and thus have nothing in the file. Non-ZFS filesystems have up to 10 lines (for 10 levels) per filesystem. If a filesystem is no longer backed up the line is deleted. As such the size of the file is microscopic and mimics the behavior of legacy files such as ufsdump dumpdates. Since this is metadata and not a log file, there is no notion of pruning the file by logadm rotation.
- Format: The format for each entry is:

  ```
  <volume-name>       <level>    <timestamp>
  ```

  For exampe:

  ```
  /vol1                   1     Fri Jan 28 18:25:00 2006
  /vol2                   2     Sat Jan 29 12:36:00 2006
  ```

- File name: `ndmp.log`
- Description: This is a text file that logs critical errors. If the debugging is enabled it logs every function calls and their debug log messages
- Format: The entry format in this file is:

  ```
  <date> <time> [<thread-id>][<function-name>:<line-number>]:
  <message>
  ```

  For example:

  ```
  3/07 16:38:12 [319039][ndmpd_data_get_state_v4:804]:
     state: IDLE
  ```

- File name: `ndmp.`*<n>* (where *<n>* is a job number starting at 0)
- Description: This is a temporary that is created during the backup and will be removed after the backup is finished. The file contains a bitmap of the whole data set each bit representing a file which

should be either backed up or not.

- Format: Binary

4.9 libndmp

This is a 'Project Private' library that will be used by ndmadm(1M). The library interfaces with SMF and also provides the current status of the daemon to the user of the library. The only application which is using this library is ndmpadm(1M). The detail interface of this library is provided in section 7.

4.10 Imported interfaces

The following table are the list of all imported APIs inside NDMP. More detailed function level description of these interfaces are detailed in section 7.

Imported Interface	Classification	Comments
NDMP V4	Standard	See http://www.ndmp.org and section 7.1.2
mtio (7I)	Standard	For managing the magnetic tape
uscsi (7I)	Standard	For managing the robot (changer)
libbsm (3LIB)	Contracted Project Private	Audit support. PSARC/2000/517
libmd5 (3LIB)	Stable	For password encryption support
libzfs (3LIB)	Evolving (per man page)	PSARC/2002/240. For snapshot management, ZFS ACLs.
libdoor (3LIB)	Stable	For accessing ndmpd daemon data structures from ndmpadm CLI tools
libscf (3LIB)	Evolving (per man page)	For accessing SMF properties. See section 4.3 and PSARC/2007/177 for SMF PPG
libsec (3LIB)	Evolving (per man page)	ACL library.
libsocket (3LIB)	Stable	Network send and receive calls
libnsl (3LIB)	Stable	RPC and other network services
libpthread (3LIB)	Standard	P-thread library which is part of libc
libmd (3LIB)	Stable	Message digest library. Needed for MD5 checksum calls.
Libmp (3LIB)	Stable	Multiple precision library. Needed for MD5 double precision calculations.

Imported Interface	Classification	Comments
libm (3LIB)	Standard	C math library (used for MD5 checksum calculations)
libdevinfo (3LIB)	Evolving (per man page)	Device information library (used by ZFS)
libdevid (3LIB)	Stable	Device ID library (used by ZFS)
libgen (3LIB)	Stable	String pattern matching library (used by libnsl)
libnvpair (3LIB)	Evolving (per man page)	Name-value pair library (used by ZFS)

5 Design considerations

5.1 Assumptions and Dependencies

- It is assumed that there will be no conflict or functional overlap between NDMP server and current or future HSM work on Solaris.
- It is assumed that NDMP is just a user-space program that there should not have any explicit dependency on any other Solaris component such as filesystem. Also it is not dependent on S11 or S10 and should be easily back portable to any of the Solaris updates.
- It is assumed that there would be new requirements or features within other ongoing projects that might affect NDMP such as Anti-virus and HSM work. We assume that the current design is changeable with appropriate review.
- It is assumed that the tar format used in NDMP is compatible with Solaris tar command and it is possible to use tar or similar command lines to extract a backup stream which is backed up by NDMP.
- It is assumed that NDMP control and data management are separable and the current data archiving mechanism, i.e. tar, could be replaced with other archiving methods, such as ZFS send.
- It is assumed that the NDMP will run on different hardware platforms such as AMD, x86 or SPARC machines.

5.2 Goals and Guidelines

- Product to work seamlessly on any update of Solaris
- Maintain the same source code for different platforms.
- Pursue the OpenSolaris effort to resolve the potential licensing issues in NDMP code.
- Support the latest version of NDMP in the market (NDMPv4) on Solaris

5.3 Development Methods

5.3.1 Description of methodology

- The original code supporting NDMP v2 and v3 has been ported to Solaris.
- The ported NDMP code has been upgraded to v4 version of the protocol
- Integration with ZFS and SMF is done further down the line.

5.3.2 Solaris interfaces and dependencies

The following areas have been addressed during the port to Solaris:

- MTIO generic magnetic tape interface
- USCSI user SCSI command interface
- POSIX threads interface
- MD5 checksum library interface
- ZFS snapshot management interface
- SCF property group interface
- ACL library interface

5.3.3 Portable source base for all platforms

As NDMP is mostly independent from the underlying system, it should be portable and runnable on virtually all platforms as long as the standard underlying interface is maintained. The ZFS dependency is an extra “as-needed” dependency which won't cause the functionality of NDMP to break, if it is not present on the running host.

5.3.4 Open sourcing considerations

Open sourced NDMP is currently under process. The code has been prepared and ready but came across to some legal issues with the licensing on the original SDK which was first introduced on the http://www.ndmp.org website. As a result, the code will be put back under “closed” directory on Nevada gate, but later on, after resolving the issues it can be moved to the OpenSolaris community.

6 Architectural Strategies

The first release of NDMP in Sun was designed and implemented on the ST5000 series NAS products and supported only V2 and V3 versions of the protocol. The new NDMP is to support V4 and run on Solaris.

6.1 Use of Solaris MTIO library

The NDMP code accesses tape frequently but the interface in the original code wasn't standard. The original IOCTL calls have been replaced by the more general and portable MTIO library calls.

6.2 Use of Solaris USCSI library

The NDMP code accesses SCSI device directly for both tape and changer (arm or robot) to acquire status of the device or retrieve sense data. Using USCSI library gives a more general and portable interface to NDMP.

6.3 Error logging and diagnostics

Critical errors are always logged in the syslog. NDMP however has a separate logging mechanism which is configurable through the CLI ndmpd command. The critical errors with more data are always logged in the NDMP log file. A valid path needs to be provided to NDMP in order to maintain the NDMP log file. The NDMP debugging also can be done using the same log file by specifying debug levels and focuses. The following table shows the available categories and focus levels that can be specified during debugging.

Debug level	Code	Description
CAT_CONN	0x10	Connection management logs
CAT_COMM	0x20	Communication library logs
CAT_CONNECT	0x40	Connection initiation logs
CAT_CONFIG	0x80	Configuration request logs
CAT_TAPE	0x100	Tape request logs

Debug level	Code	Description
CAT_SCSI	0x200	SCSI request logs
CAT_DATA	0x400	Data interface logs
CAT_NOTIFY	0x800	Notification posts logs (server to client)
CAT_HISTORY	0x1000	File history and index logs
CAT_LOG	0x2000	Log posts logs (server to client)
CAT_MOVER	0x4000	Data mover logs
CAT_BACKUP	0x8000	Backup and restore module logs
FOC_FLOW	0x1	General backup flow focus level
FOC_DETAIL	0x2	Function details focus level
FULL_DEBUG	0xFFFF	Full log of all interfaces and focus levels.

6.4 Session and connection management

There are some issues in the connection management that should be considered in V3 and later versions. Some of these issues are considered in V3 with a provision of V4 features.

6.4.1 Configurable Listening Port

NDMP connection port on data and tape servers is 10,000 TCP by default. On some servers, this port is used by some other services so another port should be used for NDMP connection. V4 specifies that NDMP servers should be able to listen to other TCP ports too. Currently the listening port is configurable while starting up the service using -p option. Refer to ndmpd (1) man page for details.

6.4.2 Configurable Data Connection Ports

When a data server or tape server receives a listen request, the server creates TCP socket(s) and binds the socket(s) to some free ports. These addresses are passed to the DMA and then to the peer server for creating a direct data connection between two peer NDMP servers. Sometimes the other server is behind a firewall and cannot connect to that TCP port reported by the other server, so establishing the connection fails.

V4 addresses this problem. The NDMP servers should be configurable and listen to some specified ports instead of any unused port. The specified ports will be open by the firewall and so that the other peer server can connect to the listener server. This feature is not implemented yet but will be added through

the SMF variables using a specific variable called *dc.ports*, which holds a comma-separated list of TCP port ranges. For example, 12000-12050,13400-13440,14000, which means that the port numbers 12000 through 12050, 13400 through 13440 and port 14000 can be used for creating data connection. The default value for this parameter, is any free port, which will be assigned by the system automatically at run-time.

6.5 Concurrency and synchronization

NDMP server depends on many threads mostly designated for reading or writing to network, filesystem or a tape device. These threads are synchronized using mutexes and condition variables. There is a simple IPC buffers which consists of multiple wrapping buffers that is used as pipes and IPC between the running producer and consumer threads. These buffers are part of TLM library which is explained in 2.5.1

6.5.1 Reader/writer threads

The following table shows the reader/writer threads in NDMP.

Thread	Comments
MoverTapeReader	Mover thread which reads from the tape during restore
MoverTapeWriter	Mover thread which writes to the tape during backup
MoverSocketReader	Mover thread which reads from the network during remote backup
MoverSocketWriter	Mover thread which writes to the network during remote restore
NdmpBackupReader	The thread that actually performs the backup
NdmpTarReader	The thread that actually performs the restore
ndmp_worker	The worker thread that handles each NDMP connection
tlm_BackupReader	TLM thread which reads the filesystem during backup
tlm_RestoreWriter	TLM thread which writes the files to the disk during restore

6.5.2 Concurrent access issues

During the transition to Solaris, one of the main changes was migrating from a non-preemptive thread based environment to a preemptive model. This needed more protection and locking for some critical and commonly used data structures. Extra locks and controls are added for these to the NDMP code.

6.6 Communication mechanisms

As a RPC based protocol, all the control communication to the server is obtained using RPC calls for processing the NDMP messages passed to the server. Other data communications are either through the local IPC buffers for the internal threads or for running a local backup on the server. For a remote backup the data channel between the Data and Mover interfaces is provided by TCP network sockets.

6.6.1 RPC calls

These calls consist of NDMP control messages that are sent from the DMA to the NDMP server. This is the only control communication mechanism between the client and the server in any backup scenario. Some of the messages are sent from the server to the client like logs and index catalog information, i.e., file history.

6.6.2 TCP Data connections

The TCP data connection is designed for performing remote backup/restore or copying. An NDMP server can be instructed by DMA to create and open a socket for incoming remote requests both on a data interface and mover interface. The data interface resides on a data server which handles the data transfer between the network socket and the filesystem. A mover interface which is on a backup server handles the tape stream data channeling between the network socket and the tape drives and media.
The model for opening such a socket on two distant data server and backup server would be as following:

Backup scenario	Action
Remote backup from a data server to a backup server	- DMA sends a MOVER_LISTEN request to the backup server. - The backup server creates a socket and listens to the port and sends back the information to the DMA - DMA sends a DATA_CONNECT request to the data server along with the socket information - Data server connects to the backup server to perform a backup.
Remote restore from a backup server to a data server	- DMA sends a DATA_LISTEN request to the data server. - The data server creates a socket and listens to the port and sends back the information to the DMA - DMA sends a MOVER_CONNECT request to the backup server along with the socket information - Backup server connects to the data server to perform a restore.

Backup scenario	Action
Disk to disk copy from a data server to another data server (Solaris to Solaris, no tapes necessary)	- DMA sends a DATA_LISTEN request to the destination server. - The destination server creates a socket and listens to the port and sends back the information to the DMA - DMA sends a DATA_CONNECT request to the source server along with the socket information - Source server connects to the destination server to perform the copying.
Tape to tape copy from a backup server to another backup server (no copying to the disk)	- DMA sends a MOVER_LISTEN request to the destination server. - The destination server creates a socket and listens to the port and sends back the information to the DMA - DMA sends a MOVER_CONNECT request to the source server along with the socket information - Source server connects to the destination server to perform the copying.

6.6.3 Byte ordering

NDMP runs as an application layer user-space program and can communicate with any other NDMP server existing in the network. Interoperability is not an issue as the nodes communicate with each other via the protocol and it would be independent of the underlying operating system and hardware platform. In order to properly handle the data which is passed between different hardwares, such as byteordering issues, all the binary metadata are converted to text before being saved on the tape or being transferred to another NDMP host. This way each host could extract the data to the properly byteordered binary format before using it.

If the ZFS(1M) send command is used as archiving method instead of tar(1), this issue is automatically handled while creating ZFS backup stream for ACLs through ZFS send.

6.6.4 Connection failures

Normally the client will be notified of any hardware or internal errors happening during the backup operation by NDMP_NOTIFY messages. However if the control connection fails it would be hard to reach the DMA to notify the problem. It this situation each DMA will behave differently. Some DMA's will try to reconnect while others will abort the current operation and stop. As in NDMP, there is not a separate descriptor or handle for NDMP sessions, all sessions are bound to the TCP connections. This means that even after the reconnect the previous operation cannot be resumed and all the progress data will be lost. Because of that, the behavior of the NDMP server in case of connection failure will always

be the same, which is closing the session and cleaning up the state. The cleanup involves deallocating and resetting the state variables and removing the temporary snapshots.

6.6.5 Server Authentication

Before a DMA can begin a backup or restore operation on an NDMP server it must be authenticated by the server. The NDMP specification supports two modes for user/password authentication:

- Text mode: The DMA identity is authenticated using an `auth_id` representing the user account name and an unencrypted (clear text) password.
- MD5 mode:The DMA identity is authenticated using an `auth_id` based on CRAM-MD5, as defined in RFC 1321. The MD5 `auth_digest` is based on a random server challenge and a shared secret (password).

An NDMP user identity and password is created on the NDMP server by using the ndmpadm command. The username and password is stored as a Protected Property Group in SMF (PSARC 2007/177) with read authorization. The default value for the password is null-string which will be interpreted as "no access" by default. The password can be changed by ndmpadm(1M) tools which is run by the administrator which has proper authorizations (see ndmp_rbac.txt). Subsequently during backup or restore operations, the user name and password is sent from the DMA to the server either as clear text or as an MD5 digest. The server retrieves a separate password for each password type from SMF and compares it to the clear text password (text-mode) or produces the MD5 digest based on the cram-md5 password (MD5-mode) and compares it to the received digest.

6.7 Filesystem issues

NDMP needs to access filesystem both at backup and restore time. These accesses are limited to the standard FS system calls and basically is filesystem independent. These accesses are basically for open/close, reading and writing the data and setting or checking the file attributes and timestamps. NDMP would work the best if the filesystem has support for checkpoints or snapshot like ZFS, but at the same time, it would work without any issues on UFS and other types of filesystem with standard FS system call interfaces.

6.7.1 Filesystem type and capabilities

The filesystem default type would be UFS or any UFS-like filesystem. However NDMP has extra capabilities for filesystems with snapshots. Currently NDMP uses libzfs to perform some of the ZFS

snapshot controls but ultimately it could be replaced with general IOCTLs to control the creation and handling of snapshots. If the snapshot capability is not available the backup is taken from the live filesystem instead of the snapshot. If the backup path is originally a path of a snapshot, e.g., mypool/myfs@snap1 or mypool/myfs/.zfs/snapshot/snap1, the backup would be taken directly from the snapshot. NDMP finds out about the filesystem type using getmentent(3C) and hasmntopt(3C).

6.7.2 Standard file attributes

The basic standard file attributes such as mode and permissions, owner and group and the timestamps are always backed up and restored along with the files and directories. Some attributes like CTIME timestamp is also used by NDMP internally to determine certain actions like incremental backup.

6.7.3 Extended file attributes

6.7.4

Other than standard attributes, which are backed up along with the file, the extended file attributes are also checked for each file and, if they exist, they will be backed up too. This consists of backing up the contents of the attribute files and their attributes as well. The tar engine in TLM is extended to have support for these attributes.

NFSv4 ACLs are also backed up in NDMP using acl_get(3SEC) and acl_set(3SEC) calls. To avoid potential potential byte-ordering issues, ACLs are converted to text before getting backed up.

6.7.5 Checkpoints or snapshots

NDMP needs to make a snapshot of the volume or filesystem before backing it up. The backup is taken from the snapshot and it is removed after the backup is finished successfully. NDMP uses the following generic calls to manipulate the snapshots:

- GETSTATUS returns a snapshot status structure for checking if it is enabled or disabled.
- GETENTRY returns the next snapshot of the given volume in a snapshot entry structure. The structure contains the name and the date of creation of the snapshot.
- CREATE creates a snapshot with the given name for a volume.
- REMOVE removes a snapshot.
- RENAME renames a snapshot to a different name. This is useful when backup fails.

Currently the above is obtained using libzfs (3LIB) library calls. Refer to section 7.1.2.

This won't be needed if ZFS send is used as archiving component as ZFS send directly creates the backup stream from a already created snapshot.

7 Detailed System Design

7.1 Ndmpd

7.1.1 Purpose

This is the main CLI command that brings up the NDMP server with different command line options. Normally when it started it tries to detect the locally attached tape drives and libraries. If none found the NDMP server runs as a data server to the remote backup servers. If local devices are found it could be both a data and backup server.

7.1.2 Interfaces

Exported interfaces

There are different options and parameters that can be passed to the ndmpd command. Refer to ndmpd(1) man page for more details.

Imported interfaces from NDMP protocol

The interface with NDMP protocol is considered a 'Standard' interface. For more details of the protocol refer to the NDMP v4 functional specification in the reference section. This interface is based on a set of ONC/RPC calls over TCP connection from DMA client which are called NDMP messages. Although version four of the protocol is considered here, the NDMP server is able to use earlier versions of the protocol, v3 and v2 as well. The following table is a summary of NDMP massages based on the protocol version (✕ means that the command is not supported or no longer exists in the protocol).

NDMP message	V2	V3	V4
CONNECT_OPEN	✓	✓	✓
CONNECT_CLIENT_AUTH	✓	✓	✓
CONNECT_CLOSE	✓	✓	✓
CONNECT_SERVER_AUTH	✓	✓	✓
CONFIG_GET_HOST_INFO	✓	✓	✓

NDMP message	V2	V3	V4
CONFIG_GET_SERVER_INFO	✗	✓	✓
CONFIG_GET_CONNECTION_TYPE	✓	✓	✓
CONFIG_GET_AUTH_ATTR	✓	✓	✓
CONFIG_GET_BUTYPE_INFO	✗	✓	✓
CONFIG_GET_BUTYPE_ATTR	✓	✗	✗
CONFIG_GET_FS_INFO	✗	✓	✓
CONFIG_GET_TAPE_INFO	✗	✓	✓
CONFIG_GET_SCSI_INFO	✗	✓	✓
CONFIG_GET_EXT_LIST	✗	✗	✓
CONFIG_SET_EXT_LIST	✗	✗	✓
SCSI_OPEN	✓	✓	✓
SCSI_CLOSE	✓	✓	✓
SCSI_GET_STATE	✓	✓	✓
SCSI_RESET_DEVICE	✓	✓	✓
SCSI_EXECUTE_CDB	✓	✓	✓
SCSI_SET_TARGET	✓	✓	✗
SCSI_RESET_BUS	✓	✓	✗
TAPE_OPEN	✓	✓	✓
TAPE_CLOSE	✓	✓	✓
TAPE_GET_STATE	✓	✓	✓
TAPE_MTIO	✓	✓	✓
TAPE_WRITE	✓	✓	✓
TAPE_READ	✓	✓	✓
TAPE_EXECUTE_CDB	✓	✓	✓
DATA_CONNECT	✓	✓	✓
DATA_LISTEN	✓	✓	✓
DATA_START_BACKUP	✓	✓	✓
DATA_START_RECOVER	✓	✓	✓
DATA_START_RECOVER_FILEHIST	✗	✗	✓
DATA_GET_STATE	✓	✓	✓

NDMP message	V2	V3	V4
DATA_GET_ENV	✓	✓	✓
DATA_STOP	✓	✓	✓
DATA_ABORT	✓	✓	✓
MOVER_SET_RECORD_SIZE	✓	✓	✓
MOVER_SET_WINDOW	✓	✓	✓
MOVER_CONNECT	✗	✓	✓
MOVER_LISTEN	✓	✓	✓
MOVER_READ	✓	✓	✓
MOVER_GET_STATE	✓	✓	✓
MOVER_CONTINUE	✓	✓	✓
MOVER_CLOSE	✓	✓	✓
MOVER_ABORT	✓	✓	✓
MOVER_STOP	✓	✓	✓
NOTIFY_DATA_HALTED	✓	✓	✓
NOTIFY_CONNECTION_STATUS	✓	✓	✓
NOTIFY_MOVER_HALTED	✓	✓	✓
NOTIFY_MOVER_PAUSED	✓	✓	✓
NOTIFY_DATA_READ	✓	✓	✓
LOG_MESSAGE	✗	✓	✓
LOG_FILE	✗	✓	✓
LOG_LOG	✓	✗	✗
LOG_DEBUG	✓	✗	✗
FH_ADD_FILE	✗	✓	✓
FH_ADD_DIR	✗	✓	✓
FH_ADD_NODE	✗	✓	✓
FH_ADD_UNIX_PATH	✓	✓	✗
FH_ADD_UNIX_DIR	✓	✓	✗
FH_ADD_UNIX_NODE	✓	✓	✗

These messages which are fixed based on the protocol version being used, but there are two areas in the protocol that are extensible depending on the vendor's needs: NDMP v4 extensions and NDMP

environment variables.

The NDMP community can develop and standardize new functionality in NDMP without requiring a revision of core NDMP. The new functionality could be proprietary of a vendor and DMA can discover and negotiate the use of these extensions through NDMP standard messages. NDMP extensions are identified by *classes*. Currently there are two classes of extensions: standard extensions and proprietary extensions. Standard extensions need to be standardized by the NDMP community into a separate standard specification and will be owned by the community while the proprietary extensions are owned by the implementers of the NDMP server. The upper 16-bit of the 32-bit NDMP message code is chosen for encoding the NDMP class. That means that there are virtually 64K classes available. These class spaces are allocated into separate ranges as following:

Class	Range
Core NDMP	0x0000
Standard NDMP extensions	0x0001 – 0x0007
Reserved	0x0008 – 0x1FFF
Proprietary extensions	0x2000 – 0x7FEF
Reserved for test use	0x7FF0 – 0x7FFE
Reserved	0x7FFF - 0xFFFF

NDMP extensions are not currently considered in this release and its current status is TBD. However as the protocol version is V4 it still has the extensibility feature for including the extensions in the future work. For more information about NDMP extensions refer to NDMP v4 functional specification link in the reference section.

Environment variables are a set of name/value pairs which are passed between the NDMP server and DMA on certain occasions. These variables contain information such as backup path, options, types and other information necessary during backup. The server needs to keep these variables and their values along with a NDMP session. Environment variables should not be mistaken with the NDMP configuration parameters which are settable by ndmpadm(1M) or SMF. Environment variables are only set by the DMA or server daemon itself. There are two sets of environment variables in NDMP, standard variables and proprietary variables. For both cases, DMA can get the list of all environment variable using NDMP_DATA_GETENV message.

The following environment variables can be defined by the NDMP Server and used for all backup types:

Variable Name	Meaning	Supported Values	Default Value
PREFIX	Prefix path for the request	path name	(no default value)
TYPE	The backup type	dump, tar	(no default value)
USER	User ID to run backup	User name	(no default value)
DIRECT	Utilize direct access recovery (DAR)	Yes or No	No

Variable Name	Meaning	Supported Values	Default Value
HIST	Flag to generate file history data	Yes or No	No

The following environment variables are specific to the **dump** backup type:

Variable Name	Meaning	Supported Values	Default Value
FILESYSTEM	Device of filesystem nameto be backed up	Filesystem or device name	(no default value)
LEVEL	dump level	0 - 9	0
EXTRACT	Dump extract option (-x or -r)	Yes for -x No for -r	Yes
UPDATE	Update the dumpdates file	Yes or No	Yes

The following environment variable is specific to the **tar** backup type:

Variable Name	Meaning	Supported Values	Default Value
FILES	List of files to be backed up.	Eg. ./* , ./*.c, ...	(no default value)

Note: The cpio backup type is not supported in this implementation of NDMP

The following environment variables are specific to the **zfs** backup type:
(Note: ZFS backup is outside the scope of this project and will be defined in a future project)

Variable Name	Meaning	Supported Values	Default Value
TBD / future			

The following environment variables can be used with the NDMP_CONFIG_GET_FS_INFO command:

Variable Name	Meaning	Supported Values	Default Value
LOCAL	Whether file system is Local to the machine on which the data service is running	Yes or No	Yes
TYPE	Type of file system	UFS and ZFS are supported; However, ZFS filesystems are currently treated like UFS but the backup is taken from the snapshot rather than the live filesystem. ZFS ACLs will be backed up.	UFS
AVAILABLE_BACKUP	Backup types available on target server	dump, tar	dump, tar
AVAILABLE_RECOVERY	Restore types available on target server	dump, tar	dump, tar

The following environment variables can be

Variable Name	Meaning	Supported Values	Default Value
EXECUTE_CDB	Comand supported for tape as well as other generic SCSI devices	t-tape, s-scsi, b-both, n-none	t
COMPRESSION	Compression ration	1 – no compression	1 – no compression

There are other proprietary environment variables that could be used during backup. These environment variables must be recognized from both NDMP server and DMA. Currently we have the following proprietary environment variables:

Variable Name	Meaning	Supported Values	Default Value
EXCLUDE	Specify the files to be excluded from backup	Path name list	No default
DMP_NAME	Dump date file in token based backup (Section 2.5.4)	Path name	No default
BASE_DATE	Based date in token based backup (Section 2.5.4)	Date value	No default

Variable Name	Meaning	Supported Values	Default Value
DUMP_DATE	Dump date value in token based backup (Section 2.5.4)	Date value	No default

Above variables are only set by certain vendors (e.g. Tivoli Storage Manager) but if they are present, NDMP server would recognize them and do the proper action. The DMA can find out about these variables and whether or not they are supported by issuing a NDMP_DATA_GET_ENV message to the server.

Imported interfaces from TLM unit

The following function calls are imported from TLM unit. As TLM is not a shared library, these calls are only 'Project Private' internal calls.

Interface	Comments
tlm_BufferAdvanceInIdx	Advance the input index of the IPC buffers and return pointer to the next buffer in the buffer pool.
tlm_BufferAdvanceOutIdx	Advance the output index of the IPC buffers and return pointer to the next buffer in the buffer pool.
tlm_BufferMarkEmpty	Mark the buffer as empty and clears its flags
tlm_BufferReleaseInBuf	Another buffer is filled. Wake up the consumer if it's waiting for it.
tlm_BufferReleaseOutBuf	A buffer is used. Wake up the producer to re-fill a buffer if it's waiting for the buffer to be used.
tlm_BuildCheckpointName	Create the checkpoint/snapshot name out of a volume name
tlm_CreateReaderWriterIPC	Create the IPC area between the reader and writer thread
tlm_DbgEnable	Enable TLM debugging.
tlm_DecodeBkType	Decode the backup type and return the operation flags based on that
tlm_GetDataOffset	Return the data offset in the IPC buffer pool
tlm_GetReadBuffer	Get a read record from the tape buffer, and read a tape block if necessary.
tlm_GetWriteBuffer	Get the next tape buffer from the drive's pool of buffers.
tlm_Get_ChkpntTime	Find out the time of creation of the current checkpoint or snapshot
tlm_Init	Initialize the TLM unit
tlm_IsExcluded	Find out if the file is considered as excluded.
tlm_NewDirInfo	Create a new dir info structure consisting of the backup path and the current directory.

Interface	Comments
tlm_NewJobStats	Create a new link for job statistics
tlm_OutputDir	Put the directory information into the output buffer
tlm_OutputFile	Put the file information into the output buffer
tlm_OutputLine	Write a line to a file, make sure not a short write.
tlm_OutputXAttr	Put the extended attribute information into the output buffer
tlm_PrintLine	Write a line to a file, used for creating archiving information
tlm_RefJobStats	Increase the reference count, so that others wont remove the job status
tlm_ReleaseList	Free a null terminated array of pointers
tlm_ReleaseReaderWriterIPC	Release the IPC buffer pool
tlm_RestoreFiles	Restore files from the tape, put it in the buffer pool
tlm_RestoreWriter	Restores files from the buffer pool to the disk
tlm_UnRefJobStats	Unreference the job status for others to delete
tlm_catPath	Concatenates two paths considering different '/' positions.

Imported interface for TAR archiving from TLM unit

This is the archiving ops table which is discussed in section 2.7.1. TLM unit only can provide tar(1) archiving so far and other archiving formats such as cpio(1) and ZFS(1M) need to be added separately in other phases of this project. This ops table is considered a 'Project Private' internal calls.

Operation	Interface	Comments
int (*tm_putfile)()	tlm_OutputFile()	Puts a single file in the backup stream
int (*tm_putdir)()	tlm_OutputDir()	Puts a directory tree in the backup stream
int (*tm_putvol)()	NULL	Puts the snapshot of a volume or filesystem in the backup stream (not defined for tar)
int (*tm_getfile)()	tlm_RestoreFile()	Extracts a single file from the backup stream
int (*tm_getdir)()	tlm_RestoreFile()	Extracts a directory tree from the backup stream. If the target name is the directory it will extract the directory.
int (*tm_getvol)()	NULL	Restores a snapshot of a volume or filesystem from the backup stream (not defined for tar)

Imported interface for directory tree traversal from TLM unit

These calls are used to quickly traverse a directory tree. The post-order is used for checking conditions

and dependencies and the level-order is for the actual backup. As TLM is not a shared library, these calls are only 'Project Private' internal calls.

Interface	Comments
fstrave_level	Traverse the filesystem in a level-order way.
fstrave_post	Traverse the filesystem in a post-order way.

Imported interface for bitmap manipulation from TLM unit

Bitmap file is used to indicate which files need to be backed up. The full backup is always a bitmap of all ones. As TLM is not a shared library, these calls are only 'Project Private' internal calls.

Interface	Comments
dbm_alloc	Allocate a bitmap file and return a handle to it
dbm_free	Free the bitmap
dbm_getlen	Return length of the bitmap
dbm_getone	To get a one bit value in the bitmap
dbm_setone	To set a one bit value in the bitmap

Imported interface for CRAM-MD5 authentication

Calls imported from libmd5 (3LIB) at stability level 'Stable'.

Interface	Comments
MD5Init	To intialize a new context
MD5Update	MD5 block update operation
MD5Final	Ends the MD5 operation and outputs the digest

Imported interface for SCF property

This interface is used to implement persistent storage for saving the NDMP configuration parameters. These parameters are mainly set by the ndmpadm(1M) command or could be also set by svccfg(1M) directly using the NDMP service identifier. Refer to ndmpadm(1M) man page for more information.

This interface is imported from libscf (3LIB) at stability level 'Evolving'.

Interface	Comments
scf_entry_add_value	Adds a new value for the transaction entry
scf_entry_destroy	Deallocate values associated with the transaction entry

Interface	Comments
scf_entry_create	Allocates a new transaction entry handle
scf_pg_get_property	Sets property to the one specified by name in the property group
scf_property_get_value	Retrieves the single value that the property to which is set
scf_property_destroy	Deallocate property
scf_property_create	Allocates and initializes a new property bound to a handle
scf_service_get_pg	Get the property group specified by name
scf_service_add_pg	Create a new property group specified by name
scf_transaction_property_delete	Delete a property
scf_transaction_create	Allocates and initializes a transaction bound to a handle
scf_transaction_start	Set up a transaction to modify the property group
scf_transaction_property_change	Change a transaction entry for the transaction
scf_transaction_property_new	Add a new transaction entry to the transaction
scf_transaction_destroy_children	Deallocates and frees all entries and values associated with a transaction
scf_transaction_destroy	Deallocate the transaction
scf_transaction_commit	Attempts to commit the transaction
scf_value_create	Creates a new value that holds a single typed value
scf_value_set_astring	Set value for ascii character string
scf_value_destroy	Deallocate the value

Imported interface from ZFS library

The following interfaces are used inside NDMP mainly for snapshot handling from libzfs (3LIB). The stability level for this library is 'Evolving' per man page. A contract will not be necessary as libzfs and NDMP are going to be integrated in the same gate.

Interface	Comments
zfs_get_name	Get the zfs name of a given handle
zfs_open	Open a zfs volume handle
zfs_iter_snapshots	Iterative call backs for all snapshots of a volume
zfs_close	Close zfs volume handle
libzfs_init	Initialize libzfs library
libzfs_fini	Terminate use of libzfs library
zfs_snapshot	Create a snapshot

Interface	Comments
zfs_destroy	Remove a snapshot
zfs_rename	Rename a zfs volume (used for snapshot)
zfs_prop_get_int	Get zfs property (used for snapshot creation date/time)

Imported interface from P-Thread library

P-thread library which resides in libc (3LIB) is used for implementing most of the thread functionality inside NDMP and TLM code. The stability level is 'Standard'.

Interface	Comments
pthread_create	Create a new thread
pthread_cond_init	Initialize the condition variable
pthread_cond_destroy	Destroy the given condition variable
pthread_cond_wait	Block on a condition variable
pthread_cond_reltimedwait_np	Block with a relative time-out value
pthread_cond_signal	Unblock at least one of the threads that are blocked
pthread_mutex_lock	Lock the mutex
pthread_mutex_unlock	Unlock the mutex

Imported interface from socket library

The following interface is imported from libsocket (3LIB) library. The stability level is 'Stable':

Interface	Comments
accept	Accept a connection on a socket
bind	Bind a name to a socket
connect	Initiate a connection on a socket
getpeername	Get name of connected peer
getsockname	Get socket name
getsockopt	Get options of a socket
htonl	Convert host to network for long values
htons	Convert host to network for short values
listen	Listen for connections on a socket
ntohl	Convert network to host for long values

Interface	Comments
ntohs	Convert network to host for short values
inet_ntoa	Convert network address to string
setsockopt	Set options for a socket
socket	Create a socket

Imported interface from libnsl library

Libnsl (3LIB) library is used for RPC calls and XDR data representation routines such as xdr_int, xdr_short, etc. The stability level of this library is 'Stable'. Due to numerous usage of this library the interface is not expanded here. The code is generated using rpcgen (1) compiler.

Imported interface from libsec library

This interface is used for handling NFSv4 and ZFS Access Control Lists during backup and restore. The stability of libsec (3LIB) library is 'Evolving' per man page.

Interface	Comments
acl_get	Get the files ACL
acl_set	Set the files ACL
acl_totext	Convert ACL to text external representation
acl_fromtext	Convert text external representation to ACL

7.2 Ndmpadm

7.2.1 Purpose

This is the main API or command line tools on the server side which queries the daemon and reports about the current active connections and their status. It also can change and update the parameters and variables for that daemon which will affect the behavior of the server.

7.2.2 Interfaces

Exported interface to the user

There are different options and sub-commands that can be passed to the ndmpadm(1M) command. Refer to ndmpadm(1M) man page for details. This interface is considered 'Committed'.

Imported interface for SCF property

Same interface used as ndmpd. Refer to the ndmpd imported interface for SCF above. The interface is 'Evolving'.

Imported interface for door calls

This interface is imported from libdoor(3LIB) with stability level of 'Stable'.

Interface	Comments
door_create	Create a door descriptor
door_call	Invoke the function associated with a door descriptor
door_return	Return from a door invocation

7.3 Ndmpstat

7.3.1 Purpose

This is a command line that reports the statistics of backup or restore. The statistics shows the number of threads involved at each interval and the number of IO operations for both disks and tapes are reported. This command also shows an estimate performance of the current operation along with the utilization of the resources showing that what percentage of the time is spent for disk, tape and other activities separately.

7.3.2 Interfaces

Exported interface to the user

The description of the options and the details about the output and the format of the reporting of the statistics are found in ndmpstat(1M) man page. This interface is considered 'Committed'.

Imported interface for door calls

This interface is imported from libdoor(3LIB) with stability level of 'Stable'. Using door calls, nndmpstat(1M), gathers the collected information by ndmpd(1) daemon and creates a report per given intervals on the screen.

Interface	Comments
door_create	Create a door descriptor
door_call	Invoke the function associated with a door descriptor
door_return	Return from a door invocation

7.4 Libndmp

7.4.1 Purpose

This is a private library that provides an API for getting and setting configuration SMF values and also getting the current status of the ndmpd(1) daemon through door calls. This interface is considered 'Project Private' and is only used by ndmpadm(1M).

Exported interface to ndmpadm

The following function calls are exported from libndmp. These calls are 'Project Private' calls.

Interface	Comments
ndmp_get_devinfo	Get the local tape library device information
ndmp_get_devinfo_memfree	Free the memory allocated for devices information
ndmp_get_dbglevel	Get the current debugging level
ndmp_get_session_info	Get the current session information
ndmp_get_session_info_mem free	Free the memory allocated for session information
ndmp_get_stats	Get the current running daemon statistics
ndmp_terminate_session	Terminate or kill the give session
ndmp_loadenv	Load the configuration SMF variables in the cache
ndmp_getenv	Get the current value of the variable in the cache
ndmp_getenv_d	Get the value from the cache, use the default if not set
ndmp_get_yorn	Get the value of the yes/no or boolean variable
ndmp_setenv	Set the value of the given variable

Imported interface

The imported interface will be exactly as ndmpadm(1M).

7.5 TLM

7.5.1 Purpose

This is a Tape Library Module which provides basic functionality to access and manipulate the tape and the robot library. It also provides the tar engine for creating and extracting the archive format which in this case is tar format. The tar format used here is compatible with Solaris tar implementation. The interface for archiving component is defined by archiving ops table discussed in section 2.7.1. This code relies on MTIO(7I), generic magnetic tape interface and USCSI(7I), user SCSI command interfaces for certain SCSI and tape operations.

7.5.2 Interfaces

Exported interface to NDMP module

The following function calls are exported from TLM unit. As TLM is not a shared library, these calls are only 'Project Private' internal calls.

Interface	Comments
tlm_BufferAdvanceInIdx	Advance the input index of the IPC buffers and return pointer to the next buffer in the buffer pool.
tlm_BufferAdvanceOutIdx	Advance the output index of the IPC buffers and return pointer to the next buffer in the buffer pool.
tlm_BufferMarkEmpty	Mark the buffer as empty and clears its flags
tlm_BufferReleaseInBuf	Another buffer is filled. Wake up the consumer if it's waiting for it.
tlm_BufferReleaseOutBuf	A buffer is used. Wake up the producer to re-fill a buffer if it's waiting for the buffer to be used.
tlm_BuildCheckpointName	Create the checkpoint/snapshot name out of a volume name
tlm_CreateReaderWriterIPC	Create the IPC area between the reader and writer thread
tlm_DbgEnable	Enable TLM debugging.
tlm_DecodeBkType	Decode the backup type and return the operation flags based on that
tlm_GetDataOffset	Return the data offset in the IPC buffer pool
tlm_GetReadBuffer	Get a read record from the tape buffer, and read a tape block if necessary.
tlm_GetWriteBuffer	Get the next tape buffer from the drive's pool of buffers.
tlm_Get_ChkpntTime	Find out the time of creation of the current checkpoint or snapshot
tlm_Init	Initialize the TLM unit
tlm_IsExcluded	Find out if the file is considered as excluded.

Interface	Comments
tlm_NewDirInfo	Create a new dir info structure consisting of the backup path and the current directory.
tlm_NewJobStats	Create a new link for job statistics
tlm_OutputDir	Put the directory information into the output buffer
tlm_OutputFile	Put the file information into the output buffer
tlm_OutputLine	Write a line to a file, make sure not a short write.
tlm_OutputXAttr	Put the extended attribute information into the output buffer
tlm_PrintLine	Write a line to a file, used for creating archiving information
tlm_RefJobStats	Increase the reference count, so that others wont remove the job status
tlm_ReleaseList	Free a null terminated array of pointers
tlm_ReleaseReaderWriterIPC	Release the IPC buffer pool
tlm_RestoreFiles	Restore files from the tape, put it in the buffer pool
tlm_RestoreWriter	Restores files from the buffer pool to the disk
tlm_UnRefJobStats	Unreference the job status for others to delete
tlm_catPath	Concatenates two paths considering different '/' positions.
fstrave_level	Traverse the filesystem in a level-order way.
fstrave_post	Traverse the filesystem in a post-order way.
dbm_alloc	Allocate a bitmap file and return a handle to it
dbm_free	Free the bitmap
dbm_getlen	Return length of the bitmap
dbm_getone	To get a one bit value in the bitmap
dbm_setone	To set a one bit value in the bitmap
tar_ops {...}	This is the archiving ops table which provides the primary functions for archiving and extracting files, directories and the filesystem. See 7.1.2 for more details.

Imported IOCTL operations from MTIO

These IOCTL calls are imported from mtio(7I) interface with stability level 'Standard':

- MTIOCTOP: This is a tape operation command, heavily used for tape positioning and filemark operations like forward skip filemark (FSF), write EOF, backward skip record (BSR). It is also used for erasing, rewinding and unloading the tape.
- MTIOCGET: This is a get status command and is used in NDMP very often. It reports the position of the tape to the client along with the current file number, block number, and other useful info.

- MTIOCGETDRIVETYPE: This is a get config data command and is used for getting the tape drive specific config information like the name, block size, vendor ID, type, densities supported and more.

Imported IOCTL operations from USCSI

These IOCTL calls are imported from uscsi(7I) interface with stability level 'Standard':

- USCSI_WRITE: This command is used for executing SCSI CDB commands sent by the NDMP clients. It also is used by TLM unit for running inventory, moving tapes from drive to slots and vice versa or other things like locking the library door.
- USCSI_READ: This command is used for executing SCSI CDB commands sent by the NDMP client and also requesting the sense data from the SCSI device. Other than that, sending the device inquiry, getting the element address page and element status are other uses of this command.
- USCSI_RESET: This is used for resetting the SCSI device.

8 Acknowledgements

The format of this document is adapted from: http://www.construx.com/survivalguide/desspec.htm Web page copyright © 1993-2002 Steven C. McConnell. Permission is hereby given to copy, adapt, and distribute this material as long as this notice is included on all such materials and the materials are not sold, licensed, or otherwise distributed for commercial gain. Software Design Specification copyright © 1994-1997 by Bradford D. Appleton