



**mdtmFTP**  
**Installation & Configuration Manual**  
➤ **Docker Release**

**Version 1.0.3**

**Fermilab**  
**Jan 2019**

## About

To address challenges in high performance data movement for large-scale science, the Fermilab network research group has developed mdtmFTP, a high-performance data transfer tool to optimize data transfer on multicore platforms. mdtmFTP has a number of advanced features.

- First, it adopts a pipelined I/O design. Data transfer tasks are carried out in a pipelined manner across multiple cores. Dedicated threads are spawned to perform network and disk I/O operations in parallel.
- Second, mdtmFTP uses multicore-aware data transfer middleware (MDTM) to schedule an optimal core for each thread, based on system configuration, to optimize throughput across the underlying multicore platform.
- Third, mdtmFTP implements a large virtual file mechanism to efficiently handle lots-of-small-files (LOSF) situations.
- Finally, mdtmFTP unitizes optimization mechanisms such as zero copy, asynchronous I/O, batch processing, and pre-allocated buffer pools, to maximize performance.

This document describes the installation and basic use of mdtmFTP.

The mdtmFTP project website: <http://mdtm.fnal.gov>

The mdtmFTP docker release: <https://hub.docker.com/r/wenji/mdtm/>

For mdtmFTP technical details, please refer to paper:

Liang Zhang, Wenji Wu, Phil DeMar, Eric Pouyoul:  
mdtmFTP and its evaluation on ESNET SDN testbed. Future Generation Comp. Syst. 79: 199-204 (2018)

## Contacts

Wenji Wu ([wenji@fnal.gov](mailto:wenji@fnal.gov))  
Liang Zhang ([liangz@fnal.gov](mailto:liangz@fnal.gov))  
Phil DeMar ([demar@fnal.gov](mailto:demar@fnal.gov))  
Sajith Sasidharan ([sajith@fnal.gov](mailto:sajith@fnal.gov))

## Intended Audience

This manual is intended for users and system administrators responsible for installing, running, and managing DTNs.

The manual assumes familiarity with multicore and DTN concepts.



## Acknowledgements

mdtmFTP uses several Globus modules (<http://toolkit.globus.org/toolkit>) for rapid prototyping. We sincerely thank Globus folks at Argonne National Laboratory and University of Chicago.

Here is a list of Globus modules that mdtmFTP uses:

- GridFTP protocol module
- Globus xio module
- Globus security module
- Globus user interface

## Section 1. System level requirements

- 1) System must have installed Docker (version 1.10 +). The Docker project website is available at <http://www.docker.com>. For some Linux distributions, you can install Docker packages through *yum* or *apt-get*.
- 2) Download and install mdtmFTP Docker package
  - The mdtmFTP Docker repository: <https://hub.docker.com/r/wenji/mdtm>
  - Download mdtmFTP container:

```
“docker pull docker.io/wenji/mdtm:mdtmFTP”
```
  - Run `“docker images”` to check the container that you have pulled

## Section 2. The mdtmFTP Docker Container

1) First start a mdtmFTP docker container by running

```
“docker run -ti --net=host docker.io/wenji/mdtm:mdtmFTP”
```

This command will start the mdtmFTP container interactively. You can login the container to launch applications, or edit/configure files.

- In the container, mdtmFTP files are located at the following folders:
  - mdtmFTP client folder @ `“/home/mdtmftp_client”`
    - `mdtm-ftp-client`, the mdtmFTP client executable
    - `mdtmconfig.xml`, configures a mdtmFTP client’s MDTM-related parameters
  - mdtmFTP server folder @ `“/home/mdtmftp_server”`
    - `mdtm-ftp-server`, the mdtmFTP server executable
    - `mdtmconfig.xml`, configures a mdtmFTP server’s MDTM-related parameters
    - `server.conf`, configures a mdtmFTP server operation parameters
    - `passfile`, stores a mdtmFTP server’s user/password pairs
- mdtmFTP uses Globus security. In the mdtmFTP container, there is a globus tool folder located at `“/home/globus_tools/”`.
  - Users can use `globus-gridftp-passwd` @ `“/home/globus_tools/”` to generate user/password pairs for `passfile` @ `“/home/mdtmftp_server”`

## Section 3. Configuring and Running mdtmFTP server

### Step 1. Configuring mdtmFTP server CONFIG files

Running mdtmFTP server requires properly configuring two files – `mdtmconfig.xml` and `server.conf`.

- 1) First start a mdtmFTP docker container by running

```
“docker run -ti --net=host docker.io/wenji/mdtm:mdtmFTP”
```

This command will start the mdtmFTP container interactively. You can login the container to edit/configure files.

- 2) In the container, following the instructions in Appendix 1 and 2 to edit `mdtmconfig.xml` and `server.conf`.
- 3) Exit the mdtmFTP container.
- 4) In the host system, run `“docker ps -a”` to find `container_id` for the container that you just exit.
- 5) In the host system, save the container changes by running `“docker commit container_id xxx:yyy”`

Note: xxx is the local repository name, yyy is the tag name for the customized mdtmFTP container.

## Step 2. Managing users for mdtmFTP server in Docker container environment

In standard environment (non-docker environment), when a user transfers files from/to a mdtmFTP server, he must have an account on the system that mdtmFTP runs.

When mdtmFTP runs in Docker container environment, two sets of user account will be involved – `user account @ container` and `user account @ host`. When a user is created into a container, this user may not be known for host machines. At this moment, if a host volume is mounted into this container, there may be “permission denied” issues.

To avoid such “permission denied” issues, we adopt 1-to-1 mapping policy between `user account @ container` and `user account @ host`:

- For each user created in a container, we can set a dedicated `uid`.
- For each group created in a container, we can set a dedicated `gid`.
- On host, we can create a “docker” user with those dedicated `uid/gid`, and manage permission.

Here is an example on how to setup an account in container and in host.

- 1) Start the previously saved mdtmFTP docker container by running

```
“docker run -ti --net=host xxx:yyy”
```

This command will start the mdtmFTP container interactively.

- 2) In the container, add a user `“mdtmftp”` and a group `“mdtmftp-group”`. We will set dedicated `uid/gid`.

```
groupadd mdtmftp-group -g 2000
useradd -u 2000 -d /home/mdtmftp --create-home --shell /bin/bash mdtmftp
usermod -g mdtmftp-group mdtmftp
```

- 3) Exit the mdtmFTP container.
- 4) In the host system, run `“docker ps -a”` to find `container_id` for the container that you just exit.
- 5) In the host system, save the container changes by running `“docker commit container_id xxx:yyy”`
- 6) In the host system, we create user `“mdtmftp”` and group `“mdtmftp-group”` with dedicated `uid/gid`.

```
groupadd mdtmftp-group -g 2000
useradd -u 2000 -d /home/mdtmftp --create-home --shell /bin/bash mdtmftp
usermod -g mdtmftp-group mdtmftp
```



### Step 3. Mounting host folders/directories to the mdtmFTP container

Data transfer requires folders/directories to hold and save data. In Docker container environment, we do not need to create large folders/directories in a container. Instead, we can mount host directories/folders to the container. Because we implement a 1-to-1 mapping policy between user account @ container and user account @ host, we can simply mount host folders/directories to the container.

A user can add a data volume (i.e., folder/directory) to a container using the `-v` flag with the docker `run` command. A user can use the `-v` multiple times to mount multiple data volumes.

For example, the following command will start the mdtmFTP container, and mount host folder `/storage_x` to container folder `/storage_y`

```
docker run -ti -v /storage_x:/storage_y --net=host xxx:yyy
```

Note: xxx:yyy is the mdtmFTP container that is customized to your DTN system.

#### Step 4. Managing passfile for mdtmFTP server

mdtmFTP uses Globus security. In the mdtmFTP container, there is a *globus* tool folder located at `"/home/globus_tools/".`

System admin can use `globus-gridftp-password @ "/home/globus_tools/"` to generate user/password pairs.

Here is an example on how to generate a user/password pair for user `mdtmftp` and add it to `"/home/mdtmftp_server/passfile"` in the container.

- 1) Start the previously saved mdtmFTP docker container by running

```
"docker run -ti --net=host xxx:yyy"
```

This command will start the mdtmFTP container interactively.

- 2) In the container, switch to `mdtmftp` by running `"su mdtmftp"`

- 3) In the container, generate the user/password pair by running

```
"$ /home/globus_tools/globus-gridftp-password >> /home/mdtmftp/temp"
```

- 4) In the container, `"$cat /home/mdtmftp/temp"`

```
mdtmftp:R/BH1rjpOagsk:2000:2000:./home/mdtmftp:/bin/bash
```

- 5) In the container, exit the `"mdtmftp"` session by running `"$exit"`.

- 6) In the container, `"#cat /home/mdtmftp/temp >> /home/mdtmftp_server/passfile"`.

- 7) Exit the mdtmFTP container.

- 8) In the host system, run `"#docker ps -a"` to find `container_id` for the container that you just exit.

- 9) In the host system, save the container changes by running `"#docker commit container_id xxx:yyy"`.

- 10) Send the password to the corresponding user.

Note: if you choose to use GSI certificate security to authenticate users, there is no need to create a password file. You may want to follow the instructions at <http://grid.ncsa.illinois.edu/ssh/> to configure GSI security.



## Step 5. Running a mdtmFTP server in Docker container environment

- *Running a mdtmFTP server as root*

```
docker run -v /storage_x:/storage_y --net=host --privileged --security-opt seccomp:unconfined xxx:yyy /bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf" &
```

This command starts a mdtmFTP server as root.

- `-v /storage_x:/storage_y` mounts host folder `/storage_x` to container folder `/storage_y`.
- `--net=host` uses the host's network stack inside the container.
- `--privileged` runs docker with privilege
- `--security-opt seccomp:unconfined` is required for Docker 1.10 and 1.11 to add capabilities.
- `xxx:yyy` is mdtmFTP container name.
- `/bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf"` executes several commands in the container
  - `cd /home/mdtmftp_server` enters mdtmFTP server working directory
  - `./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf` starts a mdtmFTP server

Please refer to Appendix 3 for mdtmFTP server command syntax

- *Running a mdtmFTP server as non-root*

For docker 1.10 and docker 1.11

```
docker run -u mdtmftp -v /storage_x:/storage_y --cap-add=IPC_LOCK --cap-add=SYS_NICE --cap-add=SYS_ADMIN --cap-add=SYS_RESOURCES --net=host --security-opt seccomp:unconfined xxx:yyy /bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf" &
```

This command starts a mdtmFTP server as non-root.

- `-u mdtmftp` runs the container as user `mdtmftp`
- `-v /storage_x:/storage_y` mounts host folder `/storage_x` to container folder `/storage_y`.
- `--net=host` uses the host's network stack inside the container.
- `--cap-add=IPC_LOCK` adds Linux capability `IPC_LOCK` to the container. mdtmFTP server requires this capability to lock memory when running as non-root.
- `--cap-add=SYS_NICE` adds Linux capability `SYS_NICE` to the container. mdtmFTP server requires this capability to bind I/O threads when running as non-root.
- `--cap-add=SYS_ADMIN` adds Linux capability `SYS_ADMIN` to the container. mdtmFTP server requires this capability to allocate pipes for the splice feature when running as non-root.

- `--cap-add=SYS_RESOURCE` adds Linux capability `SYS_RESOURCE` to the container. mdtmFTP server requires this capability to increase pipe capacity for the splice feature when running as non-root.
- `--security-opt seccomp:unconfined` is required for Docker 1.10 and 1.11 to add capabilities.
- `xxx:yyy` is mdtmFTP container name.
- `/bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf"` executes several commands in the container
  - `cd /home/mdtmftp_server` enters mdtmFTP server working directory
  - `./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf` starts a mdtmFTP server

For docker 1.12+

```
docker run -u mdtmftp -v /storage_x:/storage_y --cap-add=IPC_LOCK --cap-add=SYS_NICE --cap-add=SYS_ADMIN --cap-add=SYS_RESOURCES --net=host xxx:yyy /bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf" &
```

This command starts a mdtmFTP server as non-root.

- `-u mdtmftp` runs the container as user `mdtmftp`
- `-v /storage_x:/storage_y` mounts host folder `/storage_x` to container folder `/storage_y`.
- `--net=host` uses the host's network stack inside the container.
- `--cap-add=IPC_LOCK` adds Linux capability `IPC_LOCK` to the container. mdtmFTP server requires this capability to lock memory when running as non-root.
- `--cap-add=SYS_NICE` adds Linux capability `SYS_NICE` to the container. mdtmFTP server requires this capability to bind I/O threads when running as non-root.
- `--cap-add=SYS_ADMIN` adds Linux capability `SYS_ADMIN` to the container. mdtmFTP server requires this capability to allocate pipes for the splice feature when running as non-root.
- `--cap-add=SYS_RESOURCE` adds Linux capability `SYS_RESOURCE` to the container. mdtmFTP server requires this capability to increase pipe capacity for the splice feature when running as non-root.
- `xxx:yyy` is mdtmFTP container name.
- `/bin/bash -c "cd /home/mdtmftp_server; ./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf"` executes several commands within the container
  - `cd /home/mdtmftp_server` enters mdtmFTP server working directory
  - `./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf` starts a mdtmFTP server

## Section 4. Configuring and Running mdtmFTP client

### Step 1. Configuring mdtmFTP client

Running mdtmFTP client requires properly configuring *mdtmconfig.xml*.

- 1) First start a mdtmFTP docker container by running

```
“docker run -ti --net=host docker.io/wenji/mdtm:mdtmFTP”
```

This command will start the mdtmFTP container interactively. You can login the container to edit/configure files.

- 2) In the container, following the instructions in Appendix 4 to edit *mdtmconfig.xml*.
- 3) Exit the mdtmFTP container.
- 4) In the host system, run “`docker ps -a`” to find *container\_id* for the container that you just exit.
- 5) In the host system, save the container changes by running “`docker commit container_id xxx:yyy`”

Note: xxx is the local repository name, yyy is the tag name for the customized mdtmFTP container.

## Step 2. Managing users for mdtmFTP client in Docker container environment

In standard environment (non-docker environment), when a user transfers files from/to a mdtmFTP server, he must have an account on the system that mdtmFTP runs.

When mdtmFTP runs in Docker container environment, two sets of user account will be involved – user account @ container and user account @ host. When a user is created into a container, this user may not be known for host machines. At this moment, if a host volume is mounted into this container, there may be “permission denied” issues.

To avoid such “permission denied” issues, we adopt 1-to-1 mapping policy between user account @ container and user account @ host:

- For each user created in a container, we can set a dedicated uid
- For each group created in a container, we can set a dedicated gid
- On host, we can create a “docker” user with those dedicated uid/gid, and manage permission.

Here is an example on how to setup an account in container and in host.

- 1) In the host, start the previously saved mdtmFTP docker container by running

```
“docker run -ti --net=host xxx:yyy”
```

This command will start the mdtmFTP container interactively.

- 2) In the container, add a user “*mdtmftp*” and a group “*mdtmftp-group*”. We will set dedicated uid/gid.

```
groupadd mdtmftp-group -g 2000
useradd -u 2000 -d /home/mdtmftp --create-home --shell /bin/bash mdtmftp
usermod -g mdtmftp-group mdtmftp
```

- 3) Exit the mdtmFTP container.
- 4) In the host system, run “*docker ps -a*” to find *container\_id* for the container that you just exit.
- 5) In the host, save the container changes by running “*docker commit container\_id xxx:yyy*”
- 6) In the host, we create user “*mdtmftp*” and group “*mdtmftp-group*” with dedicated uid/gid.

```
groupadd mdtmftp-group -g 2000
useradd -u 2000 -d /home/mdtmftp --create-home --shell /bin/bash mdtmftp
usermod -g mdtmftp-group mdtmftp
```



### Step 3. Mounting host folders/directories to the mdtmFTP container

Data transfer requires folders/directories to hold and save data. In the Docker container environments, we do not need to create large folders/directories within a container. Instead, we can mount host directories/folders to the container. Because we implement a 1-to-1 mapping policy between user account @ container and user account @ host, we can simply mount host folders/directories to the mdtmFTP container.

A user can add a data volume (i.e., folder/directory) to a container using the `-v` flag with the docker run command. A user can use the `-v` multiple times to mount multiple data volumes.

For example, the following command will start the mdtmFTP container, and mount host folder `/storage_x` to container folder `/storage_y`

```
docker run -ti -v /storage_x:/storage_y --net=host xxx:yyy
```

Note: xxx:yyy is the mdtmFTP container that is customized to your DTN system.

## Step 4. Running a mdtmFTP client

a mdtmFTP client can run either as root, or no-root.

1) Start the previously saved mdtmFTP docker container by running

```
docker run -u mdtmftp -ti -v /storage_x:/storage_y --cap-add=IPC_LOCK --cap-add=SYS_NICE --net=host xxx:yyy
```

This command will start the mdtmFTP container interactively

- `-u mdtmftp` runs mdtmFTP client as user `mdtmftp`
- `-v /storage_x:/storage_y` mounts host folder `/storage_x` to container folder `/storage_y`.
- `--net=host` uses the host's network stack inside the container.
- `--cap-add=IPC_LOCK` adds Linux capability `IPC_LOCK` to the container. It will facilitate mdtmFTP client to lock memory to improve performance. This is optional for mdtmFTP client.
- `--cap-add=SYS_NICE` adds Linux capability `SYS_NICE` to the container. It will facilitate mdtmFTP client to bind I/O threads to improve performance. This is optional for mdtmFTP client.
- `xxx:yyy` is mdtmFTP container name.
- `--security-opt seccomp:unconfined` is required for Docker 1.10 and 1.11 to add capabilities.

2) Login the container, enter mdtmFTP client working directory

```
"cd /home/mdtmftp_client"
```

3) In the container, running mdtmFTP client to transfer files. Please refer to Appendix 5 for mdtmFTP client command syntax.

## Section 5. Data Transfer Examples

### 5.1 Client – Server data transfer

#### Step 1: Launch the server on DTN A

```
docker -v /storage_x:/storage_y --net=host xxx:yyy /bin/bash -c "cd /home/mdtmftp_server;  
./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf" &
```

#### Step 2: Launch the client on DTN B

Assuming the mdtmFTP client runs in a container.

#### *Authentication method: user/password*

Assuming user name/password: `mdtmftp/123456`

- Single file data transfer: transfer a single file from DTN A to DTN B

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
ftp://mdtmftp:123456@10.40.130.189:5001/storage/data1/testfiles/100G/file1  
file:///storage/data1/tmp/
```

- Single file data transfer: transfer a single file from DTN B to DTN A

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
file:///storage/data1/tmp/file1  
ftp://mdtmftp:123456@10.40.130.189:5001/storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN A to DTN B

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
ftp://mdtmftp:123456@10.40.130.189:5001/storage/data1/linux-3.18.21/  
file:///storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN B to DTN A

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
file:///storage/data1/tmp/linux-3.18.21/  
ftp://mdtmftp:123456@10.40.130.189:5001/storage/data1/tmp/
```

#### *Authentication method: GSI certificate*

- Single file data transfer: transfer a single file from DTN A to DTN B

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
gsiftp://10.40.130.189:5001/storage/data1/testfiles/100G/file1 file:///storage/data1/tmp/
```

- Single file data transfer: transfer a single file from DTN B to DTN A

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
file:///storage/data1/tmp/file1 gsiftp://10.40.130.189:5001/storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN A to DTN B

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
gsiftp://10.40.130.189:5001/storage/data1/linux-3.18.21/ file:///storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN B to DTN A

```
/home/wenji/mdtmftp/bin/mdtm-ftp-client -p 8  
file:///storage/data1/tmp/linux-3.18.21/ gsiftp://10.40.130.189:5001/storage/data1/tmp/
```

## 5.2 Third party data transfer between two remote DTNs

### Step 1: Launch mdtmFTP server on DTN A

```
docker -v /storage_x:/storage_y --net=host xxx:yyy /bin/bash -c "cd /home/mdtmftp_server;  
./mdtm-ftp-server -data-interface 131.225.2.29 -password-file passfile -p 5001 -c server.conf" &
```

### Step 2: Launch mdtmFTP server on DTN B

```
docker -v /storage_x:/storage_y --net=host xxx:yyy /bin/bash -c "cd /home/mdtmftp_server;  
./mdtm-ftp-server -data-interface 131.225.2.31 -password-file passfile -p 5001 -c server.conf" &
```

### Step 3: Launch the client on DTN C

Assuming the mdtmFTP client runs in a container.

#### *Authentication method: user/password*

Assuming user name/password: `mdtmftp/123456`

- Single file data transfer: transfer a single file from DTN A to DTN B

```
/home/mdtmftp_client/mdtm-ftp-client -p 8 -vb  
ftp://mdtmftp:123456@131.225.2.29:5001/storage/data1/testfiles/100G/file1  
ftp://mdtmftp:123456@131.225.2.31:5001/storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN A to DTN B

```
/home/mdtmftp_client/mdtm-ftp-client -p 8 -vb  
ftp://mdtmftp:123456@131.225.2.29:5001/storage/data1/linux-3.18.21/  
ftp://mdtmftp:123456@131.225.2.31:5001/storage/data1/tmp/
```

#### *Authentication method: GSI certificate*

- Single file data transfer: transfer a single file from DTN A to DTN B

```
/home/mdtmftp_client/mdtm-ftp-client -p 8 -vb  
gsiftp://131.225.2.29:5001/storage/data1/testfiles/100G/file1  
gsiftp://131.225.2.31:5001/storage/data1/tmp/
```

- Folder data transfer: transfer a Linux folder from DTN A to DTN B

```
/home/mdtmftp_client/mdtm-ftp-client -p 8 -vb  
gsiftp://131.225.2.29:5001/storage/data1/linux-3.18.21/  
gsiftp://131.225.2.31:5001/storage/data1/tmp/
```

## Appendix 1 Configuring *mdtmconfig.xml* @ *mdtmFTP* server

*mdtmconfig.xml* configures a *mdtmFTP* server's MDTM-related parameters. It should be located at *mdtmFTP* server's working directory.

*mdtmconfig.xml* consists of four sections: *Topology*, *Online*, *Thread*, and *File* section:

- **Topology section.** The syntax is defined as:

```
<Topology>
  <Device type=Device Type numa=Numa ID>Device Name</device>
  ...
</Topology>
```

Device Type refers to MDTM device type. MDTM defines three types of devices: *network*, *block*, and *virtual*.

- *Network* refers to a network I/O device.
- *Block* refers to a storage/disk I/O device.
- *Virtual* refers to a virtual device, which is defined particularly for *mdtmFTP* server.

Numa ID sets which NUMA node a device belongs to (i.e., NUMA location).

Device Name specifies a device name.

MDTM middleware is typically able to detect physical I/O devices and their locations (i.e., which NUMA node that a I/O device belongs to) on a NUMA system. However, there are two cases that MDTM middleware cannot detect physical I/O devices or their locations correctly:

- In a fully virtualized environment, where information on physical I/O devices is not exposed to guest OS.
- Some vendors' I/O devices may not comply to OS rules to expose device information properly.

Under these conditions, system admin should manually configure I/O devices and their NUMA locations.

*Virtual* device is defined particularly for *mdtmFTP* server to monitor data transfer status. *mdtmFTP* server spawns a dedicated management thread to collect and record data transfer statistics. The management thread is associated with a virtual device, which will be pinned to a specified NUMA node.

- **Online section.** The syntax is defined as:

```
<Online>
  <Device>Device Name</Device>
  ...
```

</Online>

This section specifies the I/O devices that are assigned for data transfer.

For example, assume a DTN has the following I/O devices:

- Ethernet NIC devices
  - eth0 – configured for management access
  - eth1 – configured for WAN data transfer
- Block I/O devices
  - /dev/sda – system disk
  - /dev/sdb – data repository for WAN data transfer

In this case, the online section would be defined as

```
<Online>
  <Device>eth1</Device>
  <Device>sdb</Device>
</Online>
```

- For network I/O devices, a user can run `ifconfig` to list network I/O devices available on the system.
- For storage/disk IO devices, a user can run `lsblk` to list storage/disk I/O devices available on the system; and then run `df` to find out on which storage/disk I/O devices that a data transfer folder will be located.

Assuming, a DTN system's `lsblk` output is:

```
[root@bde1 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                                  8:0  0  1.8T  0 disk
├─sda1                               8:1  0  500M  0 part /boot
├─sda2                               8:2  0  1.8T  0 part
├─scientific_bde1-root              253:0  0   50G  0 lvm /
├─scientific_bde1-swap              253:1  0    4G  0 lvm [SWAP]
└─scientific_bde1-home              253:2  0  1.8T  0 lvm /home
loop0                               7:0  0  100G  0 loop
└─docker-253:0-203522131-pool       253:3  0  100G  0 dm
loop1                               7:1  0    2G  0 loop
└─docker-253:0-203522131-pool       253:3  0  100G  0 dm
nvme0n1                             259:0  0  1.1T  0 disk /data1
```

And the `df` output is:

```
[root@bde1 ~]# df
Filesystem                1K-blocks  Used  Available Use% Mounted on
/dev/mapper/scientific_bde1-root 52403200 15999428 36403772 31% /
devtmpfs                   65855232    0 65855232  0% /dev
/dev/nvme0n1                1153584388 104952744 990009612 10% /data1
/dev/mapper/scientific_bde1-home 1895386900 23602284 1871784616 2% /home
/dev/sda1                   508588    376264    132324 74% /boot
```

If `"/data1"` is used as data transfer folder, the corresponding storage/disk I/O device is `"nvme0n1"`.

- **Thread section.** The syntax is defined as:

```
<Threads threads=Default Num>
  <Device type=Device Type threads=Num>Device Name</Device>
  ...
</Threads>
```

This section defines the number of threads that needs to be allocated for an I/O device. The number of threads allocated for an I/O device should be proportional to the device's I/O bandwidth. The rule of thumb is that a thread can handle an I/O rate of 10Gbps. For example, four threads should be allocated for a 40GE NIC while one thread be allocated for a 10GE NIC.

Default Num sets the default number of threads allocated for each I/O device.

If a different number of threads should be allocated for a particular I/O device, a separate entry for the device should to be specified here.

A virtual device should be allocated with 1 thread.

- **File section.** The syntax is defined as:

```
<File segment=File Size Threshold>
</File>
```

MDTM splits a large file into segments, which are spread to different threads for disk and network operations to increase performance.

File Size Threshold sets a file size threshold. A file with a size that exceeds the threshold will be split into multiple segments, which are spread across I/O threads to be transferred in parallel.



Here is a sample *mdtmconfig.xml* file for mdtmFTP server:

```
<?xml version="1.0" standalone="no" ?>
<Topology>
  <Device type="Virtual" numa="1">man</Device>
  <Device type="Network" numa="0">eth40.4020</Device>
</Topology>
<Online>
  <Device>eth40.4020</Device>
  <Device>sda</Device>
  <Device>man</Device>
</Online>
<Threads threads="1">
  <Device type="Network" threads="2">eth40.4020</Device>
  <Device type="Block" threads="2">sda</Device>
  <Device type="Virtual" threads="1">man</Device>
</Threads>
<File segment="2G">
</File>
```

## ***Appendix 2. Configuring server.conf @ mdtmFTP server***

*server.conf* configures a mdtmFTP server's operation parameters.

- *blocksize* sets the block size for disk I/O operations. The block size should be 4K or multiple of 4k (e.g. 4M).
- *direct* is a flag to enable or disable direct I/O. When direct I/O is enabled, file reads and writes go directly from mdtmFTP to the storage device(s), bypassing the operating system read and write caches. For bulk data transfer, enabling direct I/O would improve performance.
- *splice* is a flag to enable or disable zero-copy by using the Linux *splice* mechanism. Note, *splice* is an experimental feature that may not function well in some systems. You can turn this feature off by setting *splice* to 0.
- *monitor* is a flag to enable or disable MDTM monitoring

Here is a sample *server.conf* file:

```
blocksize 4194304  
direct 1  
splice 0  
monitor 0
```

## Appendix 3 mdtmFTP server command syntax

mdtmFTP server command syntax:

```
mdtm-ftp-server -data-interface <ip_address> -password-file <passwd_file> -p <port_num> -c <server.conf>
```

Command line options:

<code>-data-interface &lt;ip_address&gt;</code>	Specifies a server's IP interface for data transfer
<code>-password-file &lt;passwd_file&gt;</code>	Specifies a password file to authenticate users. You can use Globus tools to create a password file. If GSI certificate security is configured, you need not create a password file to authenticate users.
<code>-p &lt;port_num&gt;</code>	Specifies the port that mdtmFTP server listens on
<code>-c &lt;server.conf&gt;</code>	Specifies a configuration file to set data transfer parameters

## Appendix 4 Configuring *mdtmconfig.xml* @ *mdtmFTP Client*

Running *mdtmFTP* client requires properly configuring *mdtmconfig.xml*, which configures a *mdtmFTP* client's parameters. This file must be put in the working directory.

*mdtmFTP* client's configuration is similar to that of *mdtmFTP* server (Appendix 3), except that *mdtmFTP* client does not need to configure a *virtual* device.

Here is a sample *mdtmconfig.xml* file:

```
<?xml version="1.0" standalone="no" ?>
<Topology>
  <Device type="Network" numa="1">eth40.4012</Device>
</Topology>
<Online>
  <Device>eth40.4012</Device>
  <Device>sda</Device>
</Online>
<Threads threads="1">
  <Device type="Network" threads="2">eth40.4012</Device>
  <Device type="Block" threads="2">sda</Device>
</Threads>
<File segment="10G">
</File>
```

## ***Appendix 5 mdtmFTP client command syntax***

mdtmFTP client command syntax:

```
mdtm-ftp-client -p <parallelism> -splice src_url dst_url
```

Command line options:

<code>-p &lt;parallelism&gt;</code>	Specifies the number of parallel data streams
<code>-splice</code>	Turn on the splice feature
<code>src_url</code>	Specifies the URL of data source
<code>dst_url</code>	Specifies the URL of data destination