Final Progress Report for Project DE-FG02-99ER25378
HARNESS: Heterogeneous Adaptable Reconfigurable Networked Systems
FT-MPI: Fault Tolerant Message Passing Interface

University of Tennessee, Knoxville, TN-37996-3450

Graham E Fagg
fagg@cs.utk.edu

This report is divided into a number of subsections listed as follows:

1. Brief Harness Overview
2. Aims of HARNESS work at UTK
3. Project progress at UTK (including early project benefits)
4. Additional collaboration efforts
5. Overall status of project at UTK
6. Future goals
7. Supported students
8. Publications
9. Additional seminars
10. Appendices

1 Brief HARNESS Overview

Harness was proposed as a system that combined the best of emerging technologies found in current distributed computing research and commercial products into a very flexible, dynamically adaptable framework that could be used by applications to allow them to evolve and better handle their execution environment.

The HARNESS system was designed using the considerable experience from previous projects such as PVM, MPI, IceT and Cumulvs. As such, the system was designed to avoid any of the common problems found with using these current systems, such as no single point of failure, ability to survive machine, node and software failures. Additional features included improved inter-component connectivity, with full support for dynamic down loading of addition components at run-time thus reducing the stress on application developers to build in all the libraries they need in advance.

2 Aims of HARNESS work at UTK

The project aims at UTK are initially to support construction of applications middleware that would allow applications to be developed and used efficiently upon the new system. In HARNESS terms these would be in the form of dynamically down loadable components or plugs-in, that would fit into the HARNESS systems being developed by other institution involved in the project.

DOE Patent Clearance Granted

Mark P. Dvorscak
252-2393
mark.dvorscak@ch.doe.gov
Intellectual Property Law
Operations Office

3-31-04
Date

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

As the overall structure of HARNESS in effect gives rise to a new programming methodology, that most closely resembles distributed object, this would possibly limit initial community acceptance of HARNESS as any applications would have to be re-engineered to take advantage of HARNESS's features. Thus UTK embarked upon implementing the most commonly used programming paradigm for distributed parallel computing in the form of an MPI plug-in for the HARNESS system. In addition to supporting MPI, the UTK plug-in was designed to support fault tolerant applications via modified MPI semantics as the MPI 1.2 specification did not support any failure recovery.

Secondary project aims also included the development of a set of high performance message passing plug-ins that would allow other HARNESS applications, plug-ins and components to utilize current and emerging high performance network protocols and devices, such as AM, FM, GM and VIA.

2.1 MPI Plug-in for HARNESS

The MPI plug-in would allow a subset of MPI applications to be re-linked and ran upon a HARNESS system without modification, and thus make HARNESS accessible simply and directly and to the high performance computing community. This would also include a large number of mathematical, scientific and engineering packages currently available from software repositories such as NetLib.

2.2 FT-MPI Plug-in for HARNESS

While developing the design of MPI for HARNESS it became obvious that, MPI by itself was very static and this restriction was passed on to any MPI application verbatim. HARNESS itself was very dynamic, adaptable and robust, and thus it should be possible to pass these benefits onto an MPI application in a number of different ways depending on the applications nature, not that of the execution environment.

Thus a design was created for a Fault Tolerant version of MPI (FT-MPI). This system would provide an application a dynamically configurable range of methods for dealing with hardware and software failures. Justification for such a system came from previous experience of building long running scalable numeric applications on large computational resources including those of the DOE ASCI project. On such large systems, failures of either hardware or software become more frequent, and constant restarting of applications becomes impractical and highly inefficient greatly reducing total through-put of such systems.

Current semantic of MPI force an application to abort prematurely on a failure of one of its components. FT-MPI was designed so that an application would continue, with a range of possible recovery actions taken.

3 Project Progress at UTK

An initial FT-MPI implementation was completed at UTK during the first phase of the project. It included just over thirty of the most commonly used MPI 1.2 and MPI 2 API calls. The initial release was built upon a deliberately minimal set of generic system functions, so that it could be later easily ported to the yet un-built HARNESS systems themselves. The system functions were tested on the previously proven PVM message passing system during the first year. During year 2, work focused on improving the initial implementation and included a tuned collectives library together with efficient handling of complex user derived data types and buffer management. These are further described in sections 3.1.1 & 3.1.2.

Experiments with commonly used MPI libraries showed that this initial implementation (years 1 and 2), although sufficient for testing new fault tolerant semantics did not cover enough of the MPI API to support the most commonly used MPI libraries. Although most libraries (except PetSC) used only a few API calls, the sets of calls did not always overlap. The solution to this problem was to start implementing the *entire* MPI 1.2 specification during year 3 and beyond. This is nearing completion (as of March03) and a full release of a FT MPI 1.2 implementation is expected for the ACM Super Computing 2003 conference. It is important to note, that this is from the ground up, a new implementation of MPI that due to the need to support fault tolerance is not based on any other MPI implementation (such as MPICH or LAM). This work is described in more detail in section 3.2 and [9][10].

Experiments also revealed that this new complete implementation also needed to support common debugging and profiling tools as well utilities to help users to understand MPI and fault tolerance. The tools support developed are detailed in section 3.3.

Resource handling for use with the Harness system is also shown in section 3.4.3 together with utilities and tool support also developed.

A number of fault tolerant test applications were also developed, these form the main thrust of future work and are described in section 3.5.

3.1 FT-MPI implementation details

3.1.1 Tuned collective communications library

During initial development of FT-MPI particular attention was paid to implementing collective communications as a special class of communications rather than something just built on optimized point-to-point (p2p) communication. This led to the development of a self tuning collective communications library that can provides optimal performance on any given MPP/cluster system via micro benchmarking a number of possible implementations and then selecting the fastest.

This code was developed in two distinct directions. The first is for use in FT-MPI and is concerned with fault detection and handling within optimized group communication. The second is using the library with any existing MPI implementation to improve the implementations collective performance. So far this second mode of use has given some surprising results, such as better performance than the 'tuned' IBM native MPI implementation on IBM SP systems, shown in Figure 1. Further details are given in [3][5].

Figure 1. FT-MPI automatically tuned collectives compared to vendor MPI implementation on an IBM SP2 system.

3.1.2 Buffer and Data type management

The MPI specification allows for complex messages made from recursive strongly typed data structures to be sent from one process to another. Many MPI implementations only handle contiguous blocks of data efficiently rather than these new user Derived Data Types (DDTs). We developed a library that handles these data types efficiently by using a number of advanced buffer management functions that dynamically compress data, perform scatter/gather DMA and avoids un-need memory copies. The library has like the collective communications library, has been implemented in a standalone non FT-MPI version that is useable by other MPI implementations. The performance of our library on a real applications data structure provided to us by the University of Stuttgart showed a 680% increase in performance over MPICH-1. Additional details are given in [1][2][4].

3.2 FT-MPI implementation of MPI 1.2, testing and performance

As previously stated the initial version of FT-MPI supported new fault tolerant semantics and a limited number of MPI 1.2 API calls. Tests using common numeric libraries such

as Scalapack (via BLACS) showed that a fuller set of MPI calls was required. The decision was made to implement the entire MPI 1.2 specification. This required a complete rewrite of the previous version to support additional functionality required by the complete specification.

The first full release of FT-MPI was completed spring 2003 and the first public release is expected late summer 2003. It offers the user all functions defined in the MPI 1.2 document as well as several sections of the MPI 2 specifications including language interoperation, external interfaces, new error handlers and the MPI-2 attribute caching functions. In all, one hundred and fifty MPI calls are supported. The distribution includes the tuned collectives and derived data types previously developed.

Testing of the new implementation has been via community developed and accepted test suites including: IBM, Intel, PACX, MPICH and BLACS.

The implementation has also focused on performance. Currently raw message performance has been measured using the MPPTTEST suite, and application performance measured with High Performance Linpack (HPL). Results from MPPTTEST are given in figure 2.

Figure 2. Point-2-point MPPTTEST measured bandwidth (on gigabit Ethernet)

The performance of FT-MPI has generally been better than both LAM 6.5 and MPICH 1.2.5, and is slightly slower compared to the newest release of MPICH 2 beta for small messages and faster for larger messages. FT-MPI is also quicker than the current version of MPICH-G2 in a LAN environment as shown in figure 3.

Figure 3. Point-2-point benchmark of FT-MPI versus MPICH-G2 under Globus 2.0 inter-cluster over normal Ethernet

The FT-MPI system has also been ported to a number of different architectures including LINUX, SUN Solaris, & SGI IRIX. The system also supports heterogeneous execution of applications between these different platforms. Currently under test is IBM AIX and Tru64 compatibility, as well as interoperation between 32bit and 64 bit architectures simultaneously. Additionally FT-MPI has been ported to work with GNU, vendor and Intel compilers.

3.3 FT-MPI tools and utilities

For any new MPI implementation to be accepted there has to be sufficient end-user tool support to ease usage. This has been achieved by both creating several new tools supporting the existing end-user MPI environments as well as fault-tolerant usage.

To monitor the state of the application in case a failure occurs, a *com-monitor* has been developed and is available with the distribution as shown in figure 4 during a complete recovery of a heat-flow application. A similar tool [2][4] was previously developed but was restricted to the X windowing environment. This new tool was developed using the OpenGL environment and works under both Unix and the MS Windows systems.

Figure 4. End-user comminfo tool for monitoring MPI applications during real-time failure-recovery cycles.

Among the tools, which end-users are familiar with from their regular working environment is the tracing library MPE which is part of the MPICH project. This has been ported to work with FT-MPI. Using MPE, the user can generate trace-files of its applications running with FT-MPI, which then can be analyzed using tools such as the latest version of JUMPSHOT (version 4), as shown in figure 5.

Figure 5. JUMPSHOT (v4) trace of HPL executing under FT-MPI, using the MPE profiling library.

3.4 System support for FT-MPI

As the HARNESS systems and FT-MPI were designed in parallel, no assumptions were made on which core HARNESS services would be developed. FT-MPI was created to use a small set of generic services that could later be built into these systems. The services as is common with the other HARNESS efforts were constructed as dynamic code in the form of downloadable plug-ins.

3.4.1 Harness Core issues

During year 2 although much work was put into allowing the C based FT-MPI code use within the Emory developed Java based Harness core (see 4.2), it became apparent that FT-MPI is more suited to a C based Harness core. As a core was not available from ORNL during early year 2, a basic C based Harness core was developed at UTK known as the *g_hcore* (Generalized Harness Core). This completely replaced the previous use of PVM in year 1 as the portable systems services layer for the FT-MPI runtime.

The HARNESS core was enhanced for both performance and to allow for several advanced plug-in management features such as:

- (a) Multi-process (services) and multi-thread invocation of plug-ins.
- (b) Multi-method invocation. This also includes a lightweight UDP protocol, the standard TCP methods and in year 3, extensions to allow the use of SOAP.
- (c) Dynamic downloading of requested plug-ins via HTTP from easy to manage web repositories. This allowed ease of plug-in management as well as simplified distribution.
- (d) Plug-in caching. This reduced network bandwidth requirements.
- (e) Plug-in security. Plug-ins are combined with version information into a MIME message that is then signed using openly available public key cryptography technology. This ensured that plug-ins are not tampered with and improves user confidence in third party code.

The ability of HARNESS to invoke services using SOAP via the xSOAP package was an important step as it allows for the GHCORE Harness system in future to integrate/interoperate natively with the web services based Globus 3 infrastructure.

3.4.2 FT-MPI plug-ins

A number of plug-ins for FT-MPI were developed that include:

- (a) Spawn and Notify service. This plug-in allows remote processes to be initiated and then monitored. The service notifying other interested processes when a failure or exit of the invoked process occurs.

- (b) Naming services. These allocate unique identifiers in a distributed environment.
- (c) Distributed Replicated Database (DRD). This service allows for system state and additional MetaData to be distributed, with replication specified at the record level. This plug-in has a secondary benefit as it can be used by other PVM plug-ins to implement the PVM 3.4 Mailbox feature directly.

The plug-ins had additional features added, such as multiple parallel asynchronous startup (spawning), which had greatly reduced startup time of larger jobs. In addition the startup daemons now handle the redirection of IO correctly so that the user can selectively redirect the output to different targets dynamically during runtime for simplified monitoring and debugging of complex MPI jobs.

The overall design of the system is shown below in figure 6.

Figure 6. Overall FT-MPI HARNESS services design.

3.4.3 Support for using resources

FT-MPI jobs run on top of HARNESS controlled resources. These resources make a distributed virtual machine (DVM) much like a parallel virtual machine (as in PVM). To aid users coming from other environments, we have implemented both a command line and a graphical utility for managing the DVM and MPI jobs.

The command line utility is in the form of a simple user friendly *console*. This console allows the user the ability to dynamically add/remove hosts from the DVM, start or kill jobs etc. In the case of large parallel jobs, all processes are additionally given a single identifier known as a RUNID. This can be used in the console to control the complete job no matter how many processes or hosts it involves with a single command. Appendix A.1 lists the possible commands available.

Additionally a graphical front end has been produced that interfaces with the console to aid handling of very jobs. This new tool is shown in figure 7. The implementation is in 'C++' and Java and again works on both UNIX and MS Windows.

Figure 7. Graphical front-end to the HARNESS DVM console. Configuration of the complete system can be changed with just a couple of mouse clicks.

3.5 Applications and Fault tolerant examples

An initial real world MPI application was chosen to test the fault tolerant semantics, and how they could be retro-fitted into existing applications. The application used was the Parallel Spectral Transform Shallow Water Model (PSTSWM), a fully featured climate model that was developed as part of the DOE CHAMMP project. This application also had very well known performance characteristics, and was used to compare the

performance of the initial FT-MPI implementation verses ANLs MPICH implementation in year 1. Some of these results are to be shown at LACSI2003, and have also been included in the appendices. Current experiments with using FT-MPI for fault tolerant applications focus around 'in memory' check-pointing used together with FT-MPI automatic restarting (known as re-spawning) of failed processes. Tests to date with an explicit CG-solver have shown good results. For small to medium test cases the end-to-end execution time, including recovery time for a single failure and restart has been comparable to total execution time without failure under MPICH 1.2.5. This indicates that the overheads of using this fault tolerant implementation are not excessive as is the case with some other implementations using message logging for example. A number of simple examples are being included with the distribution.

MPI correctness and application compatibility testing consists of FT-MPI passing the BLACS, Scalapack and MPICH test suites as well as working correctly with the MPPTTEST and HPL packages. Current work in this involves testing and tuning FT-MPI for use with the Petsc numeric library.

4 Additional collaboration efforts

4.1 Extended use of the UTK HARNESS Core

The underlying services for UTKs version of HARNESS are also capable of acting as a distributed RPC system that can implement a problem solving environment (PSE). In collaboration with researchers from the Vienna General Hospital, Austria, a numeric simulation was ported to HARNESS as a plug-in.

The simulation based on the PEBBLES numeric library was used to calculate electric fields for stimulation of severed nerves in paraplegic crash victims so that through electro stimulation they could walk again. The front end of the simulation was the MATLAB package, which through RPC style calls would invoke the PEBBLES plug-in to perform parallel calculations. The HARNESS system was used due to its robustness and low overheads. Work at the hospital is continuing towards a semi-production system.

4.2 Use of FT-MPI plug-in with the Emory and ORNL HARNESS Cores

The FT-MPI plug-in was designed to work effectively on any of the HARNESS cores. This is achieved by a dual build system where construction of all FT-MPI plug-in services can be either built as UTK format plug-ins or as self contained daemons for use on other Cores. When the plug-ins are build as daemons, the other HARNESS cores only to have execute them, using fork-exec under ORNL core, or a 'system.load' under the Emory Java core. Once started the daemon based plug-ins then interoperate within themselves. An external interface is available for other cores to notify the FT-MPI services if new resources become available so that the FT-MPI scheduler can then utilize these resources. Some work was started on providing a common means of handling plug-ins the same way across any of the three systems, but this has yet to be completed.

Meanwhile to assist in this, the UTK has had a SOAP based interface added using XML descriptions of some of the FT-MPI plug-ins based on the GSOAP library. Thus some of the FT-MPI plug-ins can be accessed via SOAP calls, which are portable across all other HARNESS cores, and especially removes any problems previously encountered ('C' based library calling a Java HCore calling a 'C' based FT-MPI plug-in) when interoperating with the Emory Java based system. This work is ongoing.

4.3 Application and system demonstrations

The Java DVM core was demonstrated to members of the Computer Science and Meteorology departments at Reading University, UK, in October 2000 by Graham Fagg from UTK. A stable release was installed upon a large cluster (100 nodes) of Linux PCs, MS Window Laptops, Sun Sparc and SGI Irix workstations and used to demonstrate Harness application and plug-in design. The British university has since been using Harness on this heterogeneous cluster to develop monte-carlo based modeling libraries.

A hands on demonstration of the Java DVM was also given at the EuroPVM-MPI meeting in Balatonfured Hungary in September 2000. The demonstration included downloading of the package, its installation, and running applications.

The Java DVM was also demonstrated at SC2000 in Dallas, Texas in the ICL/UTK booth.

FT-MPI based on the UTK core was demonstrated with a fault tolerant heat flow code at SC2002.

5 Overall Status and Achievements of project at UTK

- (a) Designed an advanced fault tolerance model for the MPI standard that supported more than just checkpoint recovery
- (b) Initial PVM based version of FT-MPI (approx. 40 MPI functions)
- (c) Built graphical debugging and monitoring tools for FT-MPI applications (X windowing system).
- (d) Built a tuned collective communications library.
- (e) Built derived data type buffer management library.
- (f) Developed a C based Harness core (g_hcore).
- (g) Implemented dynamic downloading of plug-ins with caching.
- (h) Implemented signed plug-in checking.
- (i) Constructed additional plug-in service components such as a Distributed Replicated Database and process management.
- (j) Jointly developed with Emory a C interface to the Java DVM system.
- (k) Demonstrations of the JAVA DVM in both Hungary and the UK.
- (l) Demonstrations of FT-MPI and graphical debugging at SuperComputing 2000.
- (m) Designed and implemented a multi-phase fault tolerant dynamic master stop-fail-recovery model.

- (n) Developed complete MPI 1.2 specification implementation *with* fault tolerance
- (o) Added limited MPI-2 support
- (p) Verified using MPICH, IBM, INTEL, PACX, BLACS test suites
- (q) Ported MPE profiling library to support FT-MPI use with tools such as JUMPSHOT.
- (r) Updated monitoring tools for FT-MPI application state to use openGL, ported to both Unix and MS Windows.
- (s) Updated C based Harness core (GHCORE) and services for faster process management
- (t) Improved speed and flexibility of GHCOREs and validated performance with an external numeric solver PEBBLES
- (u) Created command line console to simplify DVM management
- (v) Created Java DVM management tool
- (w) Demonstrations of real time FT-MPI heat flow application with failures at SuperComputing 2002.

6 Future Goals

As the system nears the level of an initial production style release together with tools and interoperation with third party pre-existing MPI tools, work will shift from the FT-MPI system to developing algorithm classes suitable for moving fault tolerance into more common numeric libraries. Although developing a complete FT-scalapack or FT-PetSC is beyond the scope of this project in the near future, some algorithm classes can be made fault tolerant. Using the lessons from the current work with a CG-solver it is hoped that more sophisticated libraries can be developed and released into the hands of user community in a format that is close to a drop in replacement for non fault tolerant libraries. Additional short term sub-goals also include a native windows version of FT-MPI as well as support for SSL/SSH/X509 certificate security models.

7 Students Supported

1. Antonin Bukovsky: Derived Data Type (DDT) buffer management and runtime optimization.
2. Mickael Melki: Distributed Replicated Database (DRD). Exchange Student from ENS Lyon, France.
3. Sathish Vadhiyar: Collective communication library.
4. Thara Angskun: DVM console

8 Publications

1. G.E.Fagg, "HARNESSing FT-MPI", Proc. of 3rd MetaComputing Workshop, HLRS Stuttgart, Germany, June 20, 2000.
2. Graham E. Fagg & Jack J. Dongarra, "FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World", Proc. of 7th European PVM/MPI Users' Group Meeting, Springer Verlag, LNCS Vol 1908, pp346-353, September 2000.
3. Graham E. Fagg, Sathish S. Vadhiyar and Jack J. Dongarra, "ACCT : Automatic Collective Communications Tuning", Proc. of 7th European PVM/MPI Users' Group Meeting, Springer Verlag, LNCS Vol 1908, pp354-361, September 2000.
4. Graham E. Fagg and Jack J. Dongarra, "Harness and Fault Tolerant MPI", Proc. of Workshop on Clusters and Computational Grids for Scientific Computing, Lyon, Sep 2000.
5. Sathish S. Vadhiyar, Graham E. Fagg, Jack Dongarra, "Automatically Tuned Collective Communications", Proc. of SuperComputer 2000, Dallas Texas, November 2000.
6. Graham E Fagg, Antonin Bukovsky, Jack J Dongarra, "HARNESS and fault tolerant MPI", Parallel Computing (27), pp 1479-1495, 2001.
7. Graham E Fagg, Antonin Bukovsky, Sathish Vadhiyar and Jack J Dongarra, "Fault Tolerant MPI for the HARNESS MetaComputing System", Proc 1st International Conference on Computational Science, Springer Verlag, LNCS2073, pp 355-366, San Francisco May 2001.
8. Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra, "Performance Modeling for Self Adapting Collective Communications for MPI", LACSI Symposium 2001, October 15-18, Eldorado Hotel, Santa Fe, NM.
9. Graham E Fagg and Jack J Dongarra, "HARNESS Fault Tolerant MPI Design, Usage and Performance Issues", Future Generations Computer Systems (18), pp 1127-1142, Elsevier Science, 2002.
10. Edgar Gabriel, Graham E Fagg, Antonin Bukovsky, Thara Angskun, and Jack J Dongarra, "A Fault-Tolerant Communication Library for Grid Environments", 17th Annual ACM International Conference on Supercomputing (ICS'03) International Workshop on Grid Computing and e-Science, June 21, 2003, San Francisco.
11. Edgar Gabriel, Graham Fagg, and Jack Dongarra, "Evaluating The Performance Of MPI-2 Dynamic Communicators And One-Sided Communication", accepted Euro PVM/MPI Meeting, Venice, Italy, September 2003.

9 Additional Seminars (not covered by conference or workshop publications)

"Extending MPI : fault-tolerance and MPI-2 issues", Invited Talk, 6th MetaComputing and Grid Workshop, High Performance Computing Center (HLRS), Stuttgart, Germany, May 2003.

10 Appendices

A.1. Console commands for DVM control.

```
con> help
Command  Description
=====
add      Add hosts to virtual machine
conf     List hosts in virtual machine
delete   Delete hosts from virtual machine
exit     Exit from console
expire   Expire process information
help     Display help message
kill     Terminate process(es)
ps       Display process(es) status
quit     Quit from console
sig      Send signal to process(es)
spawn    Spawn process(es)
vmname   Set or print the current virtual machine name
```

A.2. Example Console output for adding resources and executing an FT-MPI job

```
boba111> console
con> conf
No host available in "dvm", Please make sure that ftmpi_notifier is running before add new hosts
con> add boba111 boba211
con> conf
Found 2 hosts
  HostID          HOST PORT
  0    boba211.sinrg.cs.utk.edu 22501
  1    boba111.sinrg.cs.utk.edu 22501
con> spawn -np 2 -o -mpi bmttest
ftmpirun: RUNID for this application is now 0x5009
```

```
MPI_Init [0]
leader: doing a handshake then doing benchmark
[4102:stdout] size 0 bytes  time 0.000046 Sec  bw 0.000000 MB/Sec
size 1 bytes  time 0.000052 Sec  bw 0.018271 MB/Sec
size 2 bytes  time 0.000052 Sec  bw 0.036471 MB/Sec
size 4 bytes  time 0.000054 Sec  bw 0.071301 MB/Sec
size 8 bytes  time 0.000052 Sec  bw 0.147296 MB/Sec
size 16 bytes time 0.000052 Sec  bw 0.293983 MB/Sec
size 32 bytes time 0.000052 Sec  bw 0.585143 MB/Sec
size 64 bytes time 0.000052 Sec  bw 1.165889 MB/Sec
size 128 bytes time 0.000053 Sec  bw 2.312034 MB/Sec
size 256 bytes time 0.000053 Sec  bw 4.597598 MB/Sec
size 512 bytes time 0.000068 Sec  bw 7.175265 MB/Sec
size 1024 bytes time 0.000087 Sec  bw 11.172941 MB/Sec
size 2048 bytes time 0.000110 Sec  bw 17.811600 MB/Sec
size 4096 bytes time 0.000137 Sec  bw 28.584638 MB/Sec
size 8192 bytes time 0.000205 Sec  bw 38.072443 MB/Sec
size 16384 bytes time 0.000312 Sec  bw 50.047538 MB/Sec
size 32768 bytes time 0.000557 Sec  bw 56.074183 MB/Sec
size 65536 bytes time 0.001038 Sec  bw 60.238062 MB/Sec
size 131072 bytes time 0.001979 Sec  bw 63.171225 MB/Sec
size 262144 bytes time 0.003890 Sec  bw 64.269785 MB/Sec
size 524288 bytes time 0.007840 Sec  bw 63.773073 MB/Sec
size 1048576 bytes time 0.015560 Sec  bw 64.267347 MB/Sec
[4103:stdout] t 1055347013.618629 t2 1055347014.619802 diff 1.001173
follower getting handshake
```

con> ps

ProcID	RunID	HostID	Command	Comment	Status	ElapsedTime..
4102	20489	0	bmtest	FTMPI:proces	exited(val:0)	7s
4103	20489	1	bmtest	FTMPI:proces	exited(val:0)	6s

con> delete 0 1

con> conf

No host available in "dvm", Please make sure that ftpi_notifier is running before add new hosts

con>