Exploiting Overlay Networks Features to Enhance the Performance of the File Mover

Cosimo Anglano, Massimo Canonico

Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria (Italy)

email:{cosimo.anglano,massimo.canonico}@unipmn.it

Abstract

The *File Mover* [6] is a file transfer infrastructure, based on the overlay networks paradigm, specifically conceived to provide high-performance file transfers for Data Grids. In its current implementation, the File Mover exploits only in part the potential benefits typical of overlay networks. In this paper we consider three possible extensions of the File Mover, aimed at increasing its performance, that better exploit the characteristics of overlay networks. The performance improvements obtained with these extensions are demonstrated by means of a set of experiments, in which the performance obtained by an extended version of the File Mover have been compared with those attained by its standard version.

1 Introduction

A rather large set of scientific disciplines, such as High-Energy Physics, Climate Modeling, and Life Sciences, requires the archival, retrieval, and analysis of massive data collections, whose size may well be in the order of tens to hundreds of Megabytes (and sometimes even Petabytes) [18]. *Data Grids* [28], providing infrastructure and services for distributed data-intensive applications, are at the moment regarded as the best solution to the processing needs arising in the above application domains. As observed in [9], one of the crucial components of Data Grids is its file transfer infrastructure, that must be conceived in such a way to reduce as much as possible the time taken to transfer a file among the resources in the Grid. The need for high-performance data transfers arises from the observation that the completion time of typical data-intensive applications is given by the sum of their execution time and of the time taken to transfer the data they need [23], and is often dominated by the data transfer time.

The *File Mover* [6] is a file transfer infrastructure specifically conceived for Data Grids that use production networks (e.g., the Internet) to connect their resources. The main characteristics of such networks are (a) lack of control, that is it is not possible to enforce any application-specific control decision (e.g. choosing the network path used to perform a transfer) on the network, and (b) presence of cross traffic, originated by other applications, that competes for the bandwidth with the traffic generated by the Data Grid applications and components of its infrastructure.

The File Mover is based on the *overlay network* [11, 12] paradigm, in which a set of machines (called *File Relays*) connected to the edges of the network are used to build a virtual topology layered on top of the physical one. File Relays communicate among them by using direct transport-level connections, and use application-level routing to overcome the inherent limitations of IP routing, namely (a) its inability to exploit alternative network paths that often exhibit better performance than those chosen to route traffic [24, 25, 27], and (b) the slow convergence time [15] of routing protocols that results in very long times to recover routing tables after a network failure [10, 19, 20]. More precisely, the transfer of a file from a source to a destination machine exploits a sequence of File Relays (that we call *virtual paths*) to force the transfer to use a network path better than the one chosen by IP routing.

As shown in [6], the File Mover is able to achieve significantly better performance than state-of-the-art transfer solutions based on IP routing. However, its performance improvements have been obtained by exploiting only in part the potential benefits typical of overlay networks. In particular, the first version of the File Mover exploits only a single virtual path between the source and the destination machines, although several virtual paths may exist among them. In this paper we study a set of possible optimizations, that may be introduced in the File Mover design in order to further increase its performance, that are based on the idea of properly exploiting all the virtual paths existing between the source and destination machines, and the storage resources available on the various machines of the overlay. In particular, we consider *multipath file transfers*, usage of *file caches* as transfer sources, and *striped file transfers* as possible means to increase file transfer performance. In order to evaluate the performance benefits yield by these techniques, we have implemented a prototype version of the File Mover that includes them, and performed a set of experiments aimed at comparing its performance against those attained by its "plain" implementation. The results of these experiments, performed on network testbed built to accurately reproduce the operational conditions of realistic production networks (like the Internet) both in terms of path capacities and traffic flows, clearly indicate that the proposed techniques have the potential of greatly enhancing the performance attainable by the File Mover.

The remainder of this paper is organized as follows. In Section 2, we briefly review the architecture of the File Mover, its behavior and operations. In Section 3 we describe the various extensions we have introduced in the File Mover. Section 4 reports the results obtained in our experiments. Finally, Section 5 concludes the paper and outlines future research work.

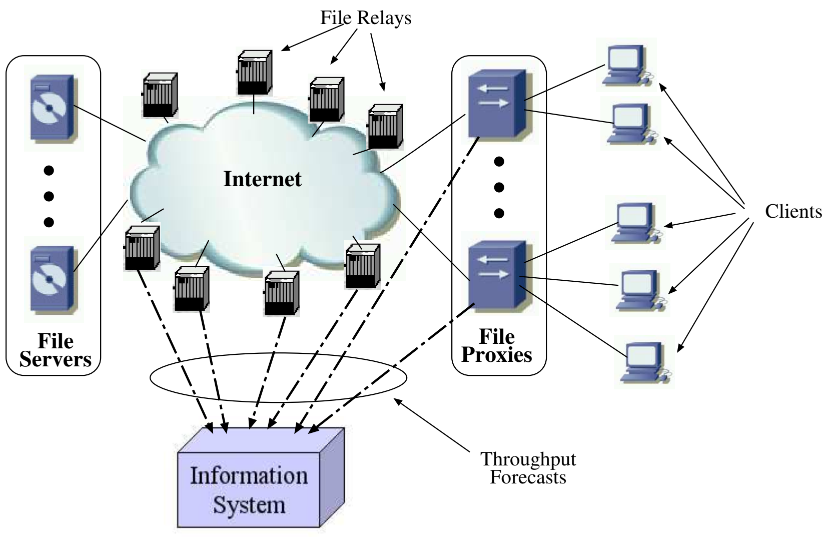

Figure 1: The File Mover architecture

2 The File Mover

Let us briefly describe the main features of the File Mover, whose complete description, as well as a thorough evaluation of its performance, may be found in [6]. The File Mover is a system providing a high-performance transfer service for (potentially very) large files. Its architecture, depicted in Fig. 1, features an overlay network – comprising a set of *File Relays* – used to carry out the transfer between a *File Server* and a *Client*. The overlay network is fully connected, that is each File Relay may communicate directly with any other File Relay via transport-level connections (henceforth denoted as *virtual links*). *File Proxies* represent the interface between clients and the File Mover overlay network: when a client needs a file, it contacts one of the File Proxies, that in turn requests the file to a server holding one of its copies, and notifies the requesting client when the transfer has been completed. [1]

Upon receiving a request, a Server computes (by means of a suitable algorithm [6]) the best *virtual path*, consisting in the sequence of virtual links (and of the corresponding File Relays) that connect itself with the requesting Proxy that provides the highest achievable throughput among all the possible virtual paths connecting them. The achievable throughput $AT(VP_i)$ on a given virtual path VP_i that includes a set L of virtual links and a set R of relays is defined as:

$$AT(VP_i) = \min_{x \in L \cup R} (Thr(x)) \tag{1}$$

where $Thr(x)$ denotes either the throughput achievable on virtual link x, or the *traversal throughput* of Relay x (i.e. the rate at which data can be moved from an input to an output virtual link). By including the traversal throughput of

[1] We assume that the identity of the Server holding a copy of the requested file is determined by the Client by means of a data location service external to the File Mover, as for instance the *Replica Location Service* [8].

File Relays, Eq.(1) takes into account also the slowdown due to the overhead introduced by data forwarding. As an example, consider the File Mover configuration reported in Fig. 2, representing a scenario in which three File Relays ($R1$, $R2$, and $R3$) are available to transfer a file between the *Server* and the *Proxy*, where circles correspond to Relays and are labeled with the corresponding traversal throughput value, while arcs correspond to virtual link and are labeled with the achievable throughput bandwidth on the corresponding network path. In this case, the throughput achievable

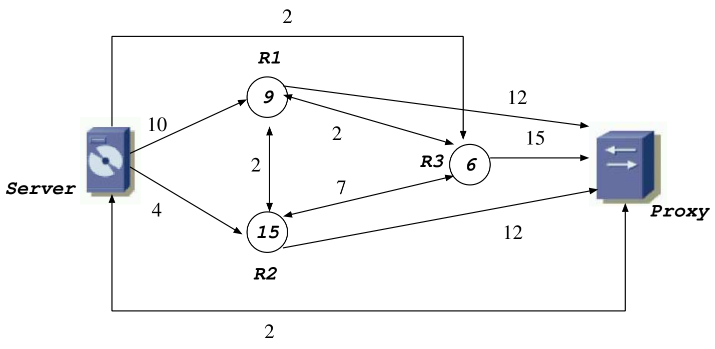

Figure 2: File Mover configuration. Throughput values are expressed in Mbps.

on the virtual path traversing relays $R1$ and $R3$ is given by $\min\{10, 9, 2, 6, 15\} = 2$ Mbps, while the highest throughput (9 Mbps) corresponds to the virtual path traversing relay $R1$ only. In order to maximize the throughput on individual virtual links (and, hence, on the virtual path they compose), the File Mover uses the *UDP-based Data Transfer* (*UDT*) protocol [14] to move data among pairs of overlay nodes. Furthermore, *cut-through* is used to transfer files, that is the transferred file is divided into fixed-size blocks that are forwarded independently from each other.

The File Mover has been conceived to be used in production networks, where the presence of cross traffic due to other network applications can introduce potentially very large variations in the bandwidth available on individual virtual links, that result in proportional variations in the achievable throughput. This implies that a virtual path that at a given time provides the highest throughput may become less effective than another one that exhibits a lower throughput at the moment of virtual path computation while the transfer is still going on. In order to minimize the impact of these changes in cross traffic, the File Mover computes mid-term forecasts of the throughput achievable on individual virtual links, and uses them to compute the virtual path characterized by the highest expected achievable throughput. These forecasts are computed by measuring (by means of *iperf* [1]), at regular time instants, the achievable throughput of each virtual link, and by feeding these measurements to the statistical forecasting algorithms of the *Network Weather Service* (NWS) [30]. These forecasts are then stored into the *Information System* for future use. Measurements are carried out on an individual basis, that is every Server and Relay periodically measures the available bandwidth for all its outgoing virtual link (special care is taken to avoid interference among concurrent measurements).

3 Extending the File Mover

As mentioned in the Introduction, although the File Mover architecture discussed in the previous section results in significant performance improvements over source-to-destination direct transfers [6], further performance improvements can be obtained by exploiting several virtual paths to transfer a single file. As a matter of fact, there are situations in which the number of available (i.e., not engaged in any file transfer) File Relays allows the set up of multiple virtual paths directed towards the same Proxy that can be used to transfer different parts of the same file. Furthermore, the possibility of storing files (or part of them) at individual Relays enables one to transfer a file either from a location different from the Server holding the file, or from multiple locations simultaneously. In the remainder of this section we discuss three optimizations that result from the above observations, namely *multipath transfers*, *transfer from cached copies*, and *striped transfers*.

3.1 Multipath File Transfers

The first optimization we consider consists in performing *multipath transfers* of the requested file, where the server splits the file into independent chunks that are sent along different virtual paths connecting the Server to the requesting Proxy. As a matter of fact, it is often the case that, after the best virtual path has been chosen, the remaining set of File Proxies can be used to set up other virtual paths characterized by lower values of the expected throughput. The idea behind multipath transfers is to use these lower-performance virtual path – in addition to the best one – so that the total transfer time can be further reduced. In order to balance the work that has to be done by the various virtual paths, the size of the chunk sent on each of them is proportional to its achievable throughput. More specifically, the size $S(C_i)$ of the chunk sent along the i^{th} virtual path VP_i is computed as:

$$S(C_i) = file\_size * \frac{AT(VP_i)}{\sum_{j=1}^{n} AT(VP_j)} \tag{2}$$

where n denotes the number of different virtual paths available at the moment of the request, $file\_size$ denotes the size (in bytes) of the requested file, and $AT(VP_i)$ is the expected value of the achievable throughput for virtual path VP_i (defined as in Eq. (1)). For example, to transfer a 1 GB file over two distinct virtual paths VP_1 and VP_2 such that $AT(VP_1) = 40$ Mbps and $AT(VP_2) = 60$ Mbps, $S(C_1) = 1GB * 40/(40 + 60) = 0.4GB$ and $S(C_2) = 1GB * 60/(40 + 60) = 0.6GB$.

To perform a multipath transfer, the Server computes first the best virtual path, and then proceeds with the computation of the next one that does not include the Relays already used for the first one, and so on until no more available

Relays exist. Note that the Server-Proxy direct virtual link is a particular case of virtual path consisting in a single link and no intermediate Relays, and as such may be used for multipath transfers.

3.2 Transfer from Cached Copies

The next optimization we consider is based on the observation that, being File Relays standard computers equipped with hard disks, temporary copies of the file being transferred may be stored locally on all the Relays belonging to the chosen virtual path. Subsequent transfers for the same file may be then performed using the Relays that results in the best virtual path towards the requesting Proxy, rather than from the origin server, if this exhibits a lower achievable throughput.

The introduction of cached replicas requires a change in the way transfers are carried out. As a matter of fact, the original File Mover design assumes that at any given point in time only a single instance of each file is present, so that only the corresponding Server has to be contacted in order to request the transfer. However, the introduction of cached copies implies that the number of potential sources to which a transfer may be requested is larger than one, so the following three problems arise: (a) the various existing replicas must be located, (b) consistency among copies must be maintained if and when the original file is modified, and (c) the replica to be used as the source must be selected. The first two problems are solved by introducing into the File Mover architecture a *Replica Location and Management* (*RLM*) subsystem, in charge of keeping track of the various existing replicas, and of maintaining consistency among them by means of a simple write-invalidate protocol that – upon writing of the original file – invalidates all the existing copies. It is worth to point out that systems of this type have been already designed and developed for Grid infrastructures [8], so one of them may be exploited for the File Mover. The third problem, namely replica selection, may be solved by having either the Server or the requesting Proxy (a) query the RLM subsystem to find all the available replicas, (b) run the virtual path computation algorithm for each of them, and (c) choose the replica that results in the virtual path exhibiting the highest expected achievable throughput. Although this selection might be performed by either the requesting Proxy or by the Server, our choice is to assign this functionality to the requesting Proxy, since this allows a simpler integration into the File Mover architecture of the optimization discussed in the next subsection.

As a final consideration, it should be noted that caching may be combined with multipath transfers, that is the Relay holding the replica that will be transferred may set up multiple virtual paths towards the requesting Proxy and transfer different chunks of the file across them, as discussed in the previous section. In this paper, however, we do not consider this option, that is left as part of our future work.

3.3 Striped File Transfers

The presence of multiple replicas of the same file enables another optimization, namely the usage of *striped transfers* in which different chunks of the same file are simultaneously transmitted *from different replicas*. The size of the chunk sent along a given virtual path is computed according to Eq. (2). One important potential advantage of striped transfers with respect to multipath ones (as defined in Sec. 3.1) is that, by using separate sources for the various file chunks, the transmission of each chunk may exploit the full bandwidth available on the link connecting the source to the Internet, so potential bottlenecks, that may arise when a single source is used, are removed. As a final consideration, note that although it is possible (and probably profitable) to combine multipath and striped transfers, in which the file chunk sent by a given cache can be further split across several virtual paths connecting the same source-Proxy pair, in this paper we do not consider this option, that is left as part of our future work.

4 Experimental Evaluation

In order to evaluate the performance benefits of the three optimization techniques described before, we developed a prototype implementation of the File Mover that included them, and used it to carry out a set of experiments aimed at comparing its performance with those attained by the standard File Mover implementation described in Sec. 2 for different background traffic scenarios.

In order to obtain reproducible and at the same time realistic results, we ran our experiments on a local testbed built in such a way to mimic as realistically as possible the characteristics and dynamics of real networks. On this testbed, in addition to the Information System (a machine running a MySQL DBMS), we deployed 5 Relays, 1 Server, 1 Proxy, and 1 Client (the motivations for using a small overlay configuration are discussed in Sec. 4.2.1). We decided to use our own, locally controlled testbed rather than publicly available ones (e.g., PlanetLab [3] or Emulab [29]) in order to maintain control over the workload ran on the composing machines and the traffic load injected into the network. In this way, we could ensure experiment repeatability, tool comparison in the same operational conditions, and accuracy of result interpretation, while at the same time maintaining the realism of the environment in which experiments were executed.

In the remainder of this section we describe (a) the characteristics of our network testbed (Sec. 4.1), (b) the scenarios in which the comparisons were performed (Sec. 4.2), and (c) the results we obtained (Sec. 4.3).

4.1 The Network Testbed

For our testbed, we used a cluster of PCs running the Linux operating system, each one equipped with an AMD 1.6 GHz Athlon, 724 MB of main memory, a 60 GB hard disk, and two Fast Ethernet network interfaces. The machines were dedicated to our testbed (i.e., no other applications were executed during the experiments), and did not have any external network connections, so no exogenous traffic could enter the testbed network.

In order to reproduce the behavior of a real network, we equipped our testbed with suitable mechanisms for the specification and enforcement of the capacities of virtual links, as well as for the injection of background traffic streams (i.e., traffic streams due to other applications using the network) on them.

Virtual link capacity is set and enforced by means of the *Traffic Control* (TC) mechanisms available in the Linux kernel [2]. TC works by associating with each network interface a set of queues, each one representing a virtual (unidirectional) link towards a specific host, and by enforcing a user-defined transmission rate for each of them. In our testbed, on each machine a queue was created for each virtual link directed towards the other File Mover components. We chose TC rather than alternative possibilities (e.g., NistNET [4]) since this resulted in maximal stability and minimum overhead.

We developed two different mechanisms for the injection of background traffic. The first one generates real traffic (i.e., real packets are transmitted across virtual links), and relies on the *Distributed Internet Traffic Generator* (D-ITG) [21]. D-ITG is a platform capable of producing packet-level traffic whose characteristics (inter-packet departure time and packet size) can be specified as random variables having various probability distributions, and of generating traffic at the network, transport, and application level. Each machine of the testbed runs a generator process, that is in charge of generating the background traffic for all the virtual links departing from that machine.

While this mechanism generates a very realistic workload, it places a rather heavy demand on the CPU of each machine (we measured an occupation of up to 20% of the CPU for each generated stream), that greatly interferes with the File Mover. In order to avoid such interference, we have developed an alternative, trace-driven mechanism that reproduces the effects of the background traffic (namely, the reduction of the bandwidth available on each virtual link) without actually generating and sending the packets. This mechanism uses a trace containing samples of the available bandwidth for each virtual link in the overlay, collected at regular intervals. The background load generator traverses the trace, sets the capacity of the virtual link to the next value contained in the trace by using TC, and then waits until the next time instant in the trace to repeat the same operation. For example, if the trace file reports that at time $t = 10$ the available bandwidth on a virtual link is 30 Mbps, while at time $t = 20$ it drops to 10 Mbps, the generator (a) at time $t = 10$ sets the capacity of the virtual link to 30 Mbps, (b) suspends itself until time $t = 20$, and (c) wakes up at time $t = 20$ and sets the capacity of the virtual link to 10 Mbps. The overhead of this method is negligible and, as we verified experimentally by comparing the file transfer throughput obtained with the emulated and the real traffic, results in realistic effects on file transfers.

In our experiments, the traces were collected by running D-ITG (in isolation) on all the machines of the testbed, with the traffic parameters discussed in Sec. 4.2.2, and by sampling the available bandwidth of each virtual link every 10 seconds.

4.2 Experimental Scenarios

For our experiments we considered three scenarios, corresponding to injecting three different background traffic workloads on an overlay configuration obtained by setting to specific values the capacities of individual virtual links. [2]

4.2.1 Overlay Configuration

The overlay configuration used for our experiments is depicted in Fig. 3 where, for the sake of readability, the two virtual links connecting the same pair of Relays (recall that IP routing is in general asymmetric so virtual links are unidirectional) are shown as a single, bi-directional edge labeled with two capacity values: the first values corresponds to capacity of the virtual links connecting the lower index Relay with the higher index one (e.g., Relay 3 to Relay 4), while the second one is associated with the virtual link connecting the higher index to the lower index Relay (e.g, Relay 4 to Relay 3). For instance, in Fig. 3 the capacity of the virtual link connecting Relay 3 to Relay 4 is 82 Mbps, while that of the virtual link connecting Relay 4 to Relay 3 is 93 Mbps.

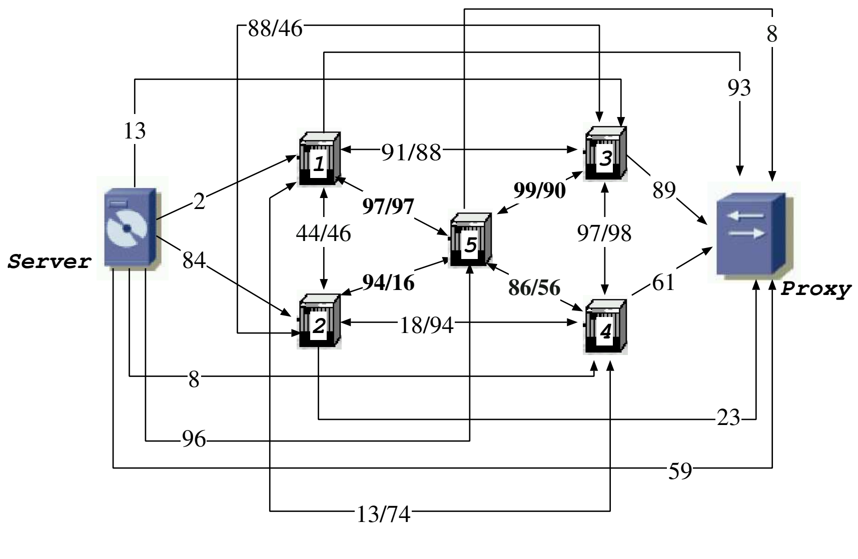

Figure 3: Overlay configuration used for the experiments.

[2]In this paper we define the capacity of a network path as the maximum throughput that the path can provide to a flow when there is no cross traffic [22].

Table 1: Distribution of virtual link capacities

Capacity (Mbps)	**Fraction (%)**
Uniform(0,20)	30
Uniform(20,50)	9
Uniform(50,80)	6
Uniform(80,100)	55

As can be seen from the above figure, our topology includes one *Server*, one *Proxy*, and five *Relays*. The choice of a relatively small configuration depends on the fact that the amount of CPU capacity required by D-ITG to generate the background traffic is directly proportional to the number of streams departing from the node (i.e., the node *fan-out*). In our case, being the overlay fully connected, the fan-out of each node is $N - 1$, where N is the total number of nodes in the overlay. The machines used in our testbed, being relatively old, could sustain the generation of at most 6 D-ITG streams, thus limiting the size of the overlay we could experiment with. However, in spite of its limited dimension, this topology allows the emergence of interesting performance differences between the plain File Mover version, and the extended ones.

The capacities of individual virtual links have been chosen in such a way to closely reproduce the characteristics of a realistic network setting by using the results presented in [16] (summarized in Table 1), where the capacity distribution of the network paths among PlanetLab nodes has been reported.

4.2.2 Background Traffic Load

As already discussed, in order to mimic the behavior of a realistic production network, we injected background traffic on all the virtual links of our overlay testbed. This traffic consists in a set of data streams transmitted between pairs of testbed nodes, each one characterized by a set of parameters related to its *intensity* (i.e., how much resources it uses) and *dynamism* (i.e., how it uses these resources). The parameters of a stream related to its dynamism are its lifetime, the protocol used to transport data, and the distribution of packet size, while intensity is expressed in terms of the time elapsing between the generation of two consecutive packets (the *inter-packet time*).

For the sake of realism, for our experiments the characteristics of these streams have been set according to the latest research results in the field of Internet traffic characterization. Although an analysis of the relevant literature reveals that no single traffic model exists that takes into consideration all the parameters mentioned above, there are studies that address each of the above characteristics separately. We have therefore developed our own traffic model, whose parameters are summarized in Table 2, that puts together the results of these studies by setting the various parameters as follows:

Table 2: Common background traffic parameters

Stream parameter	**Value**
Lifetime distribution	45% less than 2 sec 53% less than 15 minutes 2% up to 2 hours
Protocol type distribution	80% TCP 20% UDP
Packet size distribution	25% Uniform(40,44) 50% Uniform(45,576) 25% Uniform(577,1500)
Inter-packet time distribution	Exponential (parameter set according to Eq.(3)

- *Lifetime distribution*: according to the results in [7], we assume that (a) about 45% of the streams last 2 seconds or less, (b) about 53% of the streams last less than 15 minutes, and (c) about 2% of the streams last up to several hours or days (in our experiments, the maximum stream lifetime was set to one day).
- *Protocol type distribution*: we use the protocol distribution reported in [13], where a longitudinal study of various academic, research and commercial sites lasting more than four and a half years (1998-2003) revealed that as far as the number of transferred bytes is concerned, 80% of the traffic is due to TCP streams, almost 20% is due to UDP, while all the other protocols sum up to less than 1%. Consequently, in our experiments we considered only TCP and UDP streams with proportion of 80% and 20%, respectively, of the total.
- *Packet size distribution*: the distribution of packet sizes has been set according to [17], where a study lasting more than 10 months revealed that the size of about 25% of the packets exchanged between two network endpoints was uniformly distributed between 40 and 44 byte (the minimum packet size for TCP), of about 50% of the packets was uniformly distributed between 46 and 576 bytes, while for the the remaining 25% was uniformly distributed between 577 and 1500 byte (the maximum Ethernet payload size).
- *Inter-packet time distribution*: as indicated by recent work [26], the distribution of the time elapsing between the generation of two consecutive packets can be closely approximated by an exponential, so we adopted this choice. The parameter of the exponential, corresponding to the average number PPS of *packets sent per time unit*, is set (as discussed later in this section) in such a way to obtain different background load intensities.

In order to perform our comparison in various operational conditions, we defined three different background traffic workloads corresponding to situations of low, intermediate, and high intensity. The intensity of background traffic is defined as the fraction of the *average overlay capacity* (*AvgCap*) that it consumes, that is in turn defined as the arithmetic average of the capacities of the individual virtual links. The three background traffic workloads considered in our study, denoted in the rest of the paper as *Low*, *Medium*, and *High*, correspond to intensities of $20\%, 50\%$, and 80%, respectively. The parameter PPS of the inter-packet time distribution for the background traffic streams transmitted over the various virtual links is computed as

$$PPS = \lfloor \frac{AvgCap * BgTrafIntensity}{MaxPacketSize} \rfloor \quad (3)$$

where $BgTrafIntensity$ is the intensity of the background traffic, and $MaxPacketSize$ is the maximum packet length. For the overlay configuration used in our experiments, we have that $AvgCap = 63$ Mbps, so the value of PPS are $1050, 2625$, and 4200 for the *Low, Medium*, and *High* background traffic intensity, respectively.

4.3 Experimental Results

To assess whether the extensions proposed in this paper actually result in performance improvements, we performed a set of experiments consisting in the repeated transfers of a 2 GB file for all the scenarios discussed above. Our experiments were aimed at evaluating the following two features of the proposed extensions:

- the *performance gain* that they provide with respect to the plain File Mover version. The performance metric used to carry out this comparison was the *average effective throughput*, computed as the arithmetic average of the throughput obtained in a set of five transfers of the same file, which has been in turn computed as the ratio of the file size over the time taken to transfer it;
- its *adaptability* to variations of the network conditions due to background traffic, measured by the number of different virtual paths used by the File Mover to carry out all the transfers of the same file (two paths are considered different if they differ in at least one Relay).

In order to perform the comparison in the same operational conditions, we ran first all the experiments involving the plain version of the File Mover, and then we restarted the testbed and performed the experiments involving the extended version exactly at the same time instant from the beginning of the injection of background traffic.

The results obtained in our experiments, discussed in detail in the following subsections, demonstrate that the File Mover extensions we propose result in significantly improved performance.

4.3.1 Evaluation of Transfer from Cached Copies

In order to evaluate the effectiveness of using cached copies as sources for file transfers, in our experiments we placed a replica of the transferred file on the various *Relays*, and then performed a set of experiments where, for each background traffic intensity, we requested the transfer of the file to the *RLM*, that selected the replica resulting in the best expected performance. The results of our experiments, reported in Table 3, show that the use of cached copies (*Caching FM* column) greatly enhances performance for all the background traffic intensities we considered with respect to the plain File Mover version (*plain FM* column). More specifically, the gain decreased for increasing values of the background traffic intensity, and ranged from 18.12% (*Low* intensity) to 7.81% (*High* intensity). This demonstrates the effectiveness of cached file transfers, especially in situations where the amount of background traffic limits mostly the virtual links departing from the *Server*, as in the case of our experiments.

Background	**Average throughput (Mbps)**			**Num.**
Traffic Load	*Caching FM*	*plain FM*	*Gain*	**paths**
Low	82.69	70.00	18.12%	1
Medium	76.99	67.43	14.18%	2
High	70.58	65.47	7.81%	3

Table 3: Comparison of cached versus standard File Mover transfers.

The ability of the File Mover to adapt to variations in the background traffic intensity on individual virtual links is demonstrated by the results reported in column *Num. paths* (reporting the different number of virtual paths used for the experiments): while only one path has been used to carry out all the 5 transfers for the *Low* intensity, two and three paths have been selected for the *Medium* and *High* intensities.

4.3.2 Evaluation of Multipath Transfers

The second set of experiments was aimed at evaluating the possible performance benefits resulting from multipath transfers. For these experiments, all the transfers were performed by using the *Server* as the source. For each transfer request, the *Server* computed the virtual paths to be used to transfer the different chunks of the file, and determined the size of each chunk as indicated by Eq.(2). In order to force the *Server* to use several virtual paths, we set *a priori* to two the number of virtual paths used simultaneously to carry out the transfer. Without this restriction, the *Server* could have used all the available *Relays* to set up a single virtual path, an event relatively likely given the small number of *Relays* of our topology, thus preventing us of performing experiments using multipath transfers.

The results obtained in our experiments are reported in Table 4, where we can observe that, again, the introduction of multipath transfers (*Multipath FM* column) results in significantly better performance with respect to the plain File Mover version for all the background traffic intensities we considered. More specifically we note that the performance gains are substantial (they range from 79.8% to 42.06%), and they decrease for increasing background load intensities. This is a consequence of the fact that a multipath transfer terminates when *all* the transfers of all the chunks have been completed, so a reduction of the achievable throughput *on any* virtual path degrades the performance of the whole

Background Traffic Load	**Average throughput (Mbps)**			**Num. paths**
	Multipath FM	*plain FM*	*Gain*	
Low	125.86	70	79.80%	2
Medium	100.46	67.43	48.99%	2
High	93.01	65.47	42.06%	3

Table 4: Comparison of multipath versus standard File Mover transfers.

transfer. The probability of such a degradation increases with the number of used virtual paths, and is more likely to occur for higher intensities of the background load.

4.3.3 Evaluation of Striped Transfers

The last set of experiments we carried out was aimed at assessing the performance of striped transfers. We modified the overlay topology shown in Fig. 3 by replacing one of the *Relays* (Relay 5) with a *Server*, on which we created another copy of the file to be transferred in the experiments. This modification was required by the need of keeping constant the size of the overlay, for the reason discussed in Sec. 4.2.1. For each transfer request, the Proxy contacted the original Server first, that computed its best virtual path, and then the new one that used the remaining Proxies to compute its best virtual path. Each Server then independently computed the size of the chunk it sent by using Eq. (2), that was transferred to the Proxy. The transfer of the file was considered concluded when both chunk transfers were completed. The results of our experiments, shown in Table 5, once again show that the proposed extension (*Striped FM* column) results in performance improvements over the plain File Mover version. We note, however, that the gain strongly decreases for increasing values of the background load intensity. This is explained by the fact that the choice of computing the first virtual path first, and then the second one by using the Relays that had been discarded by the first Server, resulted in virtual paths characterized by very different throughput values. Consequently, variations in the throughput exhibited by the slower virtual path may have a negative impact on transfer performance even if the faster one is working at full speed. While this is unlikely to occur for relatively low intensities of the background traffic, thanks to the way the size of the individual chunks is computed, its negative impact tends to become higher for increasing intensities, as indicated by the results reported in the *Medium* and *High* rows of Table 5. This phenomenon

Background Traffic Load	**Average throughput (Mbps)**			**Num. paths**
	Striped FM	*plain FM*	*Gain*	
Low	82.32	70.00	17.60%	2
Medium	71.48	67.43	6.01 %	3
High	68.18	65.47	4.14%	3

Table 5: Comparison of striped versus standard File Mover transfers.

calls for better strategies for the computation of the virtual paths used by the various sources used to carry out a striped transfer, that we plan to include in our future work. In any case, in spite of this suboptimal choice, striped transfers always resulted in performance improvement, thus demonstrating the potentials of this technique.

5 Conclusions and Future Work

In this paper we have considered various file transfer policies (namely, *multipath* and *striped*, as well as transfers using *cached copies* as source) for possible inclusion into the File Mover, a software infrastructure providing a high-performance file transfer service for Grid platforms. We have implemented a prototype version of the File Mover that included this extensions, and used it to carry out an experimental study aimed at comparing its performance versus those attained by the plain File Mover implementation. As expected, the techniques we considered result in significant performance improvements but, given the limited size of the overlay we considered, these results should be considered only preliminary. A more thorough experimentation with larger configurations is required in order to make definitive conclusions about them. Nevertheless, although preliminary, our results can be considered encouraging.

There are several avenues of research we plan to explore in the future. First of all, we plan to study and implement transfer strategies obtained by combining the various extensions described in this paper. More precisely, in addition to the rather obvious combination of multipath and cached transfers, in which multipath transfers are used also by the Relays selected as source for the transfer, we plan to study the combination of striped and multipath transfers, in which each chunk sent by a separate source is transferred by using a multipath approach.

A second avenue of research is concerned with the investigation of better algorithms for the selection of virtual paths when more sources are used for the same transfer (as in the case of striped transfers), or when independent file transfers are performed by different sources.

Third, we plan to improve the current method used to collect measurements of the throughput achievable on individual virtual links by replacing the application-level probes used in the current version of the File Mover (that, as already mentioned, uses *iperf*) with a more scalable and accurate approach, working at the network level, like *Plab* [5]. Finally, on the experimental side, we plan to perform experiments by using larger, and more realistic, overlay configurations thanks to the use of more hence powerful machines for our network testbed.

References

[1] Iperf: The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf.

[2] Linux advanced routing & traffic control. http://lartc.org/.

[3] The planetlab project. http://www.planet-lab.org/.

[4] The NISTNET project. http://snad.ncsl.nist.gov/nistnet/.

[5] The PLAB project. http://www.grid.unina.it/software/Plab.

[6] C. Anglano and M. Canonico. The File Mover: High-Performance File Transfer for the Grid. Submitted for publication. Available from http://dcs.mfn.unipmn.it/.

[7] N. Brownlee and K. Claffy. Understanding internet traffic streams: Dragonflies and tortoises. *IEEE Communications Magazine*, 40(10):110–117.

[8] M. Cai, A. Chervenak, and M. Frank. A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table. In *Proc. of Supercomputing 2004*, Pittsburgh, PA, USA, Nov. 2004. IEEE Press.

[9] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[10] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end wan service availability. *IEEE/ACM Trans. Netw.*, 11(2):300–313, 2003.

[11] D.G. Andersen and H. Balakrishnan and M.F. Kaashoek and R.Morris. Resilient Overlay Networks. In *Proc. of 18th ACM Symp. on Operating Systems Principles*, Banff, Canada, Oct. 2001.

[12] D.D. Doval and D. O'Mahony. Overlay Networks: A Scalable Alternative for P2P. *IEEE Internet Computing*, pages 79–82, July–August 2003.

[13] Marina Fomenkov, Ken Keys, David Moore, and K Claffy. Longitudinal study of internet traffic in 1998-2003. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[14] Yunhong Gu and Robert L. Grossman. Optimizing udp-based protocol implementation. In *Proc. of the Third International Workshop on Protocols for Fast Long-Distance Networks PFLDnet*, 2005.

[15] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proc. of ACM SIGCOMM*, pages 175–187, Stockolm, Sweden, September 2000.

[16] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring bandwidth between planetlab nodes. In *Proc. of the Passive and Active Measurement Workshop (PAM 2005)*, pages 292–305, 2005.

[17] S. McCreary and K. Claffy. Trends in wide area IP traffic patterns - A view from ames Internet exchange. *Proceedings of the 13th ITC Specialist Seminar on Internet Traffic Measurement and Modelling, Monterey, CA*, 2000.

[18] H.B. Newman, M.H. Ellisman, and J.H. Orcutt. Data-Intensive E-Science Frontier Research. *Communications of the ACM*, 46(11), Nov. 2003.

[19] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. of ACM SIGCOMM*, pages 25–38, Stanford, CA, August 1996.

[20] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM*, pages 139–152, Cannes, France, September 1997.

[21] A. Pescape, S. Avallone, and G. Ventre. Analysis and experimentation of internet traffic generator, 2004.

[22] R.S. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth estimation: metrics, measurements techniques, and tools. *IEEE Network*, Dec. 2003.

[23] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[24] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: A Case for Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, Jan. 1999.

[25] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 289–299. ACM Press, 1999.

[26] M Molle T Karagiannis and M Faloutsos. Long-range dependence - ten years of internet traffic modeling. *IEEE Internet Computing*, 8(5):57–64, 2004.

[27] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The Impact of Routing Policy on Internet Paths. In *Proc. IEEE Infocom '01*, Anchorage, Alaska, April 2001.

[28] S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. *ACM Computing Surveys*, 38(1), June 2006.

[29] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[30] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Cluster Computing*, 1(1):119–132, Jan. 1998.