Design and Implementation of the Fast Send Protocol

Tilmann Rabl, Christoph Koch, Günther Hölbling and Harald Kosch
University of Passau,
Chair of Distributed Information Systems,
Innstr. 43,
94032 Passau, Germany
E-mail: {rabl,koch,hoelblin,kosch}@fim.uni-passau.de

April 2009

Over the last decades Internet traffic has grown dramatically. Besides the number of transfers, data sizes have risen as well. Traditional transfer protocols do not adapt to this evolution. Large-scale computational applications running on expensive parallel computers produce large amounts of data which often have to be transferred to weaker machines at the clients' premises. As parallel computers are frequently charged by the minute, it is indispensable to minimize the transfer time after computation succeeded to keep down costs. Consequently, the economic focus lies on minimizing the time to move away all data from the parallel computer whereas the actual time to arrival remains less (but still) important. This paper describes the design and implementation of a new transfer protocol, the Fast Send Protocol (FSP), which employs striping to intermediate nodes in order to minimize sending time and to utilize the sender's resources to a high extent.

1 Introduction

Increasing quantities of data produced and stored in grid environments in combination with high-speed wide area networks (WANs) stoke the desire for transferring tremendous amounts of data between dispersed sites. 10 Gbit fiber networks provide a theoretic throughput of more than 1 GB/s. High-performance servers are able to saturate such networks using specialized protocols.

The weakest link on the path from sender to receiver determines the achievable throughput. A slow network link, e.g. a 100 Mbit WAN connecting two fiber Gbit

local area networks (LANs), may be the bottleneck, as well as a poor receiving machine limited by its CPU, hard drive or network interface card (NIC).

There are several concrete usage scenarios which benefit from minimizing the sending time:

disaster recovery: Disasters (e.g. fire, flooding, earthquakes) may threaten a building containing information storage servers. The servers may be rendered physically inaccessible, but still be technically intact for some time. The data has to be "evacuated" as fast as possible. A good solution is to stripe the data to servers nearby before bundling it at a safe and distant server.

server maintenance: Before server hardware is changed or when a server has to be shutdown, a data backup has to be performed. To keep down backup times, it is desirable to minimize the sending time of the transfer.

parallel computers: High-performance parallel computers are frequently shared between many users and are becoming more and more widespread with the advent of computing and data grids. Computing nodes are likely to be geographically far away from their users, especially in the academic science community. Each user has a certain time-slot it has to adhere to, and costs are directly associated with the time the resources are used. After computations have finished, moving the resulting data from the parallel computer has top priority to reduce costs.

FSP aims at overcoming the weakest link by introducing *intermediate nodes*. The sender partitions the data into smaller blocks and starts a striped transfer to distribute the blocks to several intermediate nodes. As soon as all blocks are distributed, the server retreats from the transfer, while the receiver collects the blocks from the intermediate nodes. The striping and collecting phase may be overlapped to reduce overall transfer time. Intermediate nodes are chosen from a preconfigured static list or from a self-organizing peer-to-peer (P2P) overlay.

The remainder of this paper is structured as follows. In section 2 related work and preliminaries are discussed, section 3 gives a technical description of FSP. Section 4 shows experimental results and the conclusion is presented in section 5.

2 Related Work and Preliminaries

The need for high-speed transfers of bulk data between local and remote sites has led to a large number of network protocols and applications. These protocols and applications seek to utilize the network to the largest extent possible. Some approaches implement new transport layer protocols. Others make use of existing protocols (predominantly the Transmission Control Protocol (TCP)) to built application layer protocols and applications. This section examines data transfer protocols at both the transport and application layer. The protocols are compared to the FSP approach where applicable.

2.1 Transport Layer Protocols

As stock TCP is not designed for modern long fat networks[1] (LFNs), several approaches to overcome the performance limitations have been proposed.

2.1.1 pTCP

pTCP (Parallel TCP) is a transport layer protocol that uses several TCP-v connections between sender and receiver Hsieh and Sivakumar (2002). TCP-v is a modified version of TCP. The TCP-v connections do not use the same path through the network. A multi-homed host, that is a host with several network interfaces, can use a separate TCP-v connection on each interface. The TCP-v connections are handled transparently to the user who perceives only a single logical pTCP connection. The pTCP authors refer to a pTCP connection with several TCP-v connections as a striped connection. The aggregated throughput of several TCP-v connections over multiple paths yields a higher throughput than a single TCP connection. In contrast to pTCP, FSP requires only a single network interface.

2.1.2 PSockets

PSockets (Parallel Sockets) is actually an application level protocol that uses several TCP connections to transfer data Sivakumar et al. (2000). However, it is implemented as a library that can be used by applications in the same way as ordinary TCP sockets. PSockets partitions the data into parts of equal size and sends it in parallel over several TCP connections. Handling the TCP connections is hidden from the user who is only concerned with one logical PSocket connection. Parallel TCP streams between two endpoints help with TCPs poor performance in LFNs. However, it raises fairness issues as parallel streams tend to get a larger share of the network than a single stream. Section 3.4.4 provides more details on parallel streams. FSP can also use parallel TCP streams between two endpoints and therefore features the same throughput gain.

2.2 Advanced TCP Stack Implementations

There are several new proposals and implementations for TCP protocols that try to do better in LFNs than the commonly-used TCP Reno and TCP NewReno implementations. They implement new techniques for flow and congestion control. For an overview of the TCP congestion controll see section 3.2.

Fast TCP Fast Active Queue Management Scalable TCP uses a congestion control algorithm that tries to keep a constant number of packets in queues, determined by taking the delay into account.

S-TCP Scalable TCP increases the congestion window exponentially and decreases it by a factor of only 0.125 in case of congestion (compared to 0.5 for TCP Reno) and therefore utilizes the bandwidth to higher extent.

[1] LFNs are networks with a large bandwidth-delay product.

HS-TCP High-Speed TCP defines a new congestion control mechanism. The congestion window is not increased or decreased by a fixed value but in relation to the current congestion window size. It can ramp up the congestion window more aggressively and does not cut window sizes as strongly as TCP Reno.

The interested reader is referred to the empirical evaluation of some advanced TCP stacks in Bullot et al. (2004).

2.2.1 Assessment of the Transport Layer Protocols

Transport layer protocols have to be installed at the operating system level. As the design of FSP is directed to support a heterogeneous environment of machines, FSP itself does not assume that any special transport layer protocol such as an advanced TCP implementation is present. FSP uses application level tuning techniques that work with any TCP implementations. In particular, it uses TCP buffer tuning and parallel TCP streams to obtain good performance with standard TCP stacks.

2.3 Application Layer Protocols

2.3.1 File Transfer Protocol (FTP)

FTP is an application layer protocol used to transfer files from a server machine to a client or vice versa. It is an Internet standard protocol defined by the Internet Engineering Task Force (IETF) in RFC959 as early as 1985 Postel and Reynolds (1985). The core functionality has not been changed since which shows the farsighted design. Besides supporting the transfer of files, FTP allows to delete, create, and rename directories and to retrieve directory listings. It is probably the most widespread data transfer protocol on the Internet.

FTP has one feature that sets it apart from most other network protocols. It employs separate connections for control and data transfer. The terms control and data channel are used interchangeably with control and data connection. Both connections use TCP at the transport layer. The control connection is used by the client to issue requests and by the server to reply to requests. The control connection's endpoints are called User Protocol Interpreter (User-PI) at the client and Server Protocol Interpreter (Server-PI) at the server. The data transfer itself and directory listings are performed via the data connection. The data connection's endpoints are referred to as User Data Transfer Process (User-DTP) and Server Data Transfer Process (Server-DTP).

FTP does not guarantee data integrity. It completely relies on TCPs reliable byte stream transfer.

Additionally, the original FTP specification has some security issues. Most of the time, simple username/password authentication is performed and both username and password are sent over the unencrypted control channel in clear text. To counteract this problem, FTP security extensions have been defined that propose SSL encryption of the control and optionally the data connection Ford-Hutchinson (2005). SSL protected

FTP is referred to as FTPS or FTP/SSL. FTP can also be secured by tunneling it over Secure Shell (SSH). SSH-tunneled FTP is called secure FTP or FTP over SSH.

Being based on TCP, FTP potentially suffers from TCP's bad performance on LFNs (see section 3.2). It does not support features for high-speed transfers such as parallel TCP streams, striping, or TCP buffer tuning.

2.3.2 GridFTP

GridFTP is a protocol extending FTP that offers secure high-speed data transfers in Grid environments. The first GridFTP version is specified in Allcock (2003) and a revised second version was released two years later in Allcock and Perelmutov (2005). The Grid or Grid computing is a form of distributed computing. It allows to make coordinated use of decentralized resources across various administrative and geographic domains to provide high-performance computing and storage amongst others. A good introduction to the Grid may be found in Stockinger (2007).

With Grid computing, tremendous amounts of data are processed and produced. Data has to be transferred across wide areas as Grid resources are often far away from each other. GridFTP provides a set of extension to FTP that enhance performance and security.

GridFTP separates the protocol interpreter from the data transfer process in the same way as FTP. In order to use the combined resources of several machines in parallel, one machine may be used as the controlling front-end that serves as the protocol interpreter. Other machines serve as data nodes (back-end nodes) and are statically registered at the front-end. Together they form a single logical GridFTP server. The protocol used between front-end and data nodes is not part of the standard. Clients connect to the front-end and the front-end server determines which data nodes to use for a transfer.

GridFTP allows to transfer data at very high speeds. In practice, throughputs of 27.3 Gbit/s memory-to-memory and 17 Gbit/s disk-to-disk were reached using a 30 Gbit/s network with 60 ms RTT Allcock et al. (2005).

The Globus Toolkit[2] provides an open source GridFTP server and client implementation. It is tightly integrated in the Grid Security Infrastructure and requires a fair amount of configuration. While GridFTP defines commands to set the TCP socket buffers or trigger automatic buffer negotiation, the Globus implementation does not provide automatic negotiation at all. There are, however, research extensions to GridFTP which implement dynamic right sizing (DRS) (see section 3.2).

One of GridFTP's shortcomings is that it does not allow to stripe data from one server to several other independent servers and vice versa. Consequently, it can not be used for the two stage transfer proposed and employed by FSP.

GridFTP provides point-to-point transfers where both of the logical endpoints may span several hosts (n:m striping). In this sense, FSPs two stage transfer can be seen as a sequence of 1:n striping during the distribution phase and a following n:1 striping during collection at the client. While GridFTP could be used for the independent point-

[2] `http://www.globus.org`

to-point transfers from the server to the intermediate nodes, a controlling entity would still be needed.

2.3.3 UDP-based Data Transfer (UDT)

UDT is a high-performance data transfer protocol built on top of UDP. It implements its own congestion control and reliable transfer because UDP provides only a simple connectionless and unreliable service Gu and Grossman (2007). UDT can reach higher throughputs than untuned TCP because UDP does not incorporate any flow or congestion control that artificially limits the throughput. It transfers data directly between two endpoints. Applications can directly use UDT with a C++ API that compiles on most operating systems. FSP does not use UDT to perform its transfers because similar performance result can be obtained by tuning the TCP connections (see 3.2).

2.3.4 Kangaroo

Kangaroo is an end-to-end background data movement service that moves data from source to destination via a variable number of intermediate Kangaroo servers Son (2001). Kangaroo presumes that many Grid applications are not designed to deal with unexpected failures, such as network outages and out-of-storage errors. Its goal is to hide network or storage errors and latencies from Grid applications to provide a robust storage framework. A client application writes data to (or reads data from) a nearby Kangaroo server. The Kangaroo server routes the data to the next hop server or the destination server. Data is spooled to disk at any intermediate server. A data mover background process is responsible for moving the data to the next hop as soon as possible. When the throughput to the next hop does not suffice or the next hop server is currently unavailable, the mover retries the transfer until it can be finished successfully. Spooling the data to disk at intermediate servers insulates the client application from network failures as long as the temporary storage space is not exceeded. CPU and I/O usage at the client application may be overlapped as a side effect. Kangaroo is sender-centric or output-oriented in that it is mainly designed for writing data to a destination.

Kangaroo transfers data via a single route from source to destination, that is a single series of hops (Kangaroo servers). The routes have to be predefined statically at each Kangaroo server. An extension allows to take advantage of multiple routes to one destination Rajamani and Balakrishnan (2001). This is accomplished by the client application writing to several first-hop Kangaroo servers instead of a single one. However, the routing remains static. The client has to know in advance which first-hop servers to use and all servers along the route need a static routing table.

Kangaroo shares with FSP that it takes advantage of resources at intermediate servers. However, it is mainly sender-centric, that is the sender pushes data to the destination. Furthermore, Kangaroo applies static routing, whereas FSP allows to select several intermediate nodes dynamically. The sender can only decide upon the first-hop. Further hops depend on the static routing tables at every single Kangaroo server. Without its multi-route extension, Kangaroo can only use one single preconfigured path. Additionally,

Kangaroo does not exploit parallel TCP streams or buffer tuning. It has been designed to hide latencies and errors from an application, rather than to perform high-speed bulk data transfers.

2.3.5 Network Logistics

The network logistics approach to bulk data transfers uses a store-and-forwarding mechanism to improve throughput in LFNs Swany (2004). It separates a single point-to-point TCP connection into a path of shorter connections taking advantage of resources at intermediate hosts. Data is sent hop-by-hop along the path and each intermediate host buffers part of the data before sending it to the next host. The bandwidth-delay-product (BDP) of the shorter connections is smaller than the one of the single connection. In case the single connection is flow-control limited because of a TCP socket buffer size that is too small compared to the BDP, the throughput can be increased using shorter connections with the same buffer sizes. For further details of socket buffer sizes and TCP performance see section 3.2. This approach does not require any changes at the operating system or the network level. It is particularly applicable when the maximum buffer sizes at any of the endpoints can not be changed.

In order to obtain optimal performance, the path of intermediate hosts has to be selected in a way that maximizes the throughput. The total throughput is determined mainly by the slowest link on the path. A performance matrix is created containing the cost between each pair of nodes. After estimating the bandwidth with a network weather service[3] forecast, the cost is defined as 1/bandwidth. The optimal path is computed as the path between sender and receiver that has the smallest maximum cost between any pair of nodes along the path (the Minimax path). Empirical results show an increase in the throughput of 5.75% to 9% Swany (2004).

The network logistics approach takes advantage of a whole path of intermediate hosts while FSP uses only one level of intermediate nodes. However, it does not employ parallel streams or TCP buffer tuning and is limited to a single path between sender and receiver. FSP uses a number of intermediate hosts in parallel and thus multiple routes from sender to receiver. When FSP is configured to choose intermediate nodes halfway between server and client, the same network logistics effect is achieved but limited to dividing the connection into two shorter ones.

2.3.6 Assessment of the Application Layer Protocols

Most of the existing and widely used application layer protocols support the direct transfer between two endpoints. In particular, GridFTP uses interesting concepts for high-speed data transfers. There are some protocols such as Kangaroo that take advantage of resources at intermediate hosts in a way similar to FSP. However, there is no protocol that is able to stripe data to several intermediate nodes and collect it at a client machine automatically. Furthermore, FSP distinguishes itself by organizing the

[3] A network weather service is a distributed application that monitors the bandwidth (amongst others) between pairs of hosts and predicts the future bandwidth.

intermediate nodes in a P2P overlay and allowing to select them dynamically according to their estimated location in the network.

3 FSP Protocol

Our design has been motivated by the goal of minimizing the sending time when transferring data from a fast server. We observed several limitations that might hurt throughput in traditional direct file transfers:

slow network link: WAN links somewhere on the path between sender and receiver are likely to offer less bandwidth than the server is able to saturate. Additionally, cross traffic on shared links limits throughput.

slow receiver: The receiving machine may not be able to cope with the server's sending rate. Limiting factors are slow CPUs, hard disks and NICs.

transport protocol limitations: Most file transfers use TCP on the transport layer. TCP Reno, the predominant TCP implementation, performs badly on LFNs, i.e. connections with a large BDP. Its flow-control parameters are often statically tuned for small BDPs. Additionally, packet loss triggers large TCP window reductions.

The Fast Send Protocol addresses all these limitations. The main idea is to replace the direct transfer from sender to receiver by introducing intermediate nodes. Data is striped to several intermediate nodes. These nodes are closer to the sender or have more resources than the receiver. Finally, the data is collected by the receiver. Striping allows to use the combined bandwidth of several nodes and network links to overcome slow direct network links and slow receivers. The transport protocol limitations are addressed by using automatic TCP parameter tuning (see section 3.2).

FSP is an application layer protocol extending FTP Postel and Reynolds (1985), the most ubiquitous data transfer protocol. FTP has been chosen as the basis of our protocol for quite the same reasons as in GridFTP design Allcock et al. (2005): FTP is a widespread and mature protocol. With control and data channel being two separate TCP connections (the figures in this paper do not distinguish between control and data connections), third-party transfers can be implemented and extensibility is eased. A number of extension have already been proposed by the IETF[4], e.g. in the domain of security.

FSP has been designed to be completely compatible with standard FTP and may be used as a drop-in replacement for FTP. Additionally, FSP takes advantage of many GridFTP concepts which address shortcomings of FTP with respect to performance and data integrity.

An FSP data transfer is logically separated into several phases:

request: The data transfer starts with a client requesting a file or directory from an FSP server.

[4]The Internet Engineering Task Force – `http://www.ietf.org/`

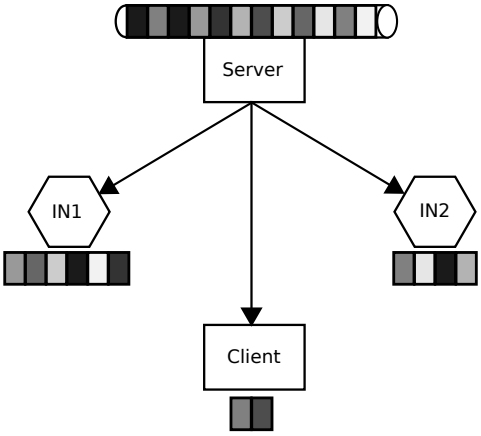

Figure 1: Partitioning and striping using a first-come, first-served distribution strategy.

partitioning: The server partitions the data into blocks of configurable size. A block header containing a descriptor flag, header length, payload length, file offset and file name is prepended to each block. The file name is represented as the relative path with respect to the current directory. Including the file name in each header allows to send complete directories with one single request. Typical header sizes range between 20 and 50 bytes which results in a negligible per-block overhead with sufficiently large block sizes.

intermediate node selection: After that, the intermediate nodes have to be selected from a static list or from a dynamic P2P overlay (see section 3.1 for details). Clever dynamic selection is crucial to the performance and accomplished by considering inter-node distances.

striping: The server opens a connection to each of the intermediate nodes and stripes the blocks according to a distribution strategy (see figure 1). Distribution strategies include *first-come, first-served* (i.e. faster intermediate nodes receive more blocks), *cyclic distribution* (i.e. blocks are allocated round-robin) and *partitioned distribution* (i.e. each intermediate node receives an equally-sized contiguous share of each file). The different strategies enhance load-balancing if it is expected that the data has to be transferred to several clients later on. For a single target transfer first-come, first-served is in most cases preferable.

collection: The server continuously informs the client of the block locations via the control channel, leaving it to the client when to start collecting the distributed blocks. Typically, the distribution includes the client to keep down the total time of transfer and the client starts collecting the distributed blocks while distribution is still in progress. The client opens a connection to each intermediate node and requests the respective blocks. Due to the block format and the header information, out-of-order delivery of the blocks does not pose a problem. When the client uses

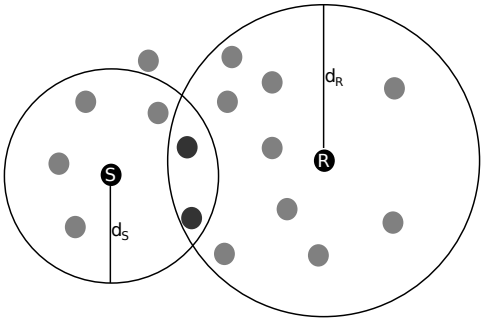

Figure 2: Intermediate node selection using Meridian.

the data for streaming purposes, the block size has to be set quite small, so that a slower intermediate node does not delay the reception of a needed block.

The following subsections describe FSP's features and techniques in more detail.

3.1 Intermediate Node Selection

Selecting those intermediate nodes which offer the highest bandwidth is crucial to minimizing the sending time. Currently, FSP selects nodes according to their network location taking into account their distances reflected by the latency (round-trip time) to both sender and receiver. The actual distance constraints are configurable. One may choose intermediate nodes very close to the sender (maybe on the same LAN) without constraints regarding their distances to the receiver. While this approach is optimal to reduce sending time, it may hurt the total transfer time. A compromise is to choose nodes somewhere in the middle between sender and receiver.

FSP organizes the intermediate nodes using a *Meridian* Wong et al. (2005) P2P overlay as its network location service. Meridian allows to select nodes based on their location in a network by multi-constraint queries (MCQ). Each Meridian node keeps track of a logarithmic number of nodes organized in a set of concentric rings centered around itself with exponentially increasing radii. The round-trip time is used to place the nodes in the rings.

A node selection query $MCQ = \{(s, dc_s), (r, dc_r)\}$ consists of two constraints, the distance constraints to the sender dc_s and receiver dc_r. The resulting nodes are located in the intersection of the two circles with the radii dc_s around the sender s and dc_r around the receiver r respectively as shown in figure 2. Meridian is specifically designed to process MCQs and find a set of nodes which satisfy multiple constraints on their network location. A Meridian node p processes an $MCQ = \{(t_1, dc_{t_1}), (t_2, dc_{t_2}), ..., (t_n, dc_{t_n})\}$ in several steps:

- First, the distances d_{pt_i} between node p and all constraint targets t_i are measured.
- After that, the node calculates its distance s to the solution space using the distance function $s = \sum_{i=1}^{n} max^2(0, d_{pt_i} - dc_{t_i})$. When the node does not satisfy one of the

constraints, it is penalized by adding the squared distance to that constraint to the total distance to the solution space. A distance of 0 indicates that the node p satisfies all constraints itself and is part of the result set.

- Subsequently, all locally known ring members j are considered whose distances d_{pj} to itself satisfy

$$\forall i = 1...n : max(0, (1-\beta) \cdot (d_{pt_i} - dc_{t_i})) \leq d_{pj} \leq (1+\beta) \cdot (d_{pt_i} + dct_i)$$

  Where β ($\beta \in [0..1]$) is called the *route acceptance threshold*.

  This includes all nodes that satisfy at least one of the constraints or that are close to the solution space at any rate. These nodes are requested to measure their distances to all the constraint targets t_i and reply with the measurement results. With these distances, the distance to the solution space s_j can be calculated for all the nodes j. Again, a distance of 0 indicates that j satisfies all constraints and it is added to the result set.

- When no or too few nodes lie in the solution space, the query is forwarded to the node closest to the solution space provided it satisfies $s_j < \beta \cdot s$. This ensures that the query is not forwarded to a nearby node which is also likely to be unable to provide better results. A larger β causes more nodes to be considered as possible solutions and as forwarding targets and increases the possibility of finding a large result set. However, the downside is a larger number of hops and an increased bandwidth requirement due to more measuring requests and query forwardings. Hence a query takes longer to be processed.

A nice property of Meridian is that a node does not have to be a member of the overlay to issue a query. It is sufficient to know at least one Meridian node as an entry point.

3.2 TCP tuning

FSP is expected to transfer data between geographically dispersed nodes via high-speed networks. These kind of networks exhibit large round-trip times (RTT) implying a large BDP with $BDP = RTT * Bandwidth$. TCP, the underlying transport protocol, has not been designed to adapt to such networks. It uses window based flow and congestion control. The sender is only allowed to send up to W_s (send window size) bytes without receiving an acknowledgement from the receiver. When W_s is small compared to the BDP, the sender has to wait for acknowledgements most of the time and the throughput T is limited by $T = W_s/RTT$. For instance, a transatlantic TCP connection on a 100 Mbit/s line with 90 ms RTT (and $BDP = 1.125\ MB$) and a standard $W_s = 64\ KB$ yields a throughput of only $T = 64\ KB/90\ ms = 5.56\ Mbit/s$. A W_s larger than the BDP wastes system memory, since the window is never completely used. Therefore best results are achieved with $W_s = BDP$. W_s is the minimum of the congestion window, the receive window and the sender's socket buffer size. The window sizes are hidden in the TCP implementation and cannot be set directly from within the application layer.

The window size is set indirectly by changing the send and receive buffer sizes via the socket API. The maximum possible buffer sizes are limited by operating system settings and have to be adjusted if necessary Allcock and Bresnahan (2004).

Automatic buffer tuning. Several techniques have been proposed to automatically tune buffer sizes. For an excellent comparison of the most important ones we refer to Weigle and Feng (2002). For instance, AutoNcFTP measures the BDP at connection set-up and sets buffers accordingly National Laboratory for Applied Network Research (2007). However, it suffers from a potentially fluctuating BDP. FSP implements a different approach called dynamic right-sizing (DRS) that has been designed for GridFTP Gardner et al. (2004). DRS continually determines both current bandwidth and round-trip time. Bandwidth is estimated by the receiver by simply calculating the current throughput. To estimate the RTT, the receiver periodically sends a special block header containing the receiver's buffer size to the sender on the data channel (unlike FTP, the data channel is used bi-directional). The sender tries to adjust its own send buffer size to the one received and acknowledges by setting a special descriptor flag in the header of the next outgoing block. Upon receiving the acknowledgement, the receiver can calculate the RTT as the time it took the acknowledgement to arrive. This method slightly overestimates the RTT because of the application layer protocol overhead, but serves as a good estimate. After calculating the BDP, the receiver updates its buffer sizes and informs the sender with the next RTT measurement header. DRS makes sure that the connection is never limited by the flow-control window.

Parallel streams. Besides buffer tuning, FSP adopts GridFTP's concept of parallel TCP streams to improve TCP throughput. The advantages of parallel TCP streams are threefold Ito et al. (2006). First, using n streams, means that n-times the TCP buffer size is available compared to a single TCP connection. Second, ramping up the transfer rate during slow-start is accelerated. Third, aggregated throughput in the congestion-avoidance phase is increased because recovery from packet losses is faster and competing TCP connections are suppressed. Using parallel streams on networks carrying cross-traffic may hurt the throughput of competing connections. Contrary, DRS connections are TCP-friendly.

Parallel streams are especially useful when buffer sizes can not be tuned due to operating system restrictions. The number of parallel streams has to be set manually by the user. A good rule of thumb is to use 4 parallel streams. Combining DRS and parallel streams may even provide better results Gardner et al. (2004). However, using too many streams hurts the throughput because of the overhead associated with driving many connections.

3.3 Compression

FSP optionally supports on-the-fly per-block compression using the library zlib Deutsch and Gailly (1996) with configurable compression levels. zlib has been chosen because it is an open standard with open-source libraries being freely available. However, other

algorithms could be integrated easily. As compression is computationally expensive, it does not increase throughput in most of the cases. Good results may be achieved with highly compressible data such as text files or on slow network connections where compression speed is able to keep up with network speed. Intermediate nodes do not decompress blocks to save CPU cycles and storage space.

3.4 Data integrity

When large amounts of data have to be transferred without any bit errors, it is crucial to detect the range of bytes where an error occurred. This allows to retransmit only the erroneous block instead of the complete file. TCP offers a reliable byte stream with its own error detection mechanism, but the checksums used are rather weak Stone and Partridge (2000). Standard FTP does not offer any integrated means to ensure data integrity. One has to resort to manually transferring a precomputed checksum and checking the validity. With only a single checksum per file, the complete file has to be retransmitted which is very inefficient.

FSP optionally computes its own per-block checksums and uses the same approach to data integrity as GridFTP. Each block is appended a checksum calculated over the complete bytes of the block. Transmission errors can be tracked down on a per-block basis. As block sizes are relatively small compared to the complete file length in most cases, retransmitting erroneous blocks is very efficient. Retransmission is always triggered by the receiving endpoint. The checksum algorithm may be chosen freely. Our current implementation supports ADLER32, CRC32, MD5 and SHA1 checksum algorithms.

3.5 Security

FSP currently uses username/password authentication over a Transport Layer Security (TLS)[5] encrypted control channel. Both server and client have to have a user account at each intermediate node or there must be a common account at each intermediate node, respectively. We are aware of the security implications and that FSP can be extended with a more secure mechanism such as X.509 certificates.

Being based on FTP, FSP has both a control and a data channel. Both are point-to-point connections, but the data they carry differs in its security needs. The control data exchanged on the control connection is only relevant to the two involved parties. Point-to-point security is sufficient and FSP encrypts the control channel with SSL/TLS in the same way as the protocol extension to FTP proposes in RFC4217 Ford-Hutchinson (2005).

On the other hand, data exchanged on the data channel has end-to-end semantics. While it is processed and stored at several intermediate nodes, only the client and server should be able to read the (confidential) data as clear text. Blocks of data distributed to intermediate nodes are not supposed to be readable at the intermediate nodes or on the wire during transmission. Thus, there is a need for end-to-end security, which is not provided by TLS. FSP encrypts the blocks with symmetric block ciphers offering AES

[5]TLS is the successor of Secure Sockets Layer (SSL).

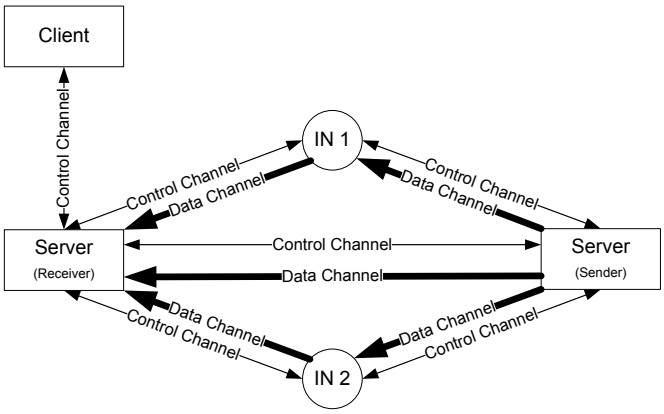

Figure 3: Third-party transfer with FSP.

National Institute of Standards and Technology (2001) and Blowfish Schneier (1994) encryption with bit sizes of at least 128 bits. The encryption key is securely exchanged between server and client via the TLS protected control channel. Intermediate nodes are not able to decrypt the blocks and are not aware of the content they store.

Encrypting is computationally expensive and has a negative performance impact. The user has to decide upon the tradeoff between security and performance.

3.6 Third-party transfers

Transfers between two servers initiated by a third party are referred to as third-party transfers. They are implemented differently than in FTP or GridFTP. The client opens a control connection to the receiving server and indirectly triggers the transfer between the two servers (shown in figure 3). It acts like a "remote control" for the transfer. The receiving server retrieves the requested file from the sending server as if it were a client. Unlike FTP, the client does not directly request one of the servers to open any data connections. This approach keeps the complexity of opening connections to the intermediate nodes at the servers.

4 Experimental Results

The test setup is shown in figure 4. The server has a 1 Gbit ethernet connection to a Gbit switch. The intermediate nodes and the client are connected to 100 Mbit switches which are themselves linked to the Gbit switch. The link to intermediate node 4 (IN4) is routed via a virtual private network and carries cross-traffic. Both the client and IN3 share the same 100 Mbit uplink to the Gbit switch. RTTs between the nodes range between 0.3 ms and 2 ms. The machines are not limited by their CPU or harddrive, but only by the network connection. All tests were conducted with transfers of a single 1 GB file using blocks of size 1 MB. DRS was enabled which generally set the TCP buffer sizes to 128 KB.

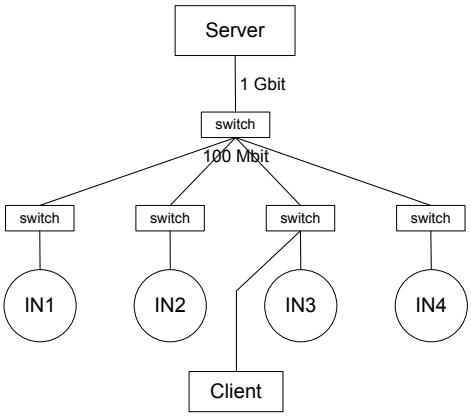

Figure 4: Test setup.

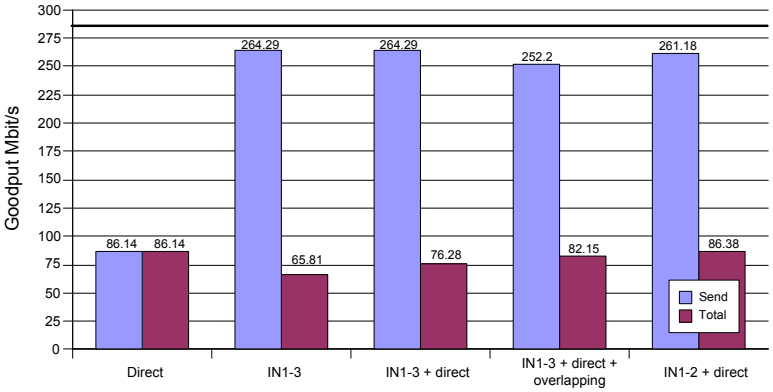

Figure 5: Experimental results using 3*100 Mbit lines.

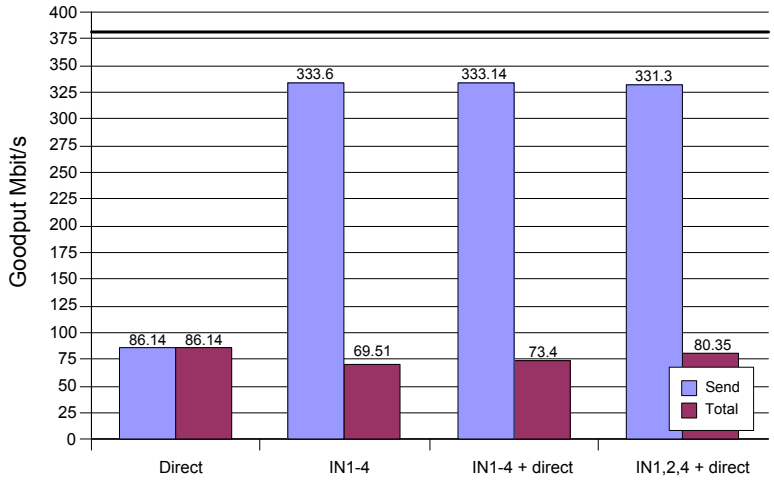

Figure 6: Experimental results using 4*100 Mbit lines.

The first test uses intermediate nodes 1 to 3. The maximum (theoretically) achievable bandwidth during striping is 300 Mbit/s while the client is limited by its 100 Mbit/s connection. Figure 5 shows the results of our tests. The left bar of each column shows the goodput[6] during striping, the right bar indicates the goodput of the total transfer. A direct transfer between server and client resulted in a goodput of 86.1 Mbit/s. Striping to IN1-3 without including the client in the striping phase yields 264 Mbit/s and 66 Mbit/s in total. Including the client in the striping phase increases the total goodput to 76 Mbit/s (striping bandwidth remains the same as the client and IN3 compete for bandwidth). Additionally, overlapping the striping and the collection phase reduces the striping goodput to 252 Mbit/s, but increases the total goodput to 82 Mbit/s. Best overall results were obtained by striping to IN1 and IN2 and including the client in the striping phase (thus removing bandwidth competition between client and IN3). With 261 Mbit/s striping goodput and 86.4 Mbit/s total goodput, we obtained 91.6% of the theoretical 284.8 Mbit/s maximum striping goodput and even exceeded the total goodput of a direct transfer between client and server.

Our second test takes advantage of striping to all four intermediate nodes (see 6). This increases the theoretical striping throughput to 400 Mbit/s (however, the line to IN4 carries cross traffic). We achieved 83.4% of the theoretical striping throughput, but the total goodput ranged between 69 Mbit/s and 80 Mbit/s. We attribute the worse total goodput to varying amounts of cross traffic on the route to IN4.

Our results show, that the striping approach significantly increases the throughput during sending and thus reduces the sending time. It is very promising to note, that under certain circumstances (test *IN1-2 + direct*) the total goodput is even slightly higher than in a direct transfer.

[6]Goodput is defined as the application level throughput, that is the amount of useful bits transferred per second. In our case, the overhead of the block headers and the protocol overhead of transport, network, data link and physical layer are excluded.

5 Conclusion

In this paper we presented a new approach for minimizing sending time in high-speed bulk data transfers. The Fast Send Protocol uses intermediate nodes and striping mechanisms to maximize the amount of data sent by the server and to reduce the impact of slow network links. The selection of intermediate nodes is either static or based on a P2P overlay. FSP adopts concepts of GridFTP and reaches equal overall transfer rates. It is completely compatible with FTP. Optionally, data integrity can be ensured by using per-block checksums. Additional mechanisms for encryption, compression and third-party-transfers have been included. Experimental results show that FSP significantly increases the bandwith utilization at the sender side, thus reducing the sending time. Another finding is that the overall transfer times of striped transfer and direct transfer are equal.

Further research will address the dynamic adjustment of the number and the selection of intermediate nodes during data transfer. To fulfil stricter security requirements support of certificates will be added. The dynamic adaptation of the number of parallel streams as described in Ito et al. (2006) could further improve the transfer rate.

References

W. Allcock, "GridFTP: Protocol Extensions to FTP for the Grid," Global Grid Forum, Tech. Rep., April 2003.

D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

H. Sivakumar, S. Bailey, and R. L. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 2000, p. 37.

J. Postel and J. Reynolds, "File Transfer Protocol (FTP) (RFC959)," IETF Network Working Group, Tech. Rep., October 1985.

W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The Globus Striped GridFTP Framework and Server," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 54.

B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 85–96, 2005.

W. Allcock and J. Bresnahan, "Maximizing Your Globus Toolkit GridFTP Server," *ClusterWorld*, vol. 2, no. 9, pp. 2–7, 2004.

E. Weigle and W. Feng, "A Comparison of TCP Automatic Tuning Techniques for Distributed Computing," in *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 265.

National Laboratory for Applied Network Research, "Automatic TCP Window Tuning and Applications." [Online]. Available: `http://dast.nlanr.net/Projects/Autobuf/autotcp.html`

M. K. Gardner, S. Thulasidasan, and W. Feng, "User-space auto-tuning for TCP flow control in computational grids," *Computer Communications*, vol. 27, no. 14, pp. 1364–1374, 2004.

T. Ito, H. Ohsaki, and M. Imase, "Automatic Parameter Configuration Mechanism for Data Transfer Protocol GridFTP," *SAINT '06: Proceedings of the International Symposium on Applications and the Internet*, pp. 32–38, 2006.

P. Deutsch and J.-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3 (RFC1950)," IETF Network Working Group, USA, Tech. Rep., 1996.

J. Stone and C. Partridge, "When the CRC and TCP Checksum Disagree," in *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication.* New York, NY, USA: ACM Press, 2000, pp. 309–319.

P. Ford-Hutchinson, "Securing FTP with TLS (RFC 4217)," IETF Network Working Group, Tech. Rep., October 2005.

National Institute of Standards and Technology, *Federal Information Processing Standard 197, Advanced Encryption Standard (AES)*, Nov. 2001. [Online]. Available: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

R. Rajamani and G. Balakrishnan, "Efficient Large-scale data movement on the Grid Augmenting the Kangaroo approach," unpublished , 2001.

B. Schneier, "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)," in *Fast Software Encryption, Cambridge Security Workshop.* London, UK: Springer-Verlag, 1994, pp. 191–204.

S.-C. Son, "The Kangaroo Approach to Data Movement on the Grid," *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, p. 325, 2001.

M. Swany, "Improving Throughput for Grid Applications with Network Logistics," *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, p. 23, 2004.

H.-Y. Hsieh and R. Sivakumar. "pTCP: An End-to-End Transport Layer Protocol for Striped Connections," in *ICNP 02: Proceedings of the 10th IEEE International Conference on Network Protocols*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 24–33.

H. Bullot, R. L. Cottrell, and R. Hughes-Jones. "Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks," in *Journal of Grid Computing*, vol. 1, no. 4, pp. 345–359, 2004.

W. Allcock and T. Perelmutov. "GridFTP v2 Protocol Description," GridFTP Working Group, Tech. Rep., May 2005.

H. Stockinger. "Defining the grid: a snapshot on the current view," in *Journal of Supercomputing*, vol. 42, no. 1, pp. 3–17, 2007.

Y. Gu and R. L. Grossman. "UDT: UDP-based Data Transfer for High- Speed Wide Area Networks," in *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, no. 7, pp. 1777–1799, 2007.

Author Biographies

Tilmann Rabl is a research assistant at the University of Passau. He received his master degree in computer science in 2006 at the University of Passau. Currently he works as a assistant lecturer at the Chair of Distributed Information Systems. His research interests include distributed databases and multimedia information systems. He is author of several research papers at international journals, conference proceedings as well as chapters of books.

Christoph Koch is a former master student at the University of Passau. He graduated with a thesis on FSP at the Chair of Distributed Information Systems in 2007 and now works for a German software company. His research interests include distributed systems with a special focus on network protocols.

Günther Hölbling is a research assistant at the Chair of Distributed Information Systems, University of Passau. He received his master degree in computer science at the University of Klagenfurt in 2006. His current research focuses on service-oriented architectures, mobile information systems and distributed systems.

Prof. Dr. Harald Kosch is a full professor and the head of the Chair of Distributed Information Systems. His research topics are multimedia metadata, multimedia databases, middleware, and Internet applications. He successfully participates in the MPEG-7 and MPEG-21 standardization process, is one of the co-founders of the multimedia metadata community, and owns several patents. He has been organizing many international conferences and workshops covering different aspects of multimedia engineering (e.g., multimedia database workshops at DEXA, Grid workshop at the VLDB 2007 and the Multimedia Information Track at IEEE SITIS 2006). He is the author of several books and publishes regularly in refereed international journals and conference proceedings (more than 60 peer-reviewed publications).