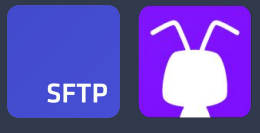
SFTP vs. FTPS Benchmarks: A Technical Performance Comparison

A Crazy Ant Labs whitepaper



Crazy Ant Labs

SFTP vs. FTPS Benchmarks: A Technical Performance Comparison (2025)

By Crazy Ant Labs

Executive Summary

This whitepaper presents a framework for evaluating the performance of SFTP and FTPS through 14 benchmark scenarios that varied file size, file count, latency, and client behavior. The results show that performance is not a fixed property of one protocol versus the other, but the outcome of multiple interacting factors.

FTPS consistently outperformed SFTP in curl-based downloads once the file sizes reached one megabyte or more, but at 100-kilobyte file sizes, SFTP was faster. On uploads with curl, SFTP led at the smallest sizes, between 100KB and 1MB, while FTPS again pulled ahead as file sizes grew, particularly beyond 5MB. With LFTP, SFTP improved markedly and often erased or reversed FTPS's advantage, most clearly in same region uploads, where SFTP outpaced FTPS across all file-counts.

Curl did not always deliver lower transfer time values than LFTP, in fact, some small file downloads ran slower with LFTP than with curl, showing that client behavior is a complex interaction with both workload and protocol.

Overall, however, the benchmarks show that no single factor explains protocol performance. We tested five parameters: file size, file count, latency, transfer direction (upload vs. download), and client, and each of these factors had a clear and dramatic impact. Depending on the combination, results shift, and in some cases protocol rankings reverse entirely. With a different client than those tested, the results could look quite different.

The core takeaway is that no protocol is universally faster. If transfer performance matters to you, then instead of choosing based on protocol alone, you need to take all these factors into account. There are too many factors to support an educated guess, especially if you're using another client, so the best option is to test.

These benchmarks demonstrate a method that other teams can adapt: test using the file sizes, counts, client software, and regions that reflect your own workloads before making a decision.

Introduction

When we went looking for reliable data on how SFTP and FTPS actually perform, most of what we found were broad claims without numbers to back them up. That absence of metrics was the trigger for running our own controlled benchmarks. The goal was not only to measure raw transfer speed, but to show how workload factors such as file size, file count, client implementation, and latency shape results, and to provide a framework that other teams can use to test in their own environments.

SFTP and FTPS are both secure file transfer protocols, but their designs lead to distinct behaviors under load.

SFTP, built on SSH, operates through a single persistent session and supports pipelined commands. While most clients don’t take advantage of concurrent requests within a session, it is technically possible, as shown in AsyncSSH. FTPS, in contrast, extends FTP with TLS and uses separate control and data channels, opening new data channels as needed for transfers and listings. These architectural choices shape their performance characteristics under different workloads and network conditions.

To illustrate this, we conducted a benchmark campaign across a range of scenarios. Each varied one or more dimensions: file size, file count, latency (same-region versus cross-region), and client software (curl versus LFTP). The purpose was not simply to measure raw transfer speed, but to show how

performance shifts with changing conditions and to demonstrate a method that can be replicated.

The following sections document the results, identify the patterns that emerged, and explain how the same framework can guide protocol selection for use cases such as batch jobs, automation pipelines, distributed workflows, or large-scale data transfers.

---

1. Methodology

To fairly compare SFTP and FTPS, we designed a set of scenarios focused on transfer performance across real-world variables: file size, file count, client implementation, and network latency. We ran a total of 80 tests, 40 per region, run multiple times. The same approach can be adapted by other teams to evaluate protocol performance under their own workload conditions.

Environment setup

- Clients: m7a.large EC2 instances in AWS us-east-1(same-region, low-latency) and eu-central-1 (cross-region, high-latency).
- Server: SFTP To Go endpoint in AWS us-east-1, used in all tests.

File structure

- Test batches varied in size and count:
    - File sizes ranged from 100KB to 100MB
    - File counts ranged from 1 to 100 files.
- All files were randomly generated to prevent compression from distorting results.

Tools and protocols

- We used two command-line clients: curl and LFTP, both supporting SFTP and FTPS.
- Each test used only a single connection per session, and no parallel threads.
- Post-transfer commands like FTPS timestamp synchronization were disabled to avoid skewing results.

- All tests were run against a live managed cloud SFTP To Go endpoint under real-world conditions.

Execution

- Scripts were written to cover all combinations of protocol (FTPS and SFTP), client (curl and LFTP), and transfer direction (upload and download). These scripts were then executed with different source file sets (varying in size and count) and run on the two EC2 instances in us-east-1 and eu-central-1. This ensured consistency across scenarios and made the results directly comparable.
- All tests were run multiple times, and the results were averaged to account for natural network variance.
- Transfers were executed serially, one file after another, to emphasize protocol-level behavior rather than complicating the benchmarks with another variable like concurrent transfers.

This methodology ensured the benchmarks were directly comparable and reflective of how SFTP and FTPS behave under different constraints.

---

2. Same Region (us-east-1-us-east-1)

To establish a controlled baseline, we began with transfers within the same AWS region using curl. This allowed us to isolate protocol behavior under low-latency, low-noise conditions and observe how SFTP and FTPS perform when distance and client optimization are not variables. From there, we switched clients and conditions, finally adding cross-region latency to the mix

Table 1: Same-region, same volume

us-east-1 to us-east-1, same batch volume:

Avg Execution time (seconds)		Protocol							
		FTPS				SFTP			
		Download		Upload		Download		Upload	
Number of files	File size (bytes)	curl	lftp	curl	lftp	curl	lftp	curl	lftp
100	1,000,000	32.0	33.7	30.8	30.4	40.9	17.9	21.4	16.3
20	5,000,000	11.5	12.9	8.6	10.2	33.9	11.1	11.9	7.2
10	10,000,000	9.0	10.2	7.3	9.0	33.4	9.1	12.0	6.4
1	100,000,000	6.5	8.2	2.8	4.7	35.3	9.0	9.5	4.0

Test setup

- **Client Region:** us-east-1
- **Server Region:** us-east-1
- **Clients:** curl and LFTP
- **Protocols**: SFTP vs. FTPS
- **Transfer:** Fixed total data volume, distributed across different file counts (100 × 1MB, 20 × 5MB, 10 × 10MB, 1 × 100MB).

Measured results

Given that the amount of data transferred is the same (total of 100MB) it is interesting to note the difference in execution time across the various tests, which indicates that there is a certain overhead with each file operation.

Client influence

It is apparent that curl's implementation of SFTP impacts download times vastly and that all in all LFTP performs better with the SFTP protocol. When comparing execution time with the FTPS protocol, curl performs better.

Protocol interpretation

It appears like SFTP runtimes are lower across the board if we take the SFTP over curl out of the equation, except when uploading or downloading a single large file - in which case, FTPS shines.

Operational takeaway

Both protocols work better with fewer, larger files, and so transfer of archives of files will perform better than many small files.

Table 2: Same-region, fixed file count

us-east-1 to us-east-1, same batch file count:

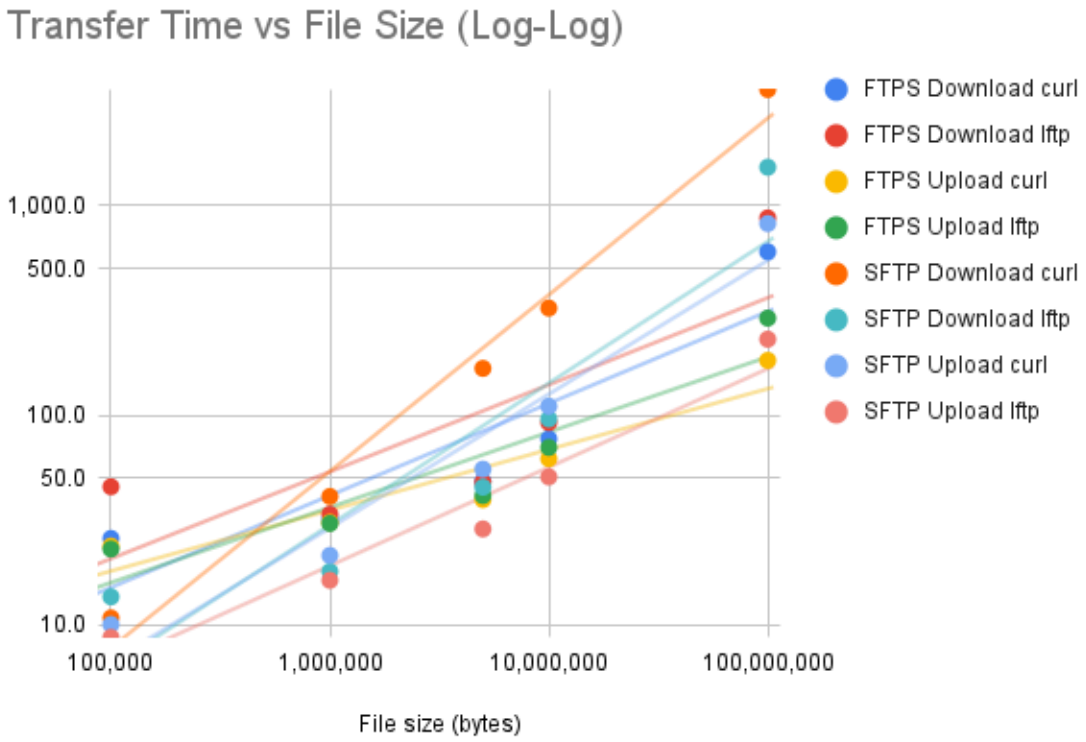
Avg Execution time (seconds)		*Protocol*		*Direction Client*					
		FTPS				SFTP			
		Download		Upload		Download		Upload	
Number of files	*File size (bytes)*	curl	lftp	curl	lftp	curl	lftp	curl	lftp
100	100,000	25.8	45.5	23.7	23.0	10.8	13.6	10.0	8.7
	1,000,000	32.0	33.7	30.8	30.4	40.9	17.9	21.4	16.3
	5,000,000	48.4	47.9	39.5	41.3	167.3	45.3	55.2	28.6
	10,000,000	77.2	92.1	61.9	70.1	324.8	95.9	110.4	50.7
	100,000,000	603.4	875.0	182.6	290.8	3,593.7	1,528.4	823.1	230.6

Test setup

- **Client Region:** us-east-1
- **Server Region:** us-east-1
- **Clients:** curl and LFTP
- **Protocols:** SFTP vs. FTPS
- **Transfer:** Fixed file count (100), file size increased progressively (100KB, 1MB, 5MB, 10MB, 100MB).

Measured results

Execution time growth seems to grow with different slopes amongst the different protocol/direction and client selections:



Protocol interpretation

SFTP shows better performance with smaller files, but as file size increases, FTPS becomes faster.

Client influence

It is apparent that curl's implementation of SFTP impacts download times vastly. Otherwise, there are cases where LFTP performs better than curl - notably with SFTP, but also with FTPS in some of the tests.

Operational takeaway

It appears that when comparing growing volumes of data over the same number of files, FTPS is the stronger protocol for larger files. SFTP remains a faster solution for smaller files and a practical option for moderate file sizes but should not be the default for very large objects unless you prioritize stability over raw throughput.

For operations teams, this means factoring in expected file growth: the protocol you choose today for small to medium sized files may act differently once your

datasets scale to hundreds of megabytes or beyond.

---

3. Cross Region (eu-central-1-us-east-1)

Table 3: Cross-region, same volume

Cross-region, same batch volume:

Avg Execution time (seconds)			*Protocol*	*Direction Client*							
			FTPS				SFTP				
			Download		Upload		Download		Upload	
Number of files	*File size (bytes)*	*Region*	curl	lftp	curl	lftp	curl	lftp	curl	lftp
100	1,000,000	us	32.0	33.7	30.8	30.4	40.9	17.9	21.4	16.3
		eu	218.9	224.4	121.7	128.8	161.9	67.6	331.9	45.7
20	5,000,000	us	11.5	12.9	8.6	10.2	33.9	11.1	11.9	7.2
		eu	80.5	80.3	30.9	32.6	134.9	40.2	307.5	27.6
10	10,000,000	us	9.0	10.2	7.3	9.0	33.4	9.1	12.0	6.4
		eu	64.7	57.0	20.7	21.2	131.3	39.1	306.2	25.5
1	100,000,000	us	6.5	8.2	2.8	4.7	35.3	9.0	9.5	4.0
		eu	16.3	19.4	7.6	7.5	129.6	33.9	300.2	21.3

Test setup

- **Client Region:** eu-central-1
- **Server Region:** us-east-1
- **Clients:** curl and LFTP
- **Protocols:** SFTP vs. FTPS
- **Transfer:** Same batch volumes tested across regions, with file counts and sizes varied.

Measured results

Latency highlighted client and file count effects. With curl, FTPS uploaded faster at high file counts; with LFTP, SFTP led. FTPS still led on very large single-file transfers in both clients.

Protocol interpretation

SFTP uses a single persistent SSH session that can pipeline requests, so it tends to stay steady as file counts rise, especially under latency. FTPS keeps a

persistent control connection but opens a separate data channel for each transfer or listing. Even when TLS sessions are reused, that per-file open/close and command sequencing adds overhead at high file counts. For single large files, FTPS still often leads once transfer gets going.

Client influence

LFTP reduced latency impact and flipped outcomes on multi-file uploads in favor of SFTP; FTPS still dominated single large-file transfers.

Operational takeaway

For distributed teams or cross-region workflows, the choice of protocol should reflect both file size and count per operation and tolerance for latency. SFTP is better suited to telemetry, logs, analytics pipelines, and any workload that generates frequent small or mid-sized files. FTPS is still viable for replication of large monolithic datasets or backups across regions but should be approached with caution if file counts can't be controlled.

In practice, this means engineering teams should probably opt for SFTP as the default for cross-region traffic, because the performance difference is between 90%-114% while with FTPS the difference is between 1300%-1400%, unless you're only doing single file operations.

7. Observations and Analysis

The benchmarks confirm that transfer performance is shaped by more than protocol choice. File size, file count, direction, latency, and client software all had measurable impact, sometimes changing which protocol came out ahead. To make informed decisions, teams should apply the following framework, using their own workloads as the test case:

- **Start with your workload.** If you expect to handle many small files, plan tests with SFTP under your own conditions, as its pipelined model is usually better suited for high file count uploads. If your operations focus on moving a few large files, benchmark FTPS in your environment, particularly for downloads, since it often sustains higher throughput there.
- **Factor in direction.** Transfer direction (upload versus download) influences protocol behavior, but it's not something most teams can control. What you *can* choose are the protocol (if your platform supports more than one), the client, and the server endpoint location. File size and

count can sometimes be shaped on upload by aggregating files, but if you're only downloading, those characteristics are usually fixed. The practical approach is to recognize which levers you control and run your tests accordingly, rather than assuming results will generalize.

- **Account for geography.** If your transfers span regions, test how each protocol behaves under your latency conditions. Across regions, FTPS slows as file counts rise with curl, while with LFTP SFTP holds steadier. Run cross-region trials that match your expected workloads before committing.
- **Test clients, not just protocols.** Client choice can double performance or even reverse protocol rankings. For example, switching from curl to LFTP has transformed SFTP outcomes in several scenarios. Always benchmark with the tools you plan to deploy.
- **Consider supporting both.** Workflows are rarely uniform. Maintaining both SFTP and FTPS endpoints allows teams to match protocol to workload, whether that means small automated exports, large archival transfers, or distributed multi-region pipelines.

By structuring decisions around these dimensions (workload shape, transfer direction, geography, and client) teams can avoid assumptions and select the combination of client and protocol that best fits their real conditions.

---

8. Conclusion

These benchmarks demonstrate that no single protocol is always faster. Performance is shaped by workload composition, client software, and geography. The practical lesson is that teams cannot outsource this decision to reputation or assumptions; instead, they need to test in their own conditions.

The observations and framework outlined in this paper offer a practical starting point for evaluation. By structuring evaluations around workload shape, transfer direction, latency, and client choice, teams can run small but representative tests before committing to a protocol for production pipelines. The same scripts and setup we used can be adapted to replicate real-world conditions, whether

that means simulating thousands of log files, a nightly backup archive, or multi-region data syncs.

The larger insight is that protocol is only one lever. Client choice alone shifted outcomes in multiple scenarios, sometimes flipping the “faster” protocol entirely. This means that optimization opportunities may come from switching tools rather than redesigning workflows.

Finally, for organizations with diverse workloads, the safest path is often to support both protocols. Maintaining SFTP and FTPS endpoints gives teams the flexibility to align protocol to use case: FTPS for large single-file transfers (especially across regions), and SFTP for high-count uploads and automation-driven tasks (where it remained faster in both same-region and cross-region tests).

By approaching file transfer as an engineering decision rather than a static rule, teams can select tools with confidence, ensure predictable performance, and avoid the pitfalls of one-size-fits-all assumptions.

---

9. Areas for Further Benchmarking

For teams designing or refining production workflows, extending these tests in a targeted way can provide even more confidence.

Practical areas to explore include:

- **Concurrency.** Most real workloads involve multiple transfers running at once. Testing concurrent uploads and downloads will show how protocols behave under saturation, especially if network or compute resources are limited.
- **Resilience.** Simulating interruptions and resuming transfers can highlight which protocol or client handles retries and partial file resumes more efficiently. This is particularly relevant in unstable or bandwidth-constrained environments.

- **Large-scale datasets.** While our tests capped files at 100MB and batches at 100 files, production workflows often exceed those numbers. Running multi-gigabyte or 10,000+ file tests can reveal scaling limits that matter at enterprise scale.
- **Client diversity.** Extending tests to graphical tools (e.g., FileZilla, Cyberduck, WinSCP) or SDKs (Python, Node.js) and integration platforms (e.g., Mulesoft, Talend, Integrate.io, Make.com, Microsoft SSIS, etc.) helps assess how protocol performance translates outside of CLI environments.
- **System integration.** Benchmarking protocols inside end-to-end workflows, such as CI/CD pipelines, ETL jobs, or telemetry exports, provides a clearer picture of operational performance, including encryption overhead and CPU load.
- **Edge conditions.** Testing under constrained environments (low memory, CPU, or bandwidth) can show how transfers degrade and where optimizations matter most.

The value in extending benchmarking is not just more data. It is a way to build a decision framework tuned to your environment. By testing with your own workload shape, client stack, and operational constraints, you ensure that protocol and client choices support both performance and reliability in practice.

---

10. Appendix:

Test scripts

The Github repository containing test scripts and additional data can be found here: https://github.com/crazyantlabs/sftp-ftps-benchmarks

Raw benchmark data

The raw benchmark data can be found here.