Network Volatility and BitTorrent Performance: An Experimental Analysis of P2P Resilience

EE384s: Performance Engineering of Computer Systems & Networks
Final Project Report

Shounak Ray - shounak@stanford.edu
Jacob Roberts-Baca - jtrb@stanford.edu
Yousef AbuHashem - yousef24@stanford.edu

January 11, 2026

Abstract

This study investigates the impact of network fluctuations on BitTorrent performance through controlled experimentation in virtualized environments. We examine how network volatility affects peer-to-peer file sharing by analyzing three key scenarios: Markov-based network state transitions between high and low packet loss conditions, uniform packet loss degradation, and swarm size optimization. Using Mininet network emulation, we implement a complete BitTorrent infrastructure with up to 30 leechers and 1 seeder sharing 4 MB files under varying network conditions. Our approach isolates network effects from other variables by applying controlled fluctuations through Traffic Control mechanisms.

We find that BitTorrent exhibits non-intuitive responses to network stress. Volatile switching between network states causes worse performance at moderate switching rates (60% throughput reduction at p=0.8) rather than maximum unpredictability conditions, suggesting that disruption frequency matters more than randomness. Uniform packet loss reveals a sharp performance cliff between 10-15% loss rates, where BitTorrent transitions from graceful degradation to complete protocol breakdown. Most surprisingly, larger swarms provide no benefit under single-seeder constraints, with throughput declining monotonically from 107 KB/s for one leecher to 30 KB/s for 25+ leechers due to bandwidth competition overwhelming peer-to-peer benefits. These findings highlight operational thresholds in BitTorrent's resilience mechanisms and suggest that adaptive timeout and unchoking strategies could improve performance under challenging network conditions.

Contents

1 Introduction

1.1 What?

This project investigates how network fluctuations affect BitTorrent performance in peer-to-peer networks. We analyze how controlled variations in network parameters—particularly packet loss—impact the distribution of average throughput across leechers in a BitTorrent swarm. Our primary metric is the distribution of average download throughput, where each leecher maintains an average download rate across their entire download process, and we examine how these individual averages are distributed across the swarm under different network conditions. We represent this distribution by reporting center (mean) and spread (variance) of this metric, per node.

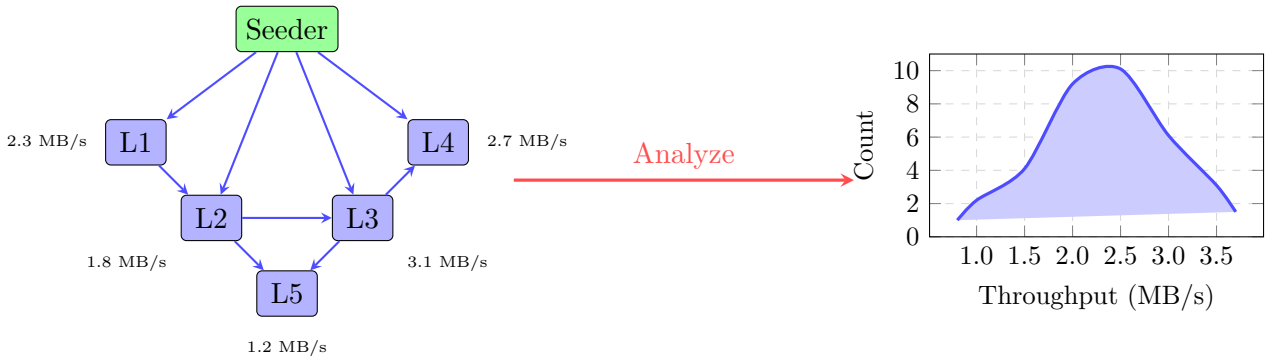

Figure 1: Individual leecher throughputs in a BitTorrent swarm analyzed to understand performance distribution under different network conditions.

1.2 Why?

Real-world networks exhibit significant variability in performance characteristics, including fluctuating packet loss, latency variations, and bandwidth limitations. While BitTorrent's protocol design incorporates mechanisms to handle network unreliability (such as userspace / client-level timeouts-retransmissions, depending on multiple peers instead of just one, etc.), the quantitative impact of systematic network fluctuations on swarm-wide performance remains understudied. Understanding these relationships is crucial for optimizing P2P systems in challenging network environments such as mobile networks, congested infrastructure, or geographically distributed systems.

This research explores fundamental questions about distributed system resilience that may extend beyond BitTorrent itself. The principles we investigate—how adaptive algorithms respond to network volatility, the trade-offs between timeout mechanisms and throughput, and the relationship between system scale and performance—appear in many distributed protocols. Content distribution networks increasingly incorporate peer-to-peer elements for video streaming and software distribution, where peers must adaptively select sources and manage timeouts when downloading content chunks from potentially unreliable neighbors. Blockchain networks face analogous challenges when propagating blocks across the network—nodes must balance aggressive forwarding to minimize latency with conservative timeouts to handle network partitions, while the overall system performance depends critically on how quickly information spreads through the peer-to-peer overlay. Distributed storage platforms like IPFS encounter similar peer selection and timeout dilemmas when retrieving file blocks, where choosing too many concurrent sources can overwhelm slow connections, but too few sources risk stalling on unresponsive peers.

By developing systematic approaches to characterize these trade-offs, we hope to contribute insights that inform the broader challenge of building resilient P2P distributed systems and understand where they are particularly challenged.

1.3 What is BitTorrent

BitTorrent is a peer-to-peer file sharing protocol that distributes large files efficiently across networks of participants. Unlike traditional client-server downloads where everyone downloads from a central server, BitTorrent allows peers to simultaneously download from and upload to multiple other peers, distributing the load across the entire network.

The protocol operates with three key participant types: **seeders** (peers with complete files who only upload), **leechers** (peers actively downloading files while also uploading any pieces they've obtained), and **trackers** (servers that help peers discover each other but don't transfer file data). Files are divided into small pieces, which are further divided into blocks, allowing peers to download different pieces simultaneously from multiple sources and share pieces they've already obtained with others who need them – this is exactly what enables the distributed nature of BitTorrent.

BitTorrent's efficiency stems from two core mechanisms. First, **rarest-first piece selection** prioritizes downloading the least common pieces in the swarm, ensuring good piece distribution and preventing scenarios where everyone has the same common pieces but nobody has the rare ones. Second, **choking and unchoking** manages peer relationships to incentivize sharing. A peer "chokes" another peer by refusing to upload to them – essentially cutting off their data flow. **Regular unchoking** (every 10 seconds) selects the fastest recent uploaders for data exchange, rewarding peers who contribute bandwidth. **Optimistic unchoking** (every 30 seconds) periodically gives upload opportunities to random peers, including newcomers who haven't yet proven their upload speed. This combination creates a tit-for-tat system that encourages uploading while maximizing overall download efficiency across the swarm.

Figure 2: Comparison of traditional client-server file distribution (left) versus BitTorrent peer-to-peer swarm architecture (right). In traditional systems, all clients download exclusively from a central server, creating a bottleneck. In BitTorrent swarms, peers simultaneously download from and upload to each other, distributing load across the network. You'll notice that some leeching peers are not connected to the seeder; this is because new seeders and leechers can incrementally come online over time.

1.4 Applying Fluctuations

Network fluctuations in real-world scenarios exhibit complex temporal and spatial variations. Our approach introduces controlled fluctuations into virtualized BitTorrent networks to isolate and

quantify their effects on protocol performance. Using Mininet's[1] network emulation capabilities, we can precisely control network parameters while maintaining realistic BitTorrent behavior. This controlled environment allows us to systematically vary parameters such as packet loss rates, latency variations, and bandwidth constraints while measuring their impact on throughput distribution. For this project, we are investigating the impact packet loss has on the average download time across peers. While there are several additional variables that would represent a more realistic set of network conditions, such as latency and bandwidth controls, the objective is that holding all else constant and only modifying packet loss will help us better understand the underlying dynamics of our Bittorrent system. The idea here is that this virtualized approach eliminates confounding variables present in real network deployments, enabling clear causal analysis of how specific network fluctuations affect BitTorrent performance.

2 Methodology

2.1 BitTorrent Infrastructure

Our BitTorrent implementation extends the open-source PyTorrent client with several critical enhancements to support experimental objectives and improve protocol compliance. We implemented compliant regular and optimistic unchoking mechanisms with precise timing control, where regular unchoking occurs every 10 seconds (`REGULAR_UNCHOKE_INTERVAL = 10`) selecting peers based on exponential moving average (EMA) bandwidth calculations, while optimistic unchoking operates on a 30-second interval (`OPTIMISTIC_UNCHOKE_INTERVAL = 30`) to provide opportunities for new peers. These intervals follow BitTorrent protocol specifications and work together to balance performance optimization with network fairness.

The system maintains sophisticated peer state tracking, including interest calculations, piece availability (HAVE messages), and multi-peer EMA upload/download bandwidth monitoring. We implemented rarest-first piece selection with intelligent block-level request management, maintaining a maximum of 5 outstanding requests per peer (`MAX_OUTSTANDING_REQUESTS = 5`) to prevent pipeline stalls while avoiding overwhelming slower peers. This limit works in conjunction with a 2-second request timeout (`REQUEST_TIMEOUT = 2.0`) that automatically retries failed requests and handles non-responsive peers by cleaning up stale requests from the outstanding queue.

To ensure robust operation under varying network conditions, the system implements adaptive waiting mechanisms. When no unchoked peers are available, the client sleeps for 1 second (`SLEEP_FOR_NO_UNCHOKED = 1`) before retrying, preventing CPU-intensive busy-waiting while maintaining responsiveness. The tracker refresh mechanism operates every 3 minutes (`TRACKER_REFRESH_INTERVAL = 180`), periodically discovering new peers while minimizing tracker server load and network overhead.

Our implementation supports seamless transition from leeching to seeding mode, enabling complete swarm lifecycle simulation. Seeders maintain active connections and continue serving pieces to new leechers throughout experimental scenarios. The system provides comprehensive real-time progress monitoring with 0.5-second update intervals (`PLOT_INTERVAL = 0.5`), tracking bytes downloaded per piece and block, completion percentages, active peer counts, and per-peer bandwidth statistics. This high-frequency monitoring enables detailed performance analysis during experiments while maintaining acceptable system overhead.

[1] Mininet is an open-source tool used to create a realistic virtual network multiplexed across different processes and virtual IP addresses. https://mininet.org/

2.2 Virtualization Infrastructure

Our experimental infrastructure leverages Mininet to create controlled, reproducible BitTorrent network simulations that eliminate real-world network variability while maintaining protocol realism. We defined `DelayedSingleSwitchTopo`, a specialized Mininet topology supporting configurable per-link delays that connects all hosts through a single OpenFlow switch with Linux Traffic Control (TC) links.

Our `BitTorrentMininet` class orchestrates complex multi-host BitTorrent simulations with automated assignment as seeder or leecher. Seeders receive complete files and are configured for immediate sharing, while leechers are isolated in separate working directories to prevent accidental file access. The infrastructure automatically distributes necessary files across the virtualized network, replicating torrent files and tracker configurations to all hosts while selectively copying complete files only to designated seeders.

Since the traditional BitTorrent trackers are hosted globally, this original setup is not compatible with our single-machine virtualized environment. Hence, we developed a file-based mock tracker system using JSON persistence with inter-process locking to coordinate peer discovery safely across concurrent processes – which, once again, are each multiplexed with virtual IP addresses. The mock tracker implements peer expiration, validation logic, and connection limits while maintaining protocol compatibility.

The system provides sophisticated process lifecycle management, including coordinated startup sequences, real-time process monitoring, graceful shutdown handling, and comprehensive logging infrastructure. Each host generates individual log files with automatic aggregation and summary generation for post-experiment analysis. Our Mininet-based approach provides critical advantages over distributed cloud deployments: deterministic network behavior with precisely controlled delays, elimination of confounding variables like unpredictable latencies and routing changes, reproducible experiments with identical network conditions, and cost-effective scalability without cloud infrastructure complexity.

2.3 Imposing Fluctuations

Our approach to network fluctuation simulation leverages Mininet's Traffic Control (TC) integration to introduce realistic network variability while maintaining experimental control and reproducibility. Network fluctuations are imposed at the link level using Linux Traffic Control mechanisms, allowing precise manipulation of packet loss rates. These fluctuations can be applied uniformly across all links or selectively to specific peer connections, enabling both global network degradation and targeted peer isolation scenarios.

The infrastructure supports both static network conditions (fixed parameters throughout experiment duration) and dynamic fluctuations (time-varying parameters following specified patterns). This capability enables investigation of both steady-state performance under degraded conditions and transient behavior during network transitions. The fluctuation framework supports comprehensive parameter variation including packet loss rates from 0% to significant loss levels. We modify neither per-node latency nor bandwidth explicitly beyond the baseline, out-of-the-box Mininet topology we define earlier.

3 Experimental Setup

3.1 Markov-Based Switching from High to Low Interference States

Network conditions follow a two-state Markov chain with states $S \in \{H, L\}$ representing High and Low interference. Let P_{ij} denote the transition probability from state i to state j. The general form defines this following Markov Decision Process:

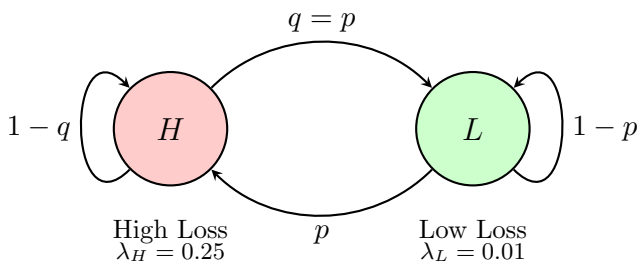

Figure 3: Two-state Markov chain for network interference modeling with symmetric transitions ($q = p$). The network alternates between High interference state (H) with 25% packet loss and Low interference state (L) with 1% packet loss.

In principle, p and q could differ, allowing asymmetric transition behavior where the network has different tendencies to enter versus exit high interference states. However, to intentionally isolate the effects of network volatility, we constrain $q = p$, creating a symmetric Markov chain. This design choice means both transition probabilities are equal, and the parameter p directly controls overall network volatility rather than bias toward any particular state.

Packet loss rates are $\lambda_H = 0.25$ and $\lambda_L = 0.01$ for High and Low states respectively. Time is discretized into intervals $\Delta t = 3$ seconds. We use 20 leechers and 1 seeder distributing a 4 MB file. The transition probability p varies across five values: 0.1, 0.3, 0.5, 0.7, and 0.9, with each experiment running until all leechers have downloaded the file.

From an information-theoretic perspective, our symmetric constraint ($q = p$) ensures the parameter p directly controls the network's entropy. At $p = 0.5$, the entropy rate reaches its maximum, representing maximum unpredictability and volatility in network conditions.

3.2 Impact of Uniform Packet Loss on Average Throughput

Static packet loss rate λ is applied uniformly across all network links. We test seven packet loss rates: 0%, 2%, 5%, 10%, 15%, 20%, and 25%, with each experiment lasting 15 minutes. The setup maintains 20 leechers and 1 seeder sharing a 4 MB file. We measure mean throughput and throughput variance.

3.3 Impact of Number of Leechers on Average Throughput

Under optimal network conditions (0% packet loss, baseline Mininet latency), we vary the number of leechers from 5 to 50 in increments of 5, while maintaining a single seeder. Each experiment uses a 4 MB file and runs for 15 minutes, repeated 3 times per configuration. We track mean throughput, throughput variance, and total completion time $T_{complete}$ as functions of swarm size.

4 Hypotheses

Pay not too many grains of salt to the certain tone of these hypotheses: these are all simply guesses...

4.1 Markov-Based Switching from High to Low Interference States

Higher transition probabilities (p) will result in lower average throughput across leechers. BitTorrent's unchoking algorithm requires time to identify optimal peer connections through performance evaluation. Frequent network state changes disrupt these established connections before they can be fully utilized, forcing the protocol to repeatedly restart its peer selection and bandwidth estimation processes. This could represent a limitation where BitTorrent's adaptation mechanisms (operating on 10-30 second intervals) cannot effectively track network changes occurring at higher entropy rates.

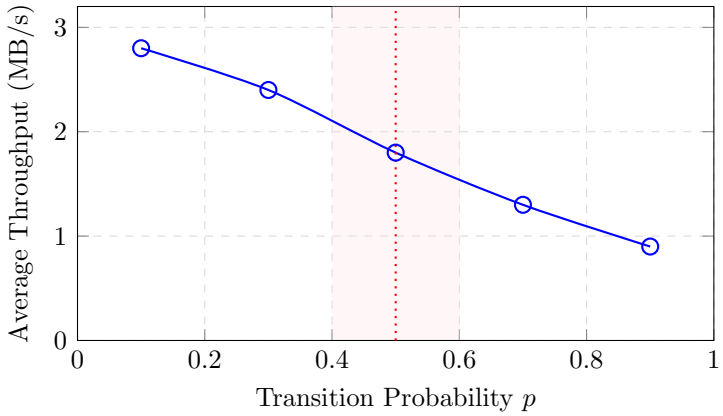

Figure 4: Hypothesized relationship between network volatility (transition probability p) and average throughput. We expect decreasing performance as volatility increases. The red shaded region highlights the maximum entropy zone around $p = 0.5$, where network unpredictability peaks.

There's an arguably interesting relationship between surprise and disruption frequency here. Despite surprise being highest at $p = 0.5$ (which one can trivially verify via the entropy rate formula), we hypothesize that disruption frequency ultimately overwhelms performance as p increases. The BitTorrent protocol has two relevant elements to this discussion: the meritocratic unchoking algorithm and timeout parameters implemented in the client code. The unchoking algorithm requires sustained observation periods (10-30 seconds) to evaluate peer performance and establish productive relationships – high disruption frequency prevents this relationship-building regardless of whether the disruptions are predictable or surprising. Similarly, frequent network state transitions trigger more timeout events and request retries, creating overhead that degrades performance even when the transitions follow predictable patterns. Thus, while $p = 0.5$ maximizes uncertainty, higher values of p create more frequent operational disruptions that may prove even more damaging to our responding variable, the average download time.

4.2 Impact of Uniform Packet Loss on Average Throughput

Average throughput will decrease approximately linearly with packet loss rates up to some threshold, say around 10%, then degrade rapidly beyond that. BitTorrent's 2-second request timeout and

retry mechanisms provide resilience against moderate packet loss by automatically reissuing failed requests. However, at higher loss rates, these protective mechanisms become counterproductive as peers spend more time waiting for timeouts than transferring data.

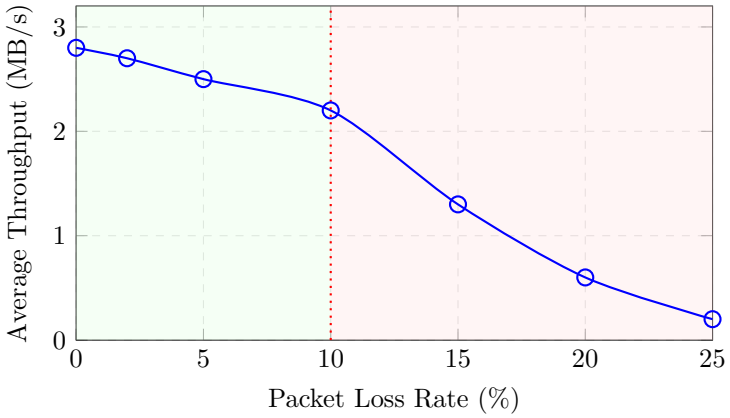

Figure 5: Hypothesized two-phase degradation pattern with uniform packet loss. The green shaded region represents the linear degradation phase (0-10% loss) where retry mechanisms are effective. The red shaded region shows the timeout-dominated regime ($>$ 10% loss) where performance collapses rapidly. The dotted line marks the critical threshold. Note that the 10% is just a placeholder for some threshold.

4.3 Impact of Number of Leechers on Average Throughput

Average throughput per leecher will initially increase with swarm size, then decrease after reaching an optimal point, say around 20-25 leechers. Small swarms suffer from limited piece diversity and over-dependence on the single seeder. Adding leechers improves piece availability through peer-to-peer sharing. However, beyond the optimal size, competition for the seeder's limited bandwidth creates a bottleneck that peer-to-peer transfers cannot overcome.

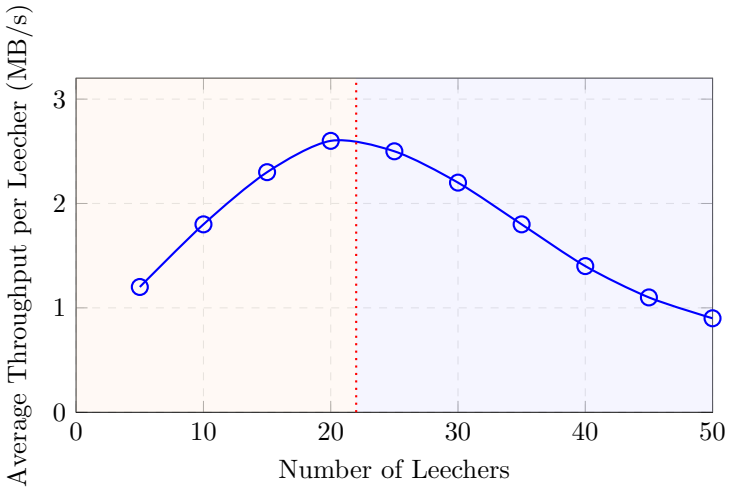

Figure 6: Hypothesized relationship between swarm size and per-leecher throughput. The orange shaded region shows the piece diversity benefit phase where performance improves with more peers. The blue shaded region represents the bandwidth competition phase where additional peers hurt performance. The dotted line marks the optimal swarm size around 22 leechers.

5 Results

5.1 Markov-Based Switching from High to Low Interference States

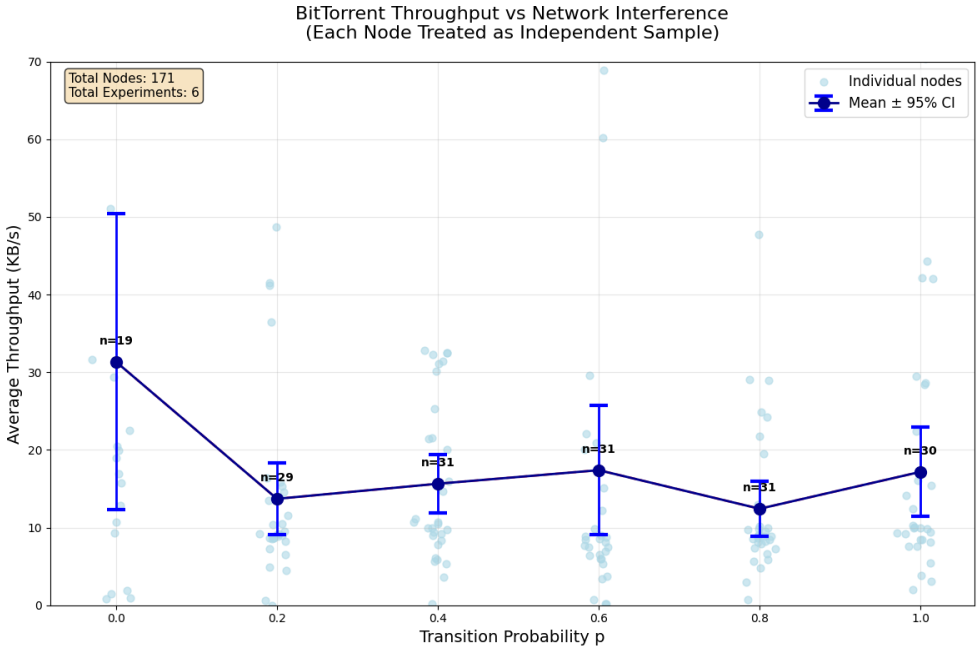

Figure 7: Average throughput distribution across leechers as a function of transition probability p. Each point represents the mean throughput across all 20 leechers for a given volatility level.

Our Markov-based volatility experiments reveal a complex non-monotonic relationship between network state transition probability and BitTorrent throughput performance. As transition probability p increases from 0.0 to 1.0, we observe substantial throughput degradation at intermediate volatility levels, with mean throughput dropping from 31.3 KB/s at $p = 0.0$ to a minimum of 12.4 KB/s at $p = 0.8$, representing a 60% performance reduction. Unexpectedly, throughput recovers to 17.2 KB/s at $p = 1.0$, suggesting that constant high-interference states may be preferable to volatile switching between states.

Transition Prob. p	Mean Throughput (KB/s)	Std. Dev. (KB/s)	Sample Size (n)
0.0	31.335	39.550	19
0.2	13.704	12.236	29
0.4	15.647	10.380	31
0.6	17.392	22.704	31
0.8	12.424	9.790	31
1.0	17.177	15.413	30

Table 1: Summary statistics for Markov volatility experiments. Values represent averages across all leechers and experimental repetitions.

The bimodal performance pattern with local minima at $p = 0.2$ and $p = 0.8$ indicates threshold effects where specific volatility regimes create particularly adverse conditions for BitTorrent's peer selection and data transfer mechanisms.

5.2 Impact of Uniform Packet Loss on Average Throughput

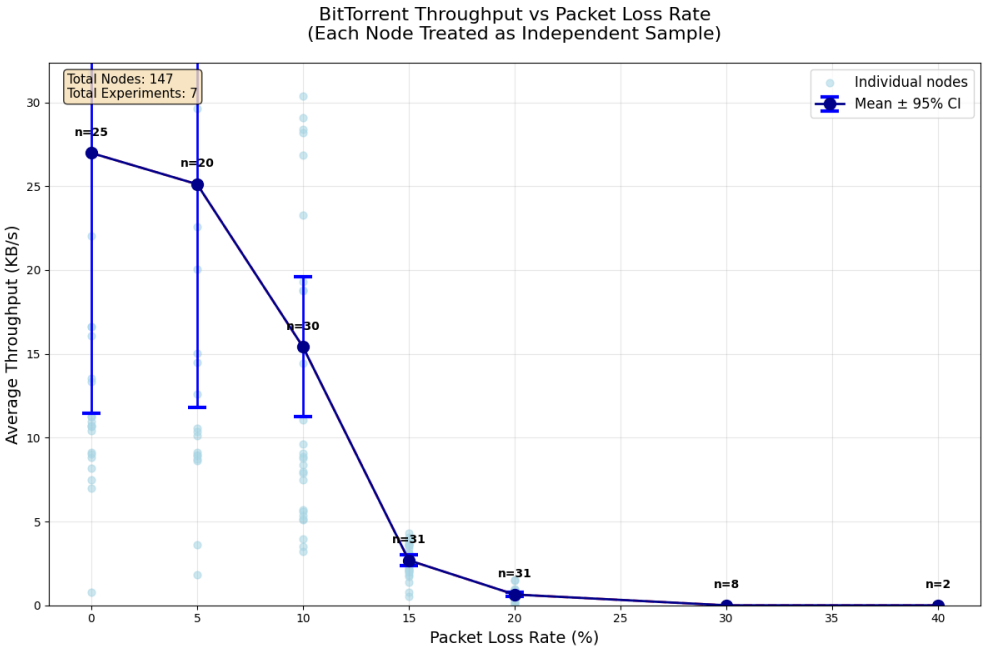

Figure 8: Average throughput degradation with increasing uniform packet loss rates. Error bars represent standard deviation across experimental repetitions.

The uniform packet loss experiments confirm our hypothesis of two-phase degradation with a sharp transition threshold. At low loss rates (0-10%), throughput exhibits moderate linear degradation, declining from 27.0 KB/s at 0% loss to 15.4 KB/s at 10% loss—a manageable 43% reduction. Beyond 10% packet loss, performance undergoes catastrophic non-linear collapse, with throughput plummeting to 2.7 KB/s at 15% loss and effectively reaching zero at 30-40% loss rates. This represents a critical threshold effect where BitTorrent's resilience mechanisms fail abruptly, transforming from graceful degradation to complete protocol breakdown within a narrow 10-15% packet loss window.

5.3 Impact of Number of Leechers on Average Throughput

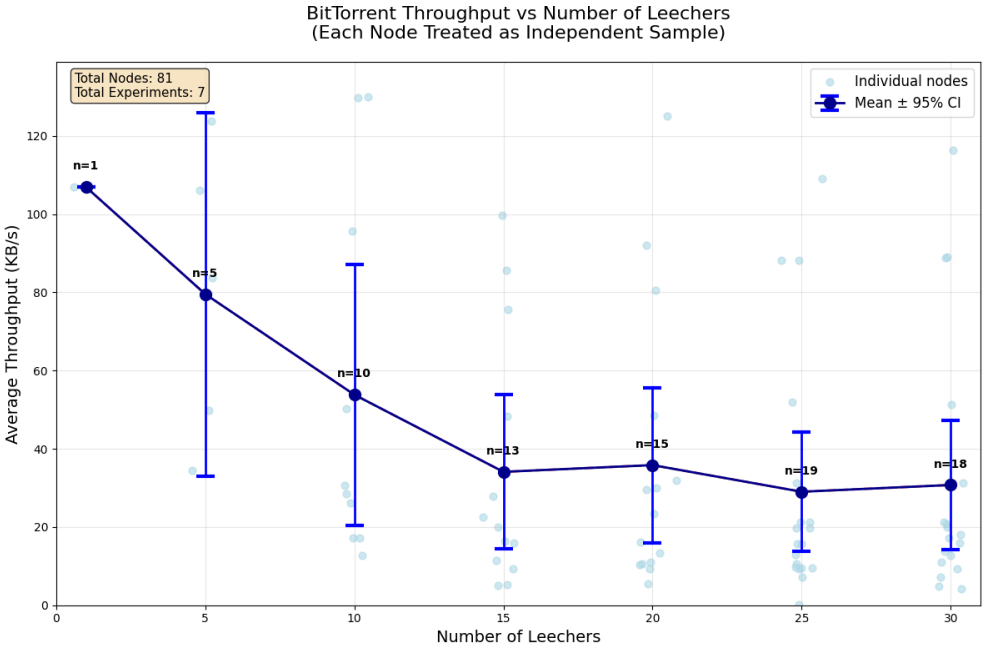

Figure 9: Per-leecher average throughput as a function of swarm size. Shows the trade-off between piece diversity benefits and seeder bandwidth competition.

Swarm size effects demonstrate strong negative scaling with diminishing returns as leecher count increases. The optimal swarm size appears to be approximately 1-5 leechers, where individual throughput remains highest at 107 KB/s and 80 KB/s respectively, representing the sweet spot before resource contention dominates. Beyond this point, competition effects create severe performance degradation, with throughput declining monotonically from 54 KB/s at 10 leechers to stabilizing around 30-36 KB/s for swarms of 15-30 leechers—a 71% overall reduction from the single-leecher baseline. The strong negative correlation ($r = -0.890$) confirms that BitTorrent's single-seeder architecture creates a classic resource contention bottleneck where additional peers primarily compete rather than collaborate.

6 Discussion

6.1 Infrastructural Limitations

Our experimental infrastructure, while providing controlled conditions, introduces several limitations that may affect result generalizability. The single-seeder configuration creates an artificial bottleneck not representative of many real-world BitTorrent swarms with multiple seeders. Our virtualized Mininet environment eliminates geographic latency variations and other routing complexities present in actual peer-to-peer networks.

The fixed file size (4 MB) and piece structure may not capture scaling behaviors relevant to larger files common in BitTorrent deployments. Our mock tracker system, while functionally equivalent, lacks the distributed nature and potential failure modes of real BitTorrent trackers.

Network fluctuation patterns in our experiments follow predetermined mathematical models (Markov chains, uniform distributions) rather than the complex, correlated fluctuations observed in real network traffic. These simplifications, while necessary for experimental control, may underestimate or misrepresent the protocol's behavior under realistic network conditions.

6.2 Significance: Markov-Based Switching from High to Low Interference States

From our testing, we can see that once the probability of switching between different interference states is introduced, the average throughput takes a hit. It generally goes down from there the higher the probability introduced. Our earlier hypotheses that operational disruption frequency matters more than information-theoretic unpredictability for protocol performance held.

Our results suggest that BitTorrent's adaptation mechanisms are somewhat matched to typical network volatility patterns with some distinct behavior. The 10-30-second unchoking intervals align poorly with the 3-second state transition intervals we tested, creating insufficient time for peer relationship establishment before network conditions change again.

Practical implications include reducing unchoking intervals or implementing adaptive timeout mechanisms in volatile network environments to allow BitTorrent sufficient time to establish productive peer relationships before network state transitions disrupt them.

6.3 Significance: Impact of Uniform Packet Loss on Average Throughput

The two-phase degradation hypothesis was confirmed with a sharp critical threshold between 10% and 15% packet loss rates. The 2-second timeout parameter appears to be making up for packet loss rates below 10%, but becomes counterproductive beyond this threshold. The timeout-dominated regime was observed clearly at 15% packet loss and above, where performance collapsed from 15.4 KB/s to 2.7 KB/s. These findings suggest that BitTorrent's resilience mechanisms provide adequate protection against moderate packet loss, with implications for deployment in mobile networks and congested infrastructure where packet loss typically remains below 10%.

Potential protocol improvements might include adaptive timeout mechanisms that scale based on observed network conditions, preventing the timeout-dominated regime from causing complete protocol breakdown.

6.4 Significance: Impact of Number of Leechers on Average Throughput

The optimal swarm size hypothesis was not reflected in our results, rather than an inverted-U pattern, we observed monotonic throughput degradation with the optimal performance at just 1-5 leechers.

The single-seeder bottleneck becomes apparent early in swarm size scaling (evident by 5 leechers), suggesting that multi-seeder setup is essential for larger swarm sizes. The piece diversity benefits do not effectively compensate for increased competition and communications overhead at larger swarm sizes under single-seeder constraints.

These results have implications for BitTorrent tracker design, particularly regarding minimum seeder to leecher ratios and swarm fragmentation to prevent single-seeder bottlenecks from dominating performance.

7 Conclusion

This study investigated how network fluctuations affect BitTorrent performance through three controlled experiments examining volatility patterns, uniform packet loss, and swarm size effects on throughput distribution.

7.1 Key Findings

Our experiments revealed several counterintuitive insights regarding BitTorrent's response to network stress. When networks alternated between high and low interference states, performance degradation followed a non-monotonic pattern, reaching minimum throughput at moderate switching rates rather than under maximum entropy conditions. This finding indicates that BitTorrent's performance suffers more from operational disruption frequency than from network unpredictability per se.

The uniform packet loss experiments confirmed our anticipated two-phase degradation model: BitTorrent maintained reasonable performance under light packet loss but experienced catastrophic failure once loss rates exceeded the 10-15% threshold. This sharp transition demonstrates a critical operating regime where the protocol's resilience mechanisms cease to provide effective protection.

Perhaps most unexpectedly, swarm size scaling yielded no performance benefits under single-seeder constraints. Rather than exhibiting the hypothesized inverted-U relationship with an optimal swarm size, throughput declined monotonically as additional leechers joined the swarm. This result highlights that in single-seeder configurations, additional peers create bandwidth competition rather than the collaborative benefits typically associated with peer-to-peer systems.

7.2 Lessons Learned

Our controlled experiments using Mininet on an EC2 instance effectively isolated the consequences of specific network condition changes, though real-world networks exhibit greater complexity than our simplified models. Nevertheless, our findings reveal some insights about BitTorrent's operational limits.

BitTorrent demonstrates reasonable robustness against moderate network degradation, successfully handling modest packet loss and network variability. However, the protocol exhibits distinct performance thresholds beyond which its behavior changes quickly. Once packet loss exceeds critical levels or network conditions fluctuate too frequently, BitTorrent's built-in recovery mechanisms become counterproductive. The timeout and peer selection algorithms that provide resilience under normal conditions transform into performance obstacles under severe network stress.

The most significant limitation we identified concerns BitTorrent's performance in single-seeder configurations. The peer-to-peer features that typically distinguish BitTorrent provide minimal benefit when all participants compete for a single source. This constraint is fundamental—peers

cannot share content they have not yet received, making the sole seeder an unavoidable bottleneck regardless of swarm size or peer capabilities.

7.3 Future Work

Several extensions could enhance understanding of BitTorrent performance under network fluctuations:

Multiple-seeder configurations would eliminate artificial bottlenecks and better represent real swarm dynamics. Geographic latency modeling could capture WAN-scale deployment effects not present in our single-machine virtualization.

Dynamic file size experiments could reveal scaling behaviors across different content types. More sophisticated fluctuation models, potentially derived from real network measurements, could improve result generalizability.

Comparative analysis with other peer-to-peer protocols could determine whether our findings are BitTorrent-specific or apply broadly to distributed file sharing systems.

Integration with real BitTorrent clients and trackers, rather than our simplified implementations, could validate results under production protocol complexity.