An Investigation of the BitTorrent Peer-To-Peer File Distribution Protocol's Behavior Via Simulation

Introduction

In a connected network such as the Internet that contains many nodes, there is perpetually a demand for the exchange of multimodal media from simple formats such as plain text to bandwidth-heavy video and industry software files. With this demand comes the need for hardware to adequately support the information superhighway without degrading the user experience significantly – this requires servers in data centers to not only securely store desirable content but to also distribute the content quickly to an unbounded amount of end hosts that can request it at any given moment. However, although the client-server file distribution architecture may work under moderate client and traffic conditions i.e. not too many clients requesting the same file all at once, centralizing a file distribution source suffers from drawbacks in practice since the Internet is a vast network – some key drawbacks are:

1. Download speeds do not scale beneficially with client count – the more clients request the file, the slower download speeds everyone gets since the server upload speed is limited – this is unacceptable for time-sensitive data.
2. There is a risk of the centralized server hardware failing, rendering clients downloading the file unserved for a period before finishing. Moreover, the data can be lost forever if the server stored the original source.
3. Server hardware can be expensive to implement and maintain, especially as requests grow with the Internet.
4. Download speeds can vary with the physical location of the client with respect to the centralized server as the data will have to take more router hops.

In contrast to the client-server architecture, the peer-to-peer (P2P) file distribution architecture takes advantage of the high node count of a large network to alleviate the bottlenecks mentioned above. A particular implementation of P2P file distribution is BitTorrent – it is a protocol that any host can use to communicate with other hosts to exchange file data and slowly accumulate a copy of the source file originally desired. In this project, several aspects of BitTorrent will be investigated via a simulator built in Python intended to approximate the behavior of a large Torrent, a group of nodes working together to distribute a file while following the rules of the protocol. We will execute many simulations runs on different parameters to represent scenarios BitTorrent can handle robustly as well as analyze the produced visual results of the swarm behavior.

Simulator Implementation

The BitTorrent behavior is simulated in Python by implementing the major components of the protocol. The nuances of the implementation are left to the reader to further investigate in the source code. For the sake of brevity, only the high-level features will be discussed.

Tracker

Every BitTorrent file starts out as a Tracker that can be accessed publicly to inform a host on 1. Meta information on the source file 2. The peers that are currently participating in the Torrent. In Python, this is implemented as a tuple that contains (list of file chunks, list of peer IPs). In practice, BitTorrent splits a source file into 256 Kb "chunks." Moreover, peer IDs are their public IPs which are unpredictable though unique. For the sake of our simulator, we can specify a parameter to set the source file's chunk count and create the chunks procedurally such that they are unique – a five-chunk file would have chunks {'1', '2', '3', '4', '5'}. Similarly, we can specify the number of peers (not necessarily all are in the tracker) by incrementing IP numbers starting from 0 – to add them to the tracker, simply append their IP to the list.

Peer

A Torrent's peer in practice must comply with the rules laid out by the protocol to reap the benefits. This translates to a set of functions that need to be implemented for each peer, calling for a "Peer" class to be created with certain internal variables and functions. The key, abstract variables are presented with the implementation-level ones not discussed:

Variable	Purpose
IP	For the simulator to know what a peer's unique identifier is for tracking its performance; also for other peers to lookup peers in a Torrent
Upload bandwidth	For a peer to know at what rate it can send data outwards
Tracker	For a peer to reference the source file and ascertain what chunks are still missing; also for lookup of available peers to potentially exchange with
Acquired chunks	For a peer to keep track of their download progress; also for other peers to know what chunks are available from others
Top 4 peers	For tit-for-tat exchange strategy that promotes fairness by guiding peer on which chunk requests from others to fulfill first

Function	Purpose
Join/leave tracker	To enter or leave a torrent at any given time
Request the rarest chunk	To ensure that the rarest chunk in the Torrent is acquired before the peer(s) holding it leave
Evaluate the top 4 peers	To gauge which peers are contributing chunks to oneself at the highest rates over others
Exchange chunks with top 4 peers	To prioritize peers that are contributing the most to oneself, a tit-for-tat strategy
Optimistically unchoke a peer	To randomly exchange chunks to a peer that historically has not been contributing and give it a chance at downloading

Once the above has been implemented and many peers have been added to a tracker, the actual swarm behavior of a Torrent should be approximated over time.

Time Domain Simulation

To simulate continuous time in a Torrent, we implemented a round-robin schedule in Python that operates on a unitless time variable t that increments in a for loop up until an end-simulation time. For example, if Peer 1 has an upload bandwidth period of 1, and Peer 2's upload period is 2, then for every simulator time step, the following routines would occur:

If t % Peer 1 upload period is 0	Peer 1 uploads
If t % Peer 2 upload period is 0	Peer 2 uploads

Additionally, if we want to reevaluate the top 4 peers every 100 time units:

If t % top 4 peers re-evaluate period is 0	All peers reevaluate top 4 peers

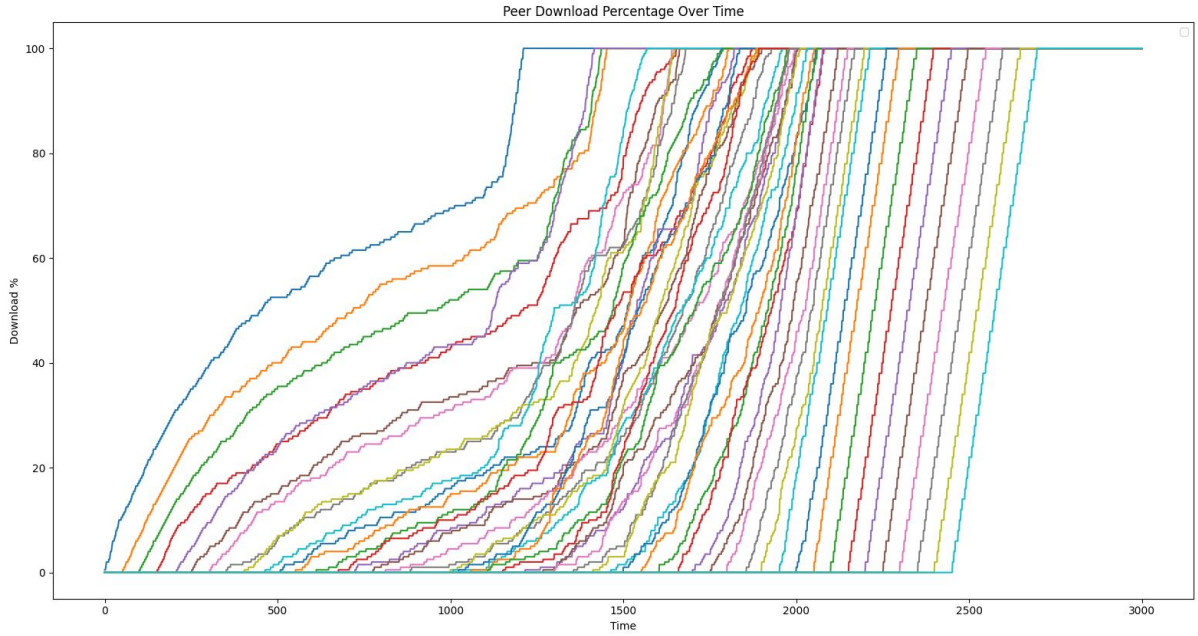
The round-robin schedule checks the periodic conditions for each routine so they can happen at specific intervals by using the modulo operator.

Main Investigation

Resilience

A beneficial facet of the P2P architecture is the perpetual circulation of file chunks within a Torrent as opposed to a source file being centralized in a server prone to being taken away from the network. We will investigate the effect of introducing a file into a Torrent via a source peer, peers leaving – churning – after finishing the download, peers staying after finishing to altruistically seed, and lastly how the Torrent survives after the source peer leaves.

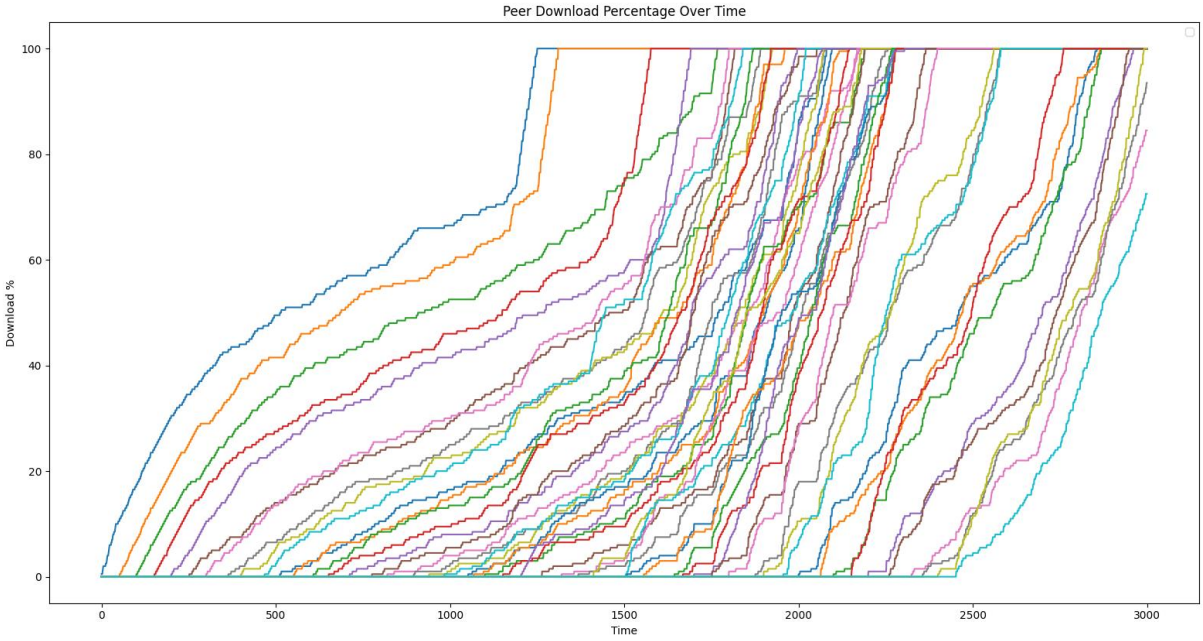
Torrent with Source Peer and Altruistic Peers



Above depicts the accumulated download progress of each peer in a Torrent that starts out with one source seeding peer. Thereafter, peers enter the Torrent at an interval of 50, each with no chunks to start with and equal upload bandwidth of 1000. Additionally, each peer that completes the download does not leave the Torrent. This is representative of a typical Torrent's conditions where there will be at least the seeding node and more peers will enter over time and seed once done. Observing the first peer that enters, we note two things: 1. As more peers enter after it, its download rate decreases but 2. Around time 1200, its download rate "explodes" and rapidly increases to completion. It's also worth noting that the peers entering after time 1000 do not experience the same initial rate decrease and subsequent explosion. Instead, they immediately enter a rapid download rate, and from time 2000 onwards, the peers experience a saturation in download rate. This can be explained by the abundance of seeding peers that chose to stay and contribute to whomever needs a chunk. As for the rapid rate increase of the first peer, this can be explained by the peer initially relying on the source peer for most of its chunks. However, once more peers enter and also accumulate their own chunks, not necessarily the same ones, it can offload

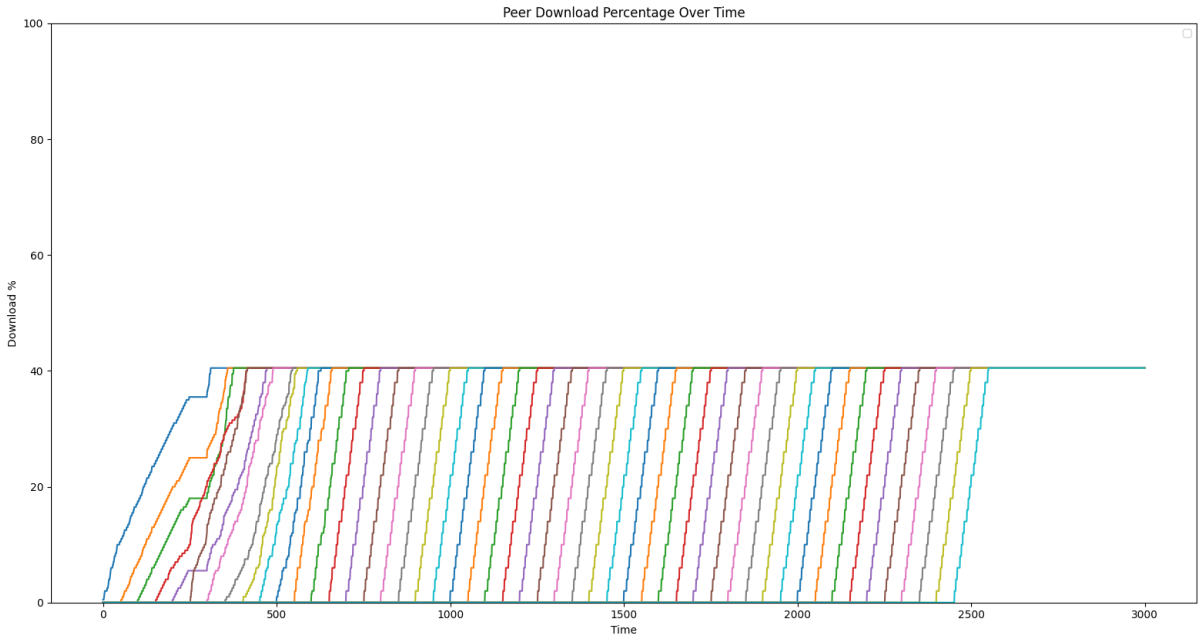
the requests from the source peer to others, and thus is not bottlenecked by the source peer's upload bandwidth.

Torrent with Source Peer and Churning Peers



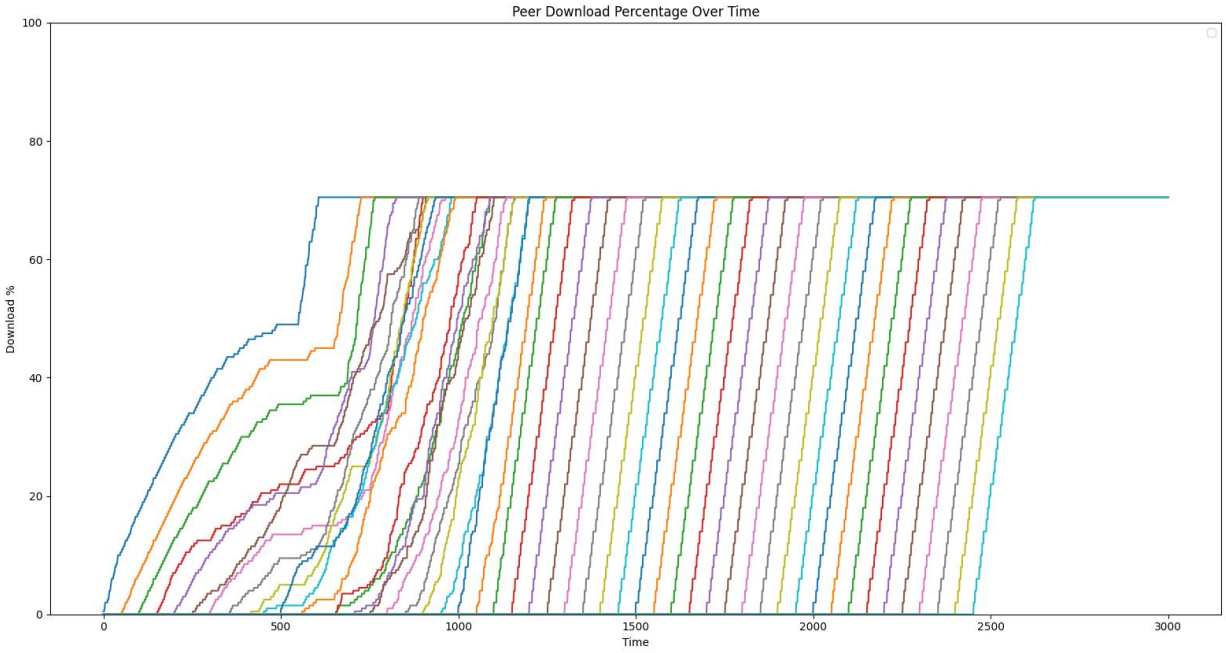
Now that the peers leave once their download is complete, there is not a perpetual abundance of help for the still-downloading peers apparent in the smaller, unsaturated slopes of the subsequent peers after the first. In essence, the remaining peers struggle to get the rarer chunks that are desirable, so it still must request from the source peer and others that have a lower chance of already acquiring them.

Torrent with Source Peer that Eventually Leaves at Time 250 and Altruistic Peers



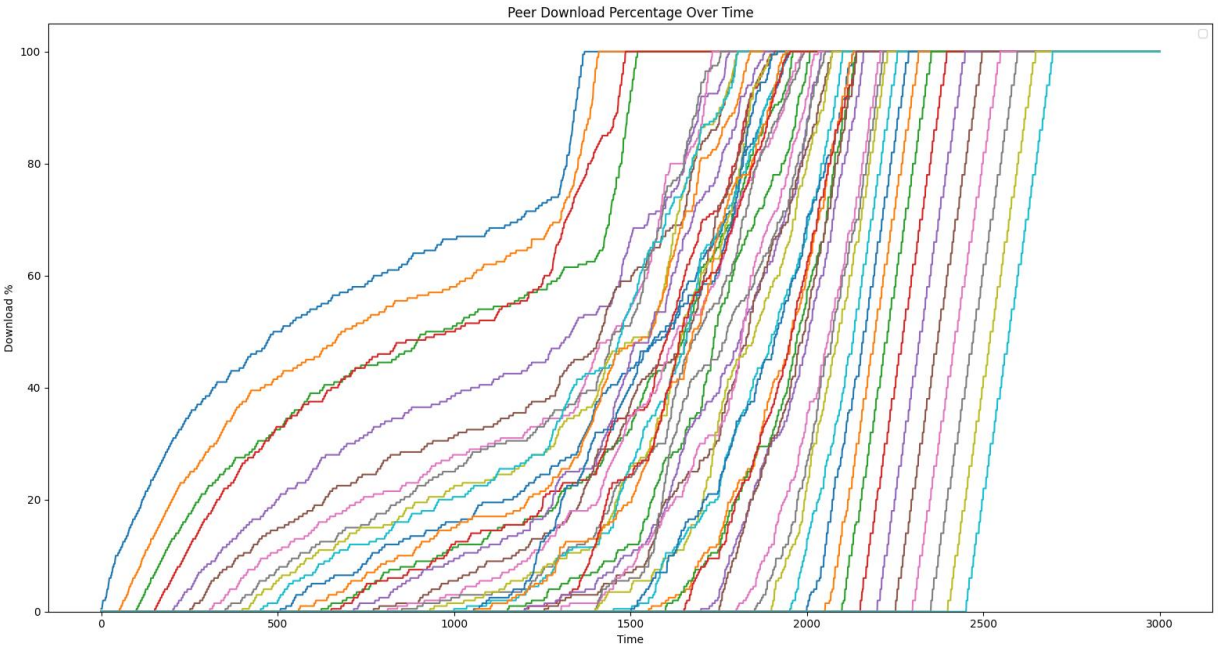
If now we make the source peer leave early into the simulation at time 250, we can see that the entire Torrent will struggle together to ever finish downloading. Effectively, the download rate saturates in this case just above 40% since at the point of the source leaving, the chunks that are in circulation can only accumulate to that amount. In other words, a Torrent is only as complete as what exists. Furthermore, the Torrent experiences a similar download rate saturation as before due to the abundance of seeding peers. The question now is: how early can the source peer leave without compromising the perpetual completeness of the Torrent?

Torrent with Source Peer that Eventually Leaves at Time 500 and Altruistic Peers



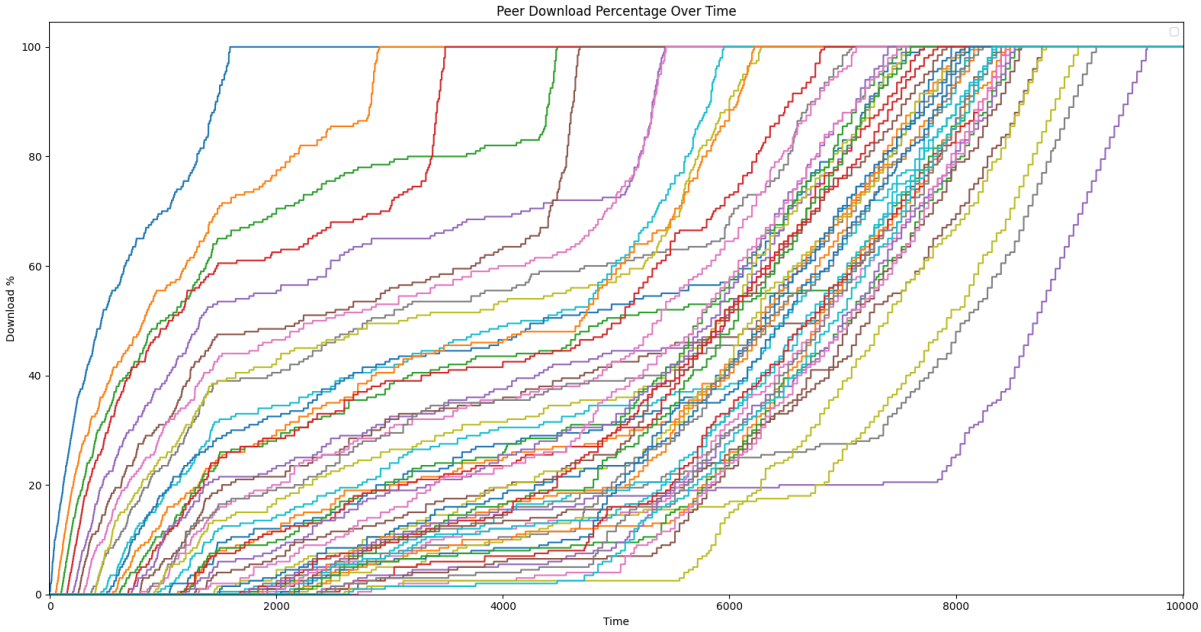
The saturated download percentage increased from around 40% to 80%. It seems the source node is still leaving too early.

Torrent with Source Peer that Eventually Leaves at Time 1500 and Altruistic Peers



If the source peer leaves at 1500, we can observe the Torrent stays alive and subsequent peers can still reach completion due to the altruistic peers helping. In short, if there are at least two peers that hold all the file chunks, a peer can leave without “killing” the Torrent.

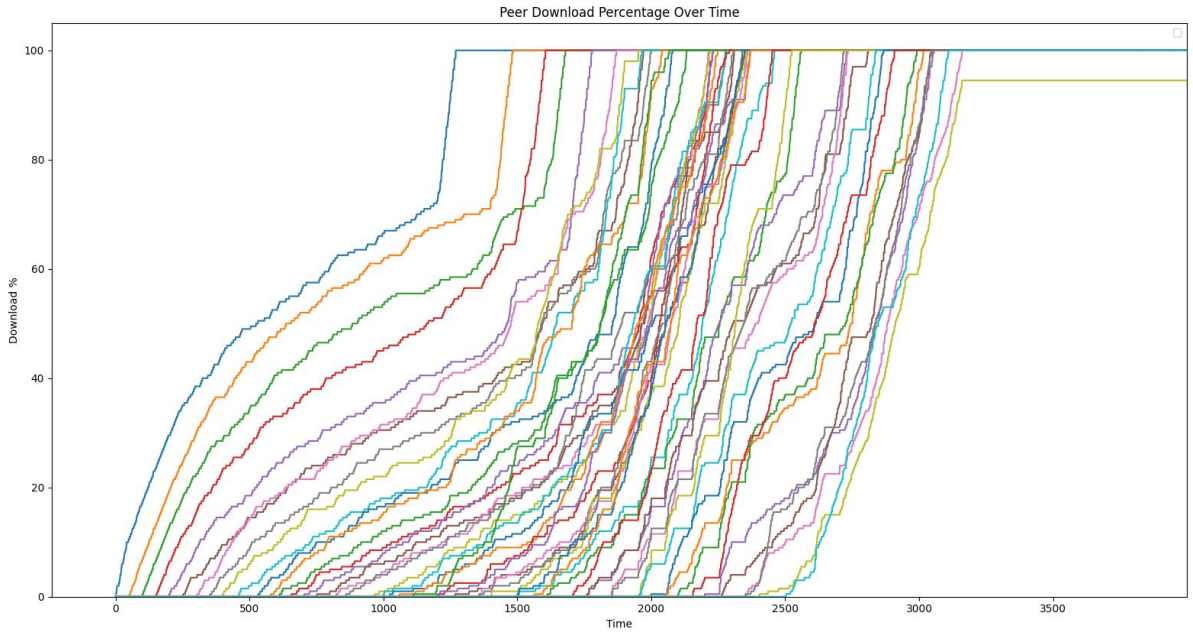
Torrent with Source Peer that Eventually Leaves at Time 1500 and Altruistic, Low-Bandwidth Peers



By decreasing the bandwidth of each peer by a factor of 10, except for the source peer, we can investigate the effect on download rate after the source peer leaves. Namely, the download rate is like the case before pre-source-leaving because all the peers rely on the source for the initial chunks and the per-peer download rate is bottlenecked by the source peer upload rate. However, once the source peer leaves after time 1500, the peers must rely on each other and seeding, complete peers. Because all peers now have a lower bandwidth, the average download rate is also reduced. Furthermore, we still see an explosion in the rate as observed before, except the saturated download is slower due to the generally low upload rates of each peer. In other words, a peer can only download as fast as the average upload rate.

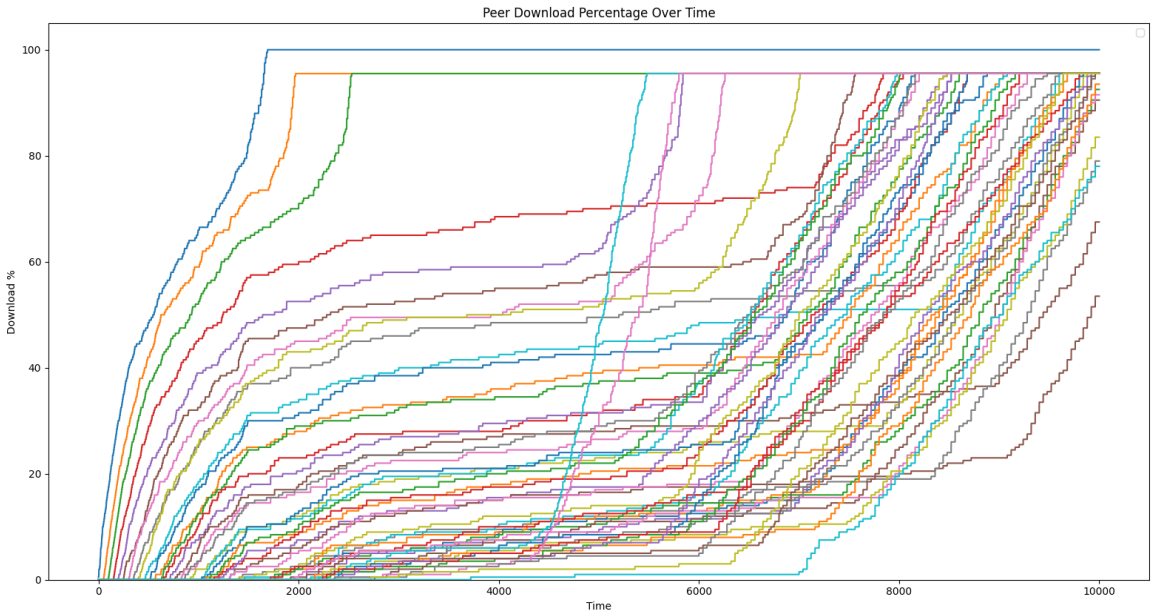
So far, we have seen how a Torrent can stay alive after a source peer leaves, but this was under the assumption that the peers were altruistic. What if they churn instead?

Torrent with Source Peer that Eventually Leaves at Time 1500 and Churning Peers



There are two observations: 1. The per-peer downloading rate does not saturate in a stable manner after the first completed peer churns since there is not an abundance of seeding peers at their disposal; they must source chunks from other peers that are also downloading 2. There is a peer that is left behind and does not successfully complete the download due to the absence of any peers left in the Torrent. To summarize, a Torrent's perpetuation depends on peer altruism.

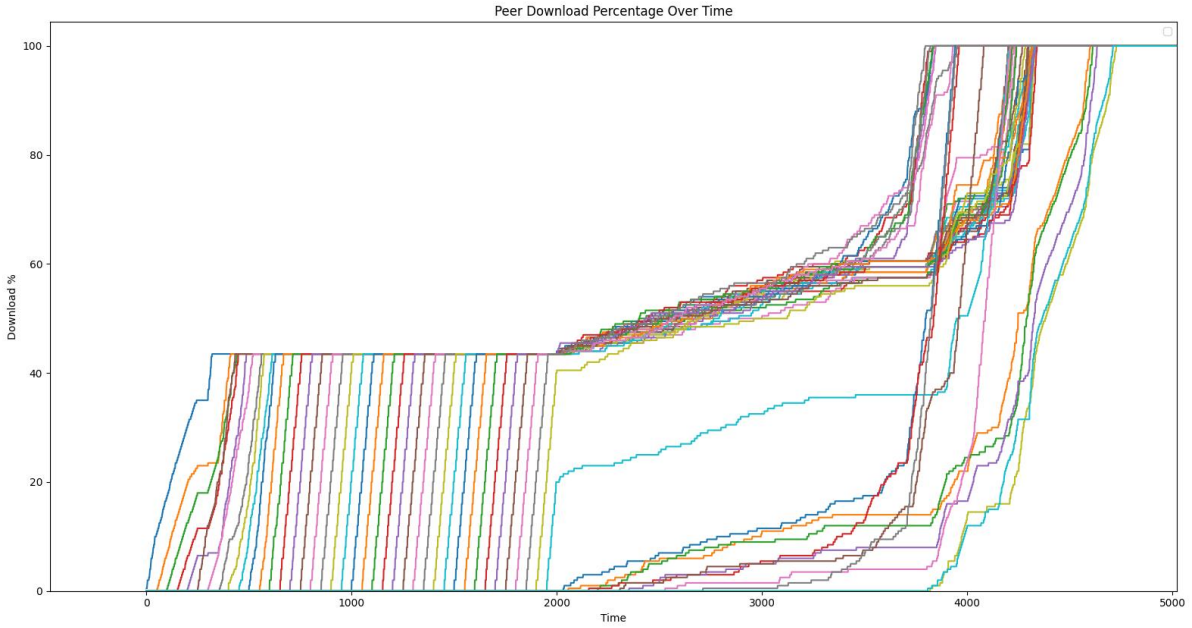
Torrent with Source Peer that Eventually Leaves at Time 1500 and Churning, Low-Bandwidth Peers



If we repeat the same scenario but with low-bandwidth peers, as soon as the source peer leaves at 1500, we only get one peer that manages to complete the download and churn. The rest of the peers are left behind to saturate at about 95%, with the download rate saturating at a lower value. This can be explained by the crucial period before the source peer leaves that requires all other peers to scavenge as many rare chunks as possible to raise the chances of perpetuation. Since the peers have a lower upload bandwidth, they cannot circulate the chunks to others in time subsequently.

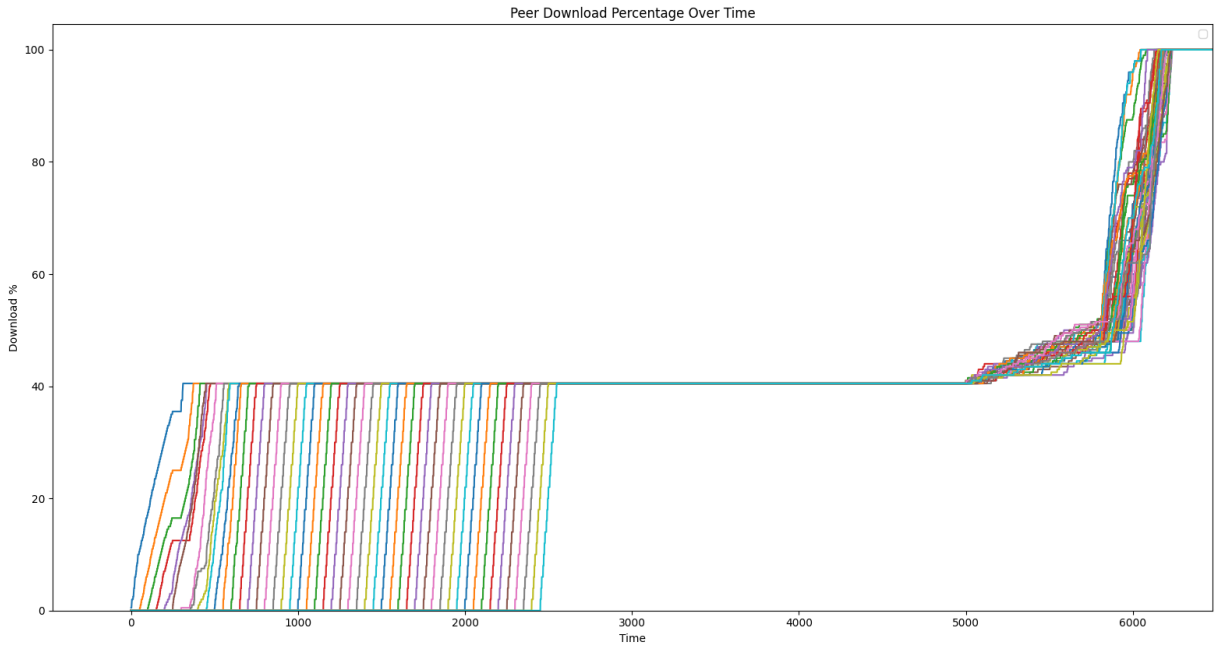
Lastly, we will simulate what happens when the source peer comes back after an early leaving.

Torrent with Source Peer Leaving at time 250 and Return at time 2000, with Churning Peers



Some peers enter the Torrent as the source peer is return so they never experience the saturated download rate.

Torrent with Source Peer Leaving at Time 250 and Return at Time 5000, with Churning Peers



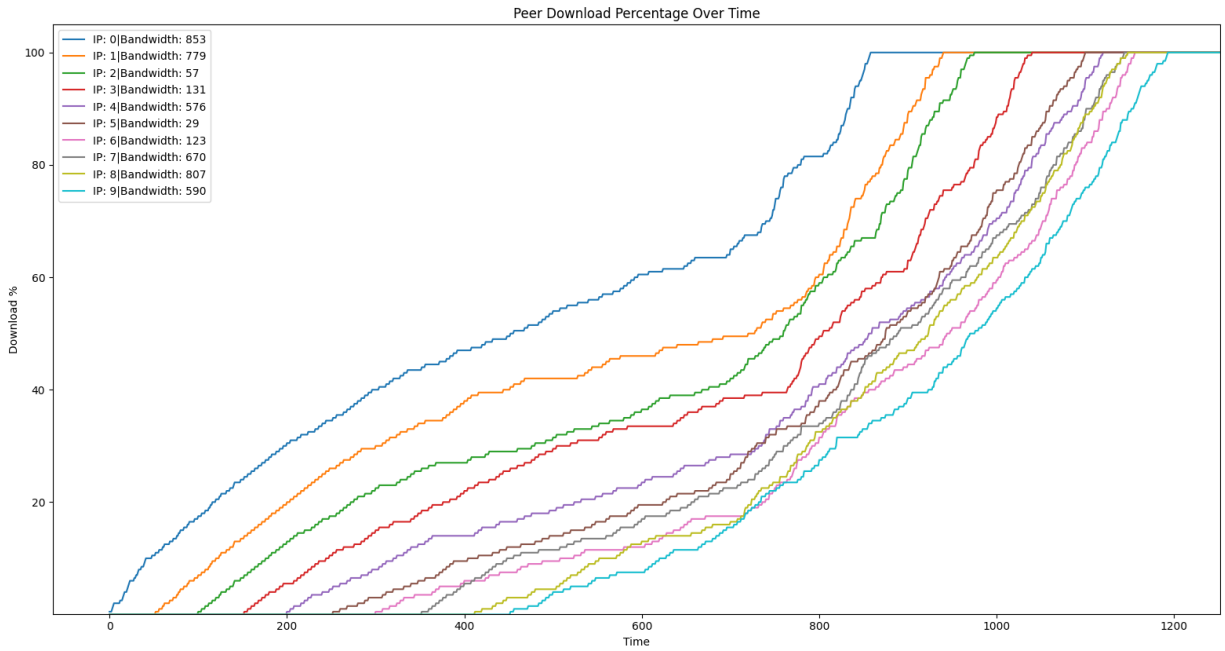
We can see that once the source peer returns, it is business as usual, and the Torrent resumes the chunk distribution behavior all the way to completion.

In summary, a Torrent's resilience to peers leaving is determined by 1. the altruism of the existing peers 2. the completeness of the chunks that are in circulation. We do not expect the download rate per-peer to significantly degrade over a couple of peers churning in a massive torrent but there is a chance that the Torrent never finishes downloading because too many peers leave at once.

Peer Heterogeneity

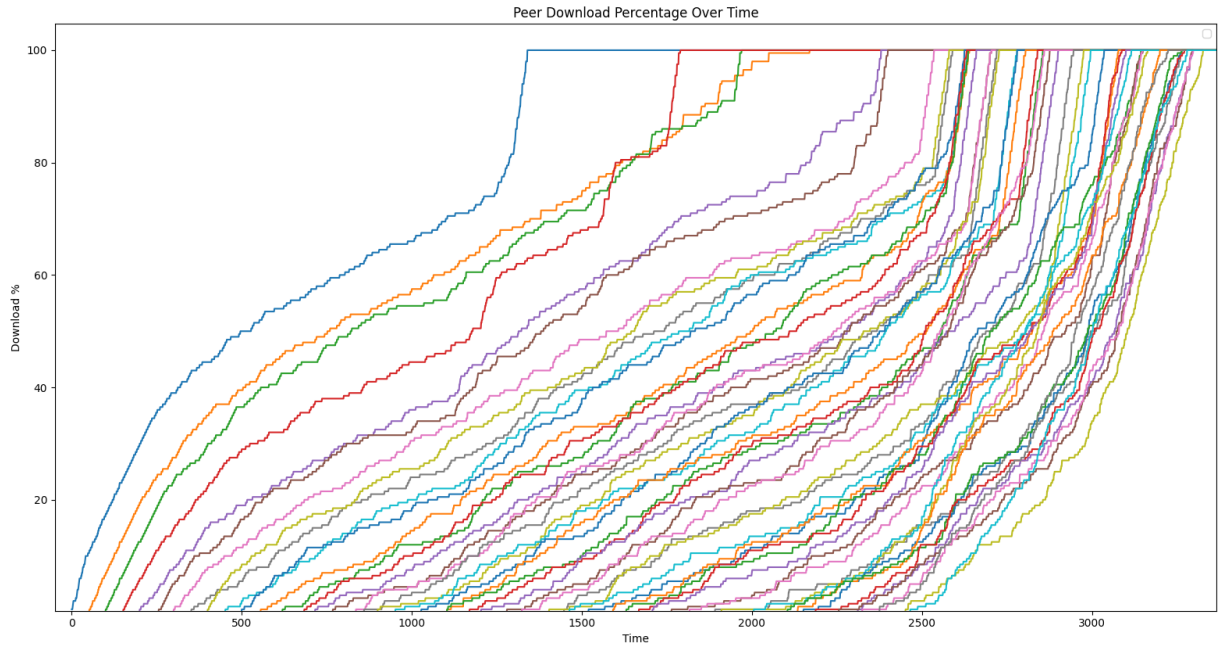
Now we investigate the effects of peers entering with different upload bandwidths. It is a given that the peers will churn.

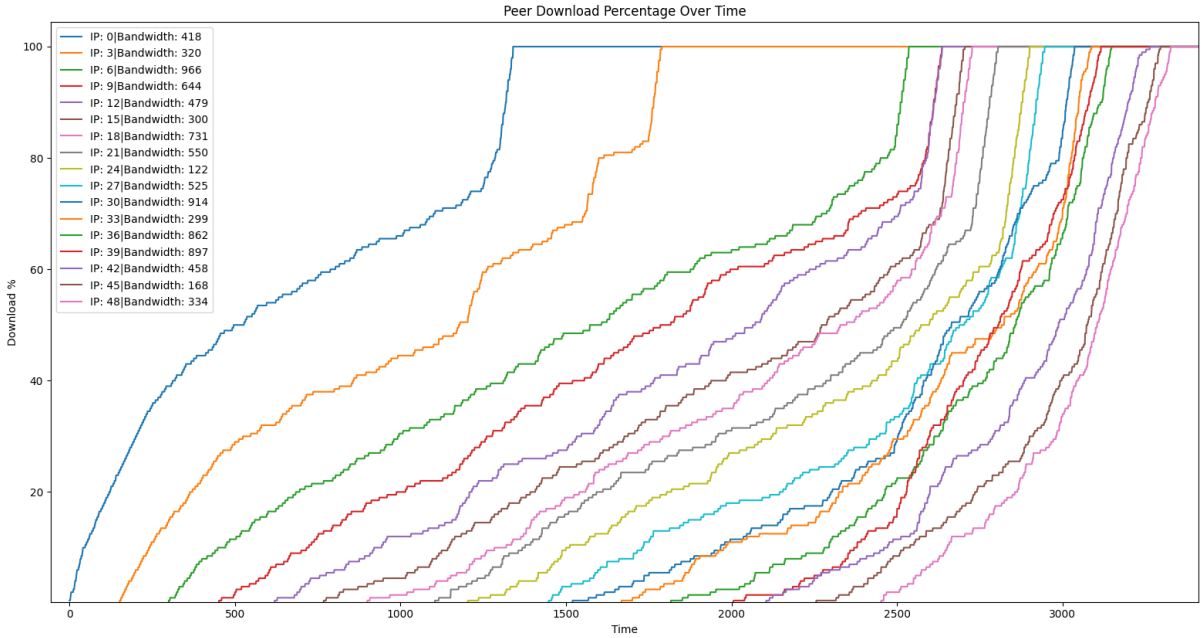
Torrent with 10 Random-Bandwidth Peers



We can witness a similar download curve behavior for the first peer as before: 1. Slowing due to peers entering 2. Explosion after shifting exchange from source peer to others. We can also see that despite the vastly different upload bandwidths, the peers generally share the same download rate trend, exemplifying the Torrent's ability to homogenize an otherwise heterogeneous set of peers. This can be explained by the diversity in the peer exchange network, where their top 4 peers can be sent chunks at different rates but average out to a global rate.

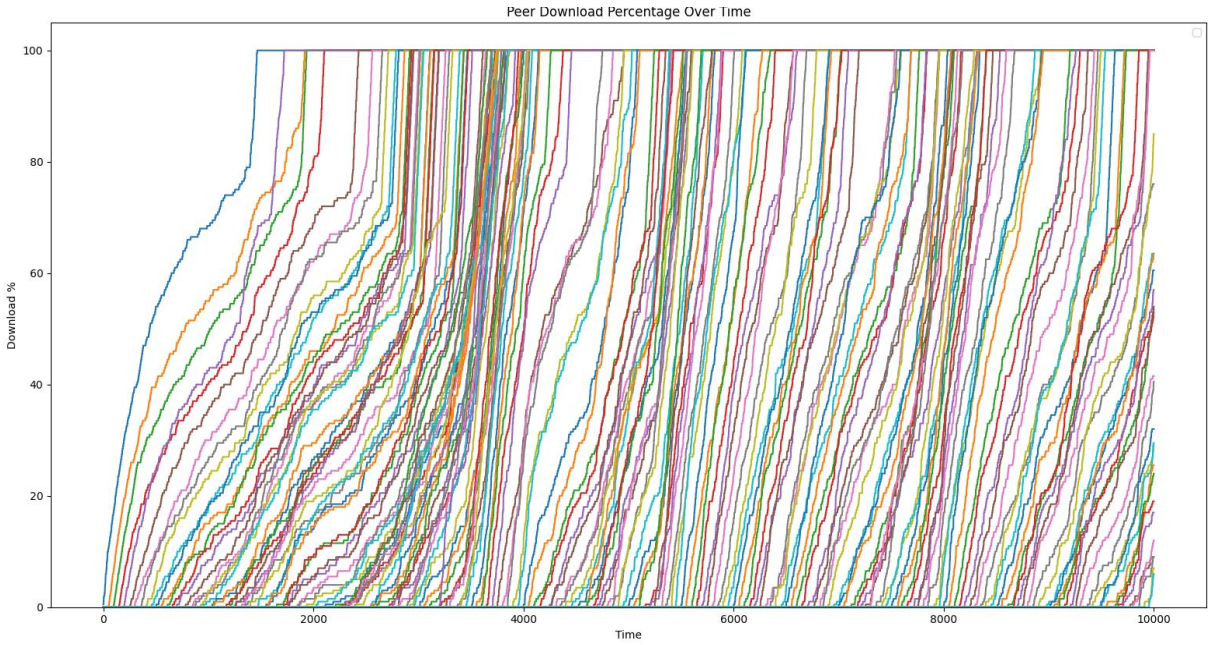
Torrent with 50 Random-Bandwidth Peers

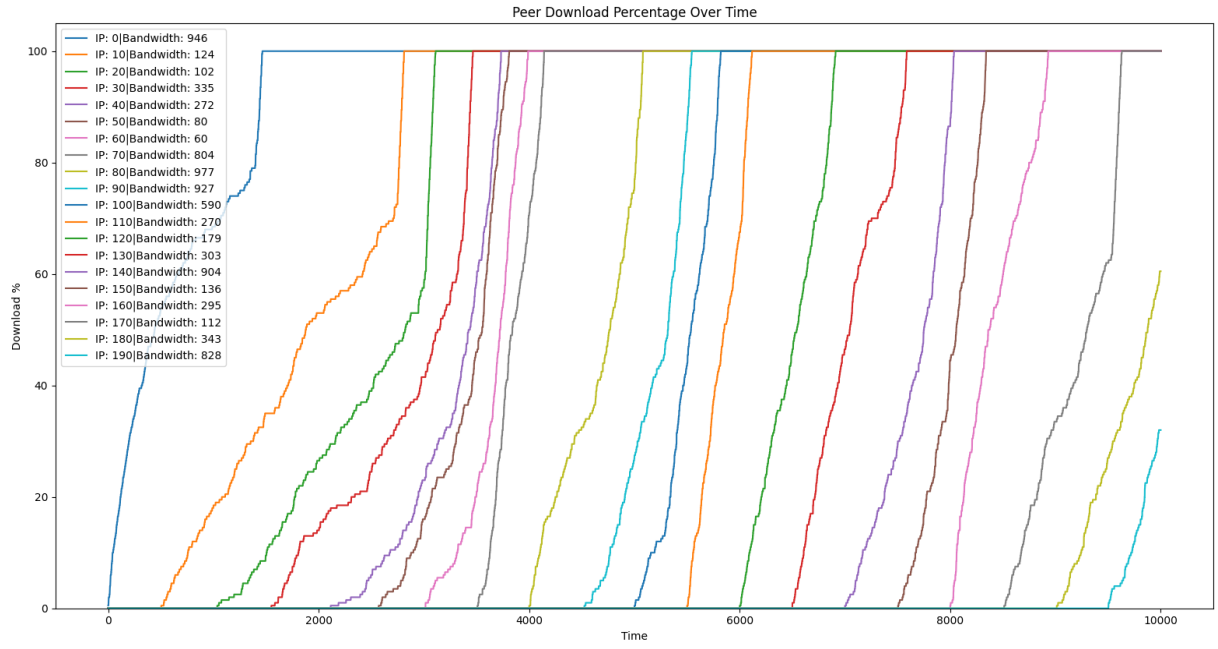




Again, we see similar download trends across all peers despite their different upload bandwidths. Moreover, we also see the effect of scale near the end of the run as more peers exist in the Torrent to help each other out equally.

Torrent with 200 Random-Bandwidth Peers

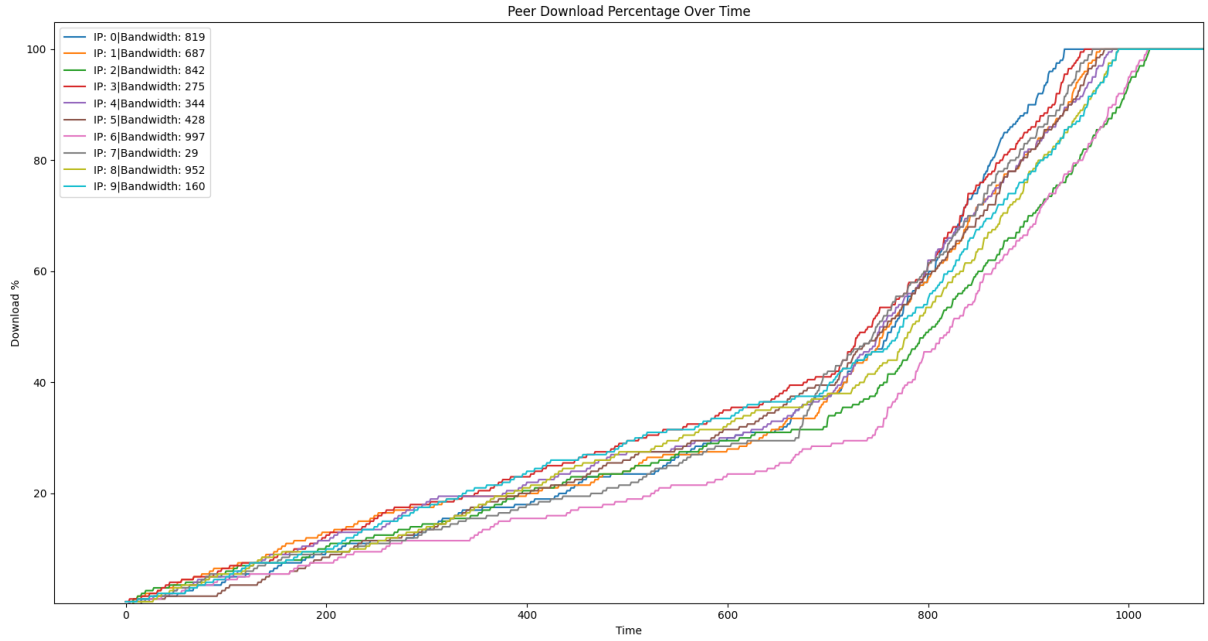


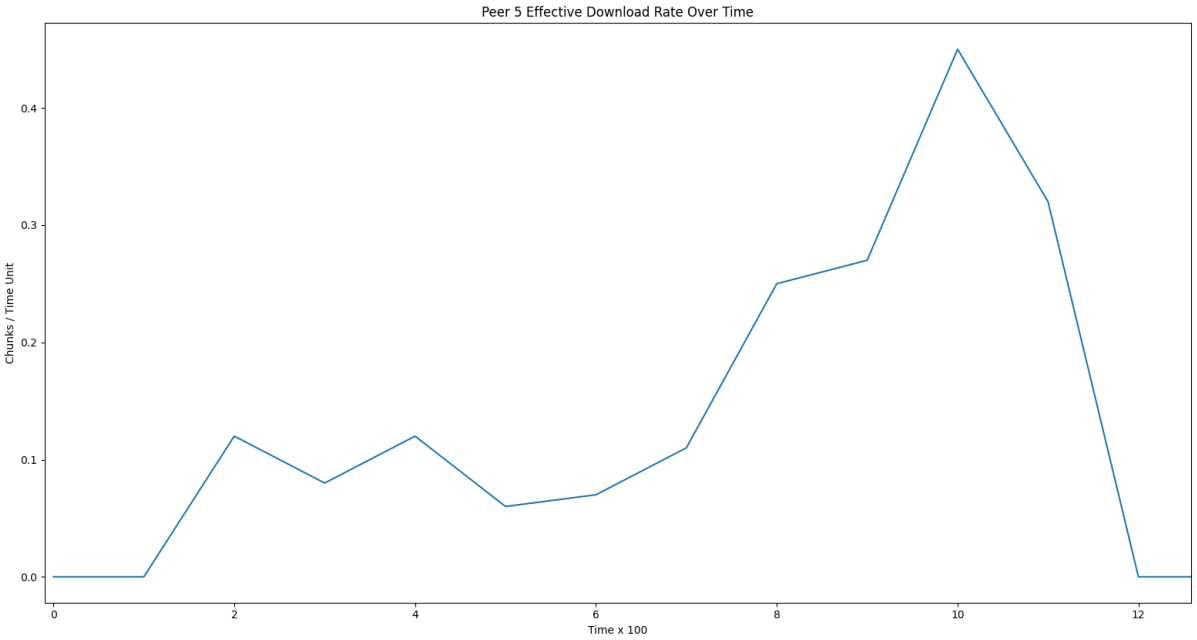


When we increase the peer count to 200, a periodic behavior on the macro scale appears. It is ostensibly a “singularity” that occurs when the current peers exchange with each other to increase the global average download rate towards saturation. Once this group of peers reach completion, they churn and thus leave the next group of peers to create the next “singularity.”

Up until this point, we have staggered the entry of each peer into the Torrent. What if they started at the same time?

Torrent with 10 Random-Bandwidth Peers

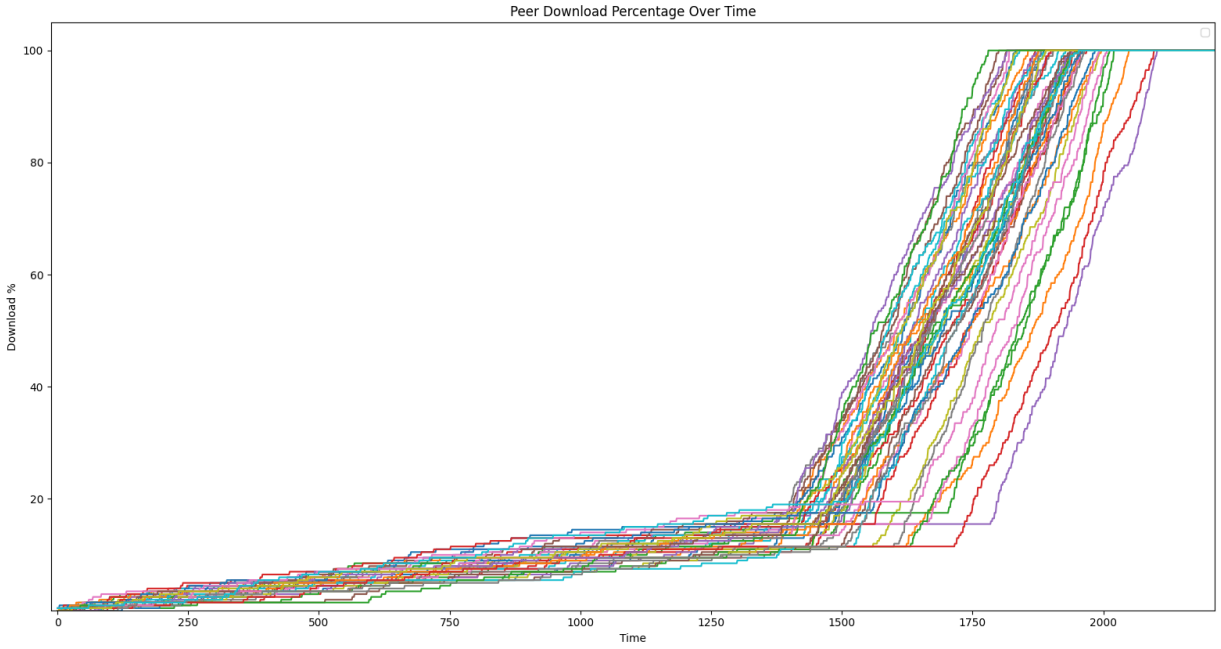


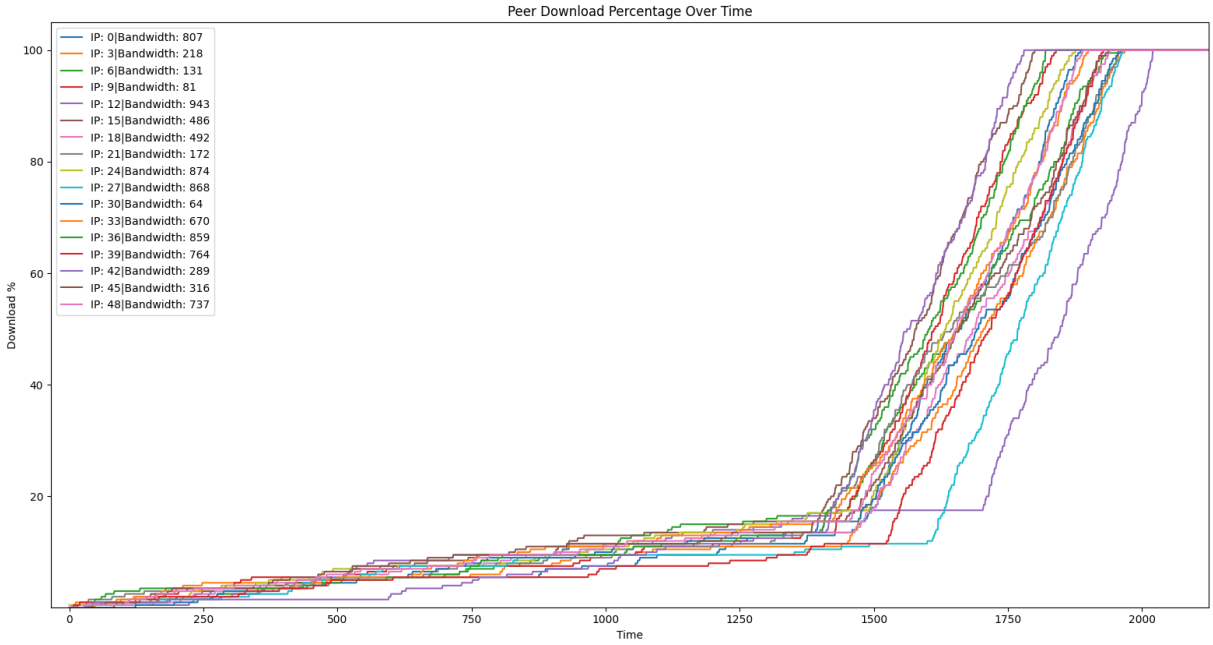


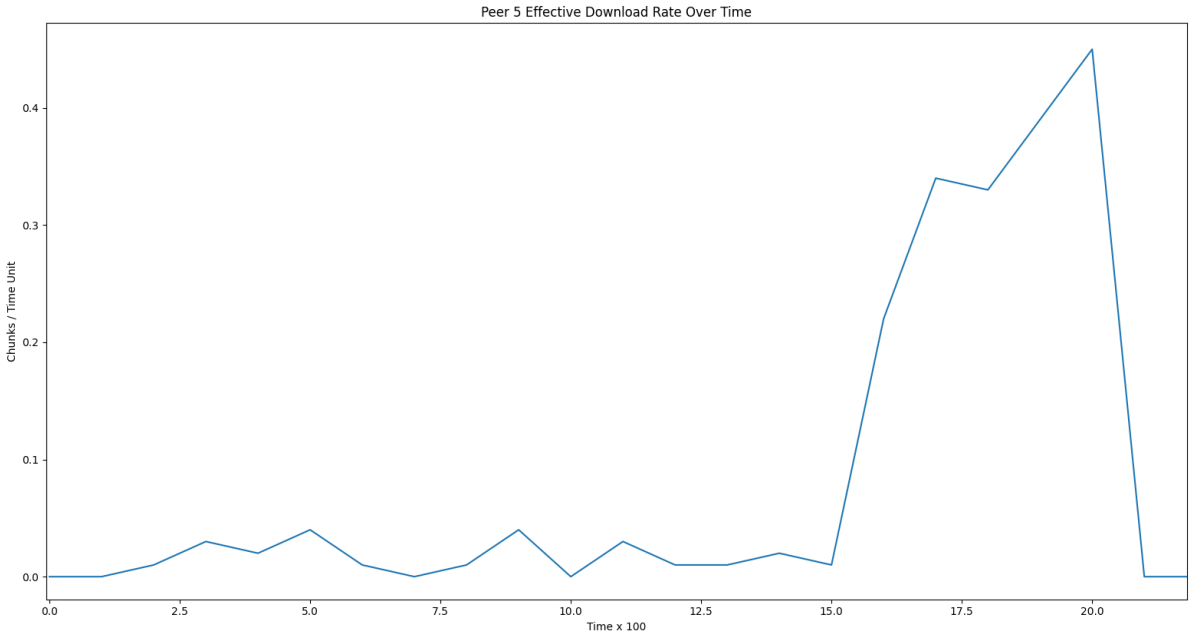
For this scenario, all peers enter the Torrent simultaneously with no chunks, thus there is a race to completion. Additionally, we selected peer 5 to inspect its quantified instantaneous download rate.

Overall, it is a very close race despite the different upload bandwidths. At its peak, peer 5, much like others, downloaded at around 0.4 chunks / time unit.

Torrent with 50 Random-Bandwidth Peers

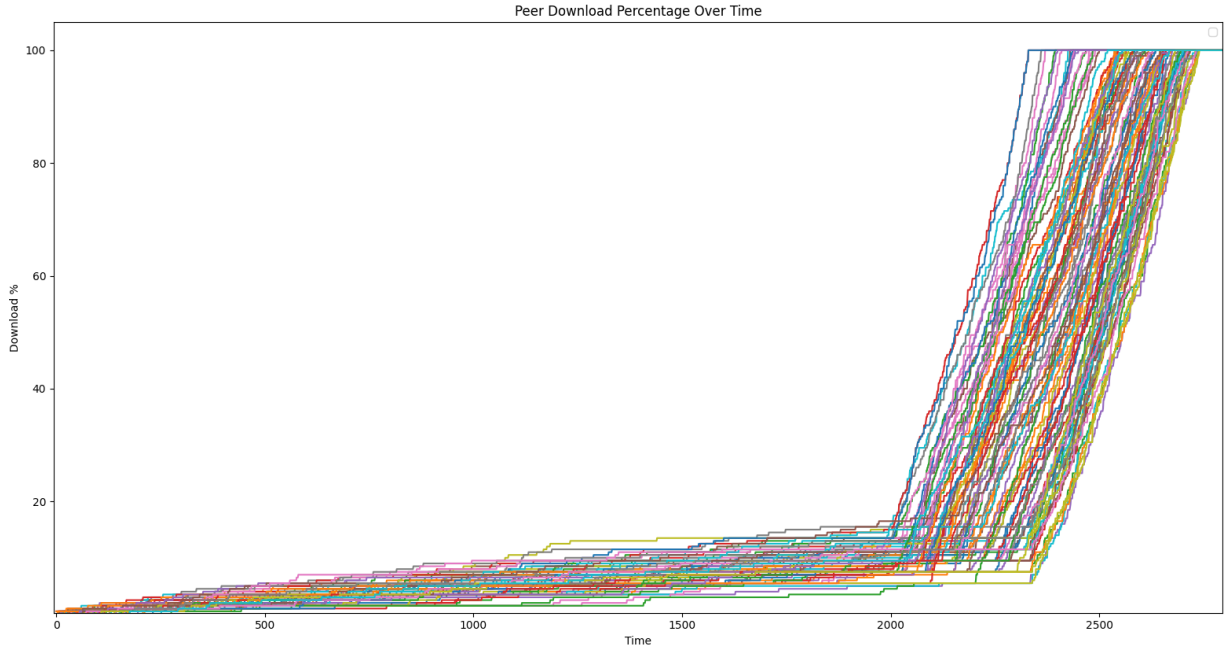


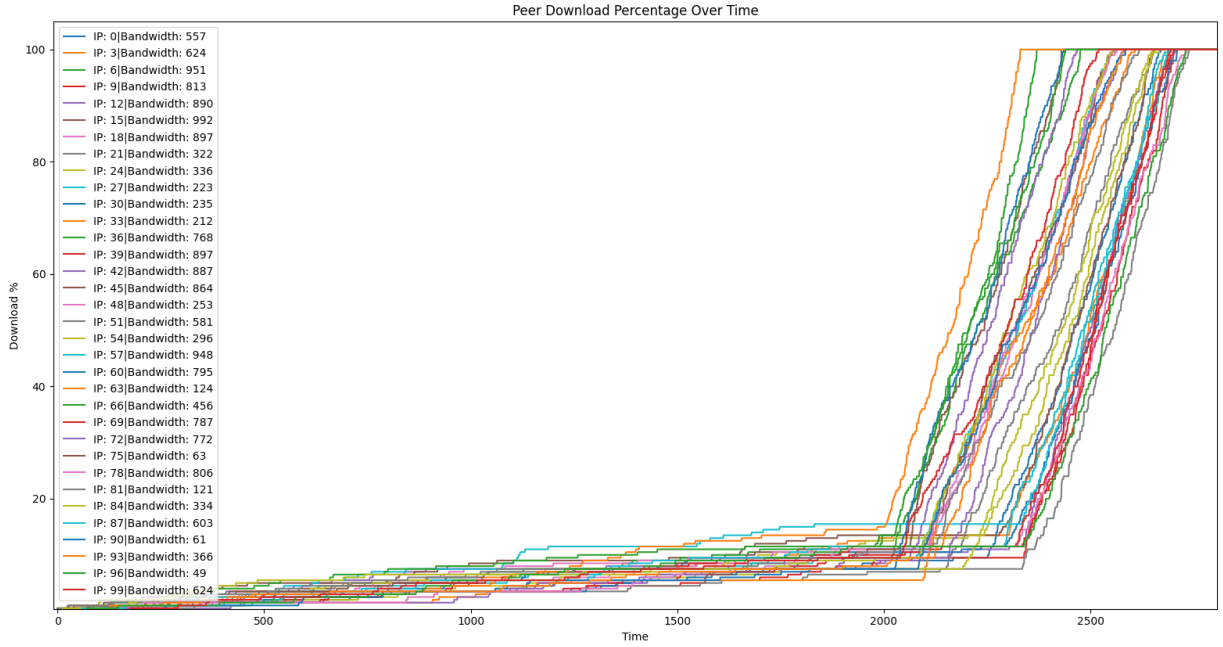


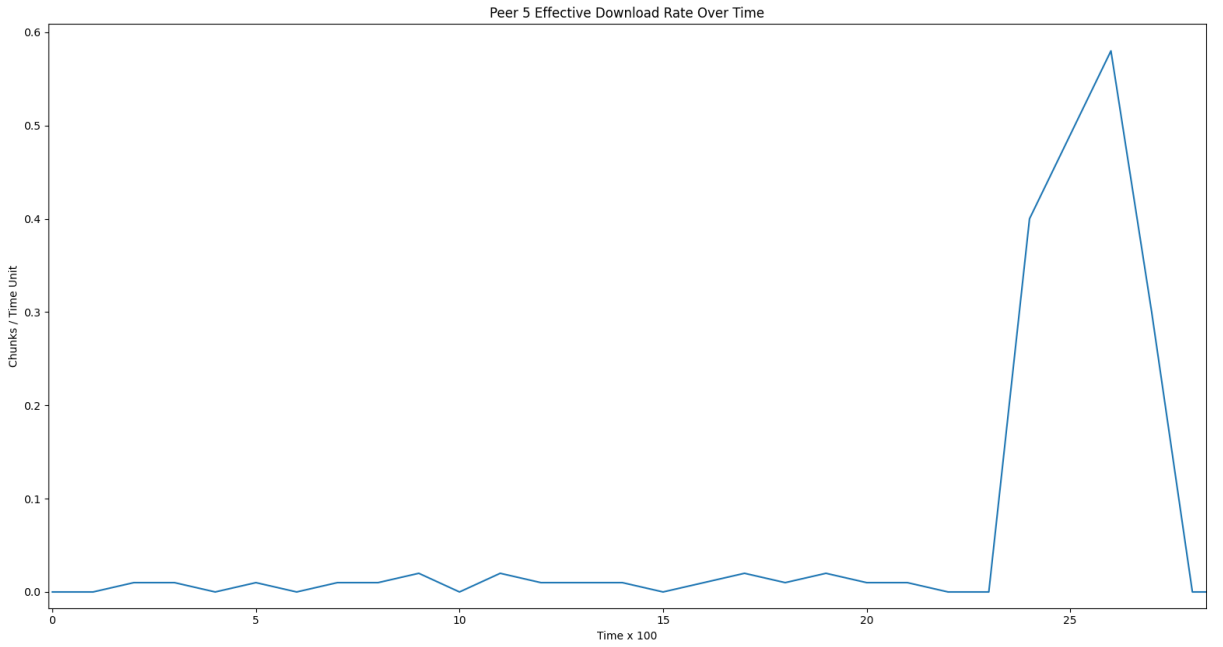


Even though we have increased the peer count to 50, the peak download rate of peer 5 remains relatively unchanged though it has taken longer for all peers to complete the download. This can be explained by scarcity of chunks initially that causes inter-peer congestion on chunk requests. In other words, because everyone is scavenging for the rarest chunk first, not all requests can be met in time.

Torrent with 100 Random-Bandwidth Peers



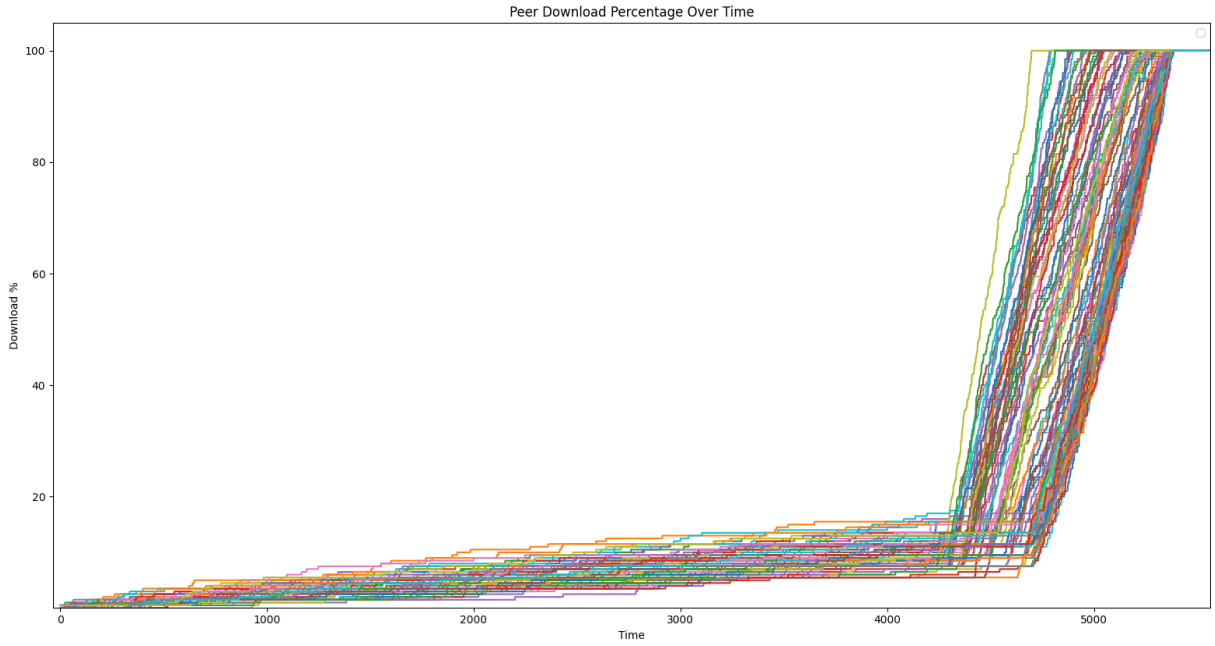


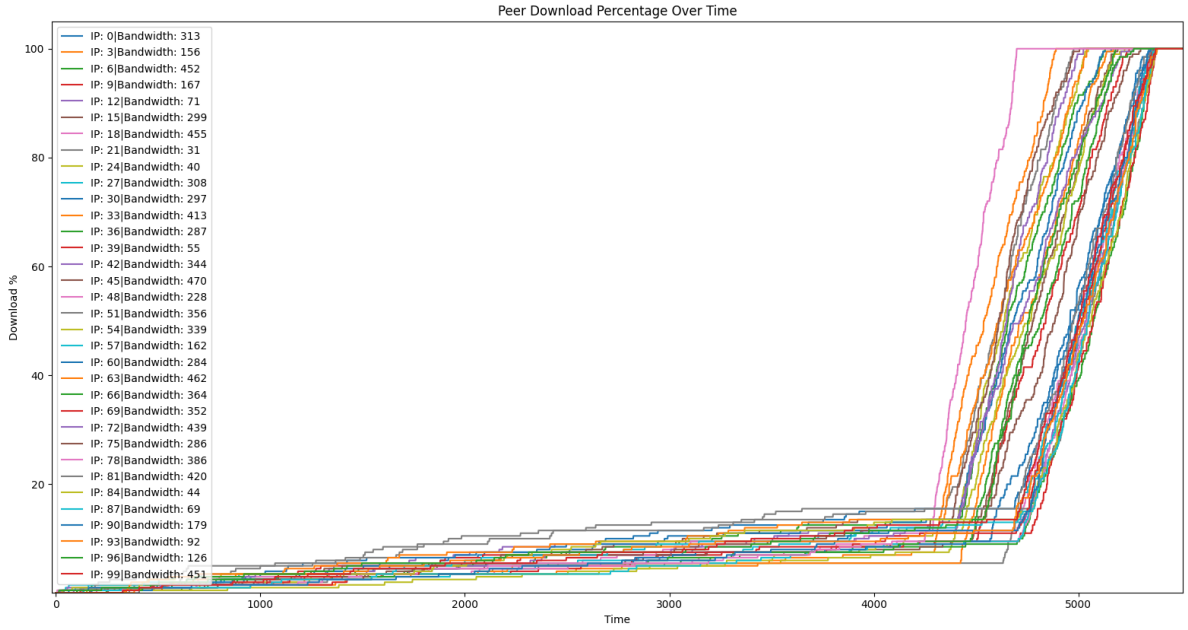


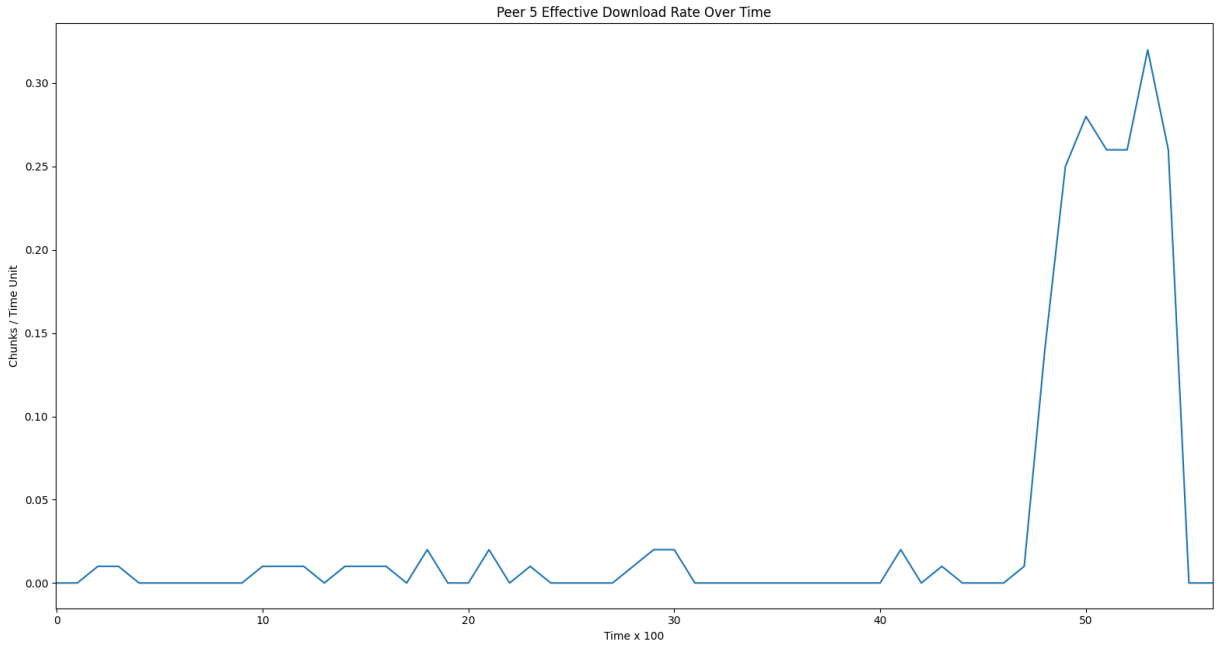
For a peer count of 100, we do see the peak download rate increase somewhat but again it takes longer for the entire Torrent to finish. What if we decreased the max random bandwidth by half?

How about if the highest bandwidth is cut in half?

Torrent with 100 Random-Bandwidth Peers with Halved Max Bandwidth







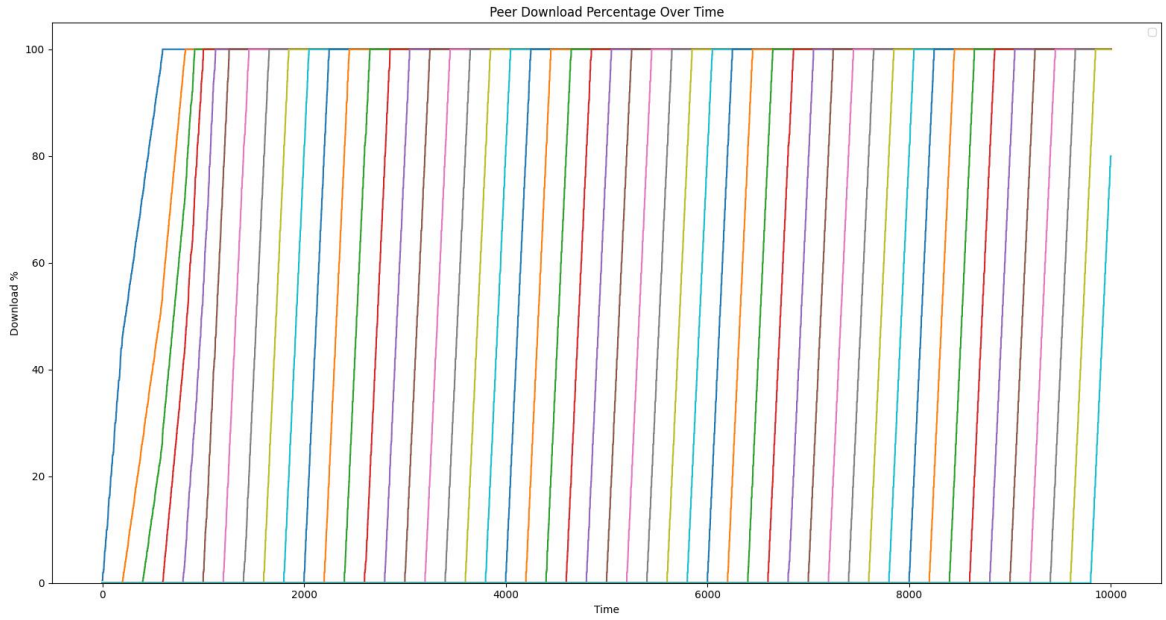
We can observe that by cutting the max random bandwidth by half, the entire network takes twice as long to finish.

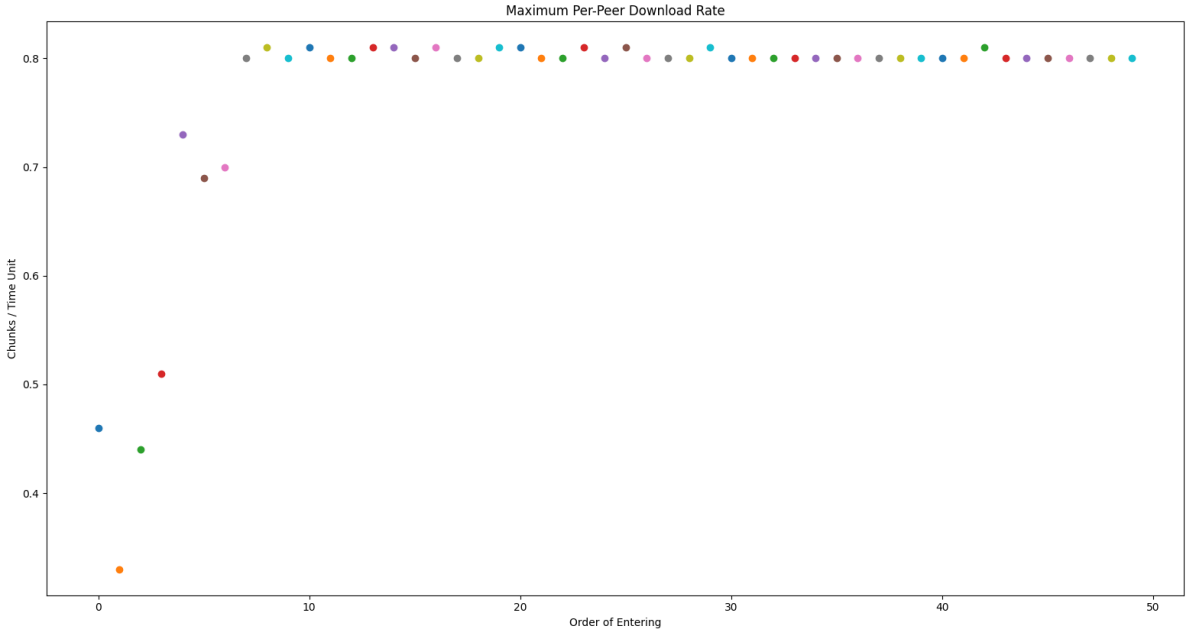
In summary, a Torrent can average out the upload bandwidths of all its peers to yield an effective global download rate that is stable regardless of per-peer characteristics. This behavior is caused by the peer's ability to always keep a list of top 4 peers to exchange with, of whom may be diverse in upload bandwidths and acquired chunks – a peer always has a portfolio of chunk sources that is constantly changing and thus has many paths towards completion.

Scalability

Now we will investigate what happens to the per-peer download rate as more peers enter the torrent and stay to seed.

Torrent with 50 Random-Bandwidth Peers Entering Sequentially at Interval of 200

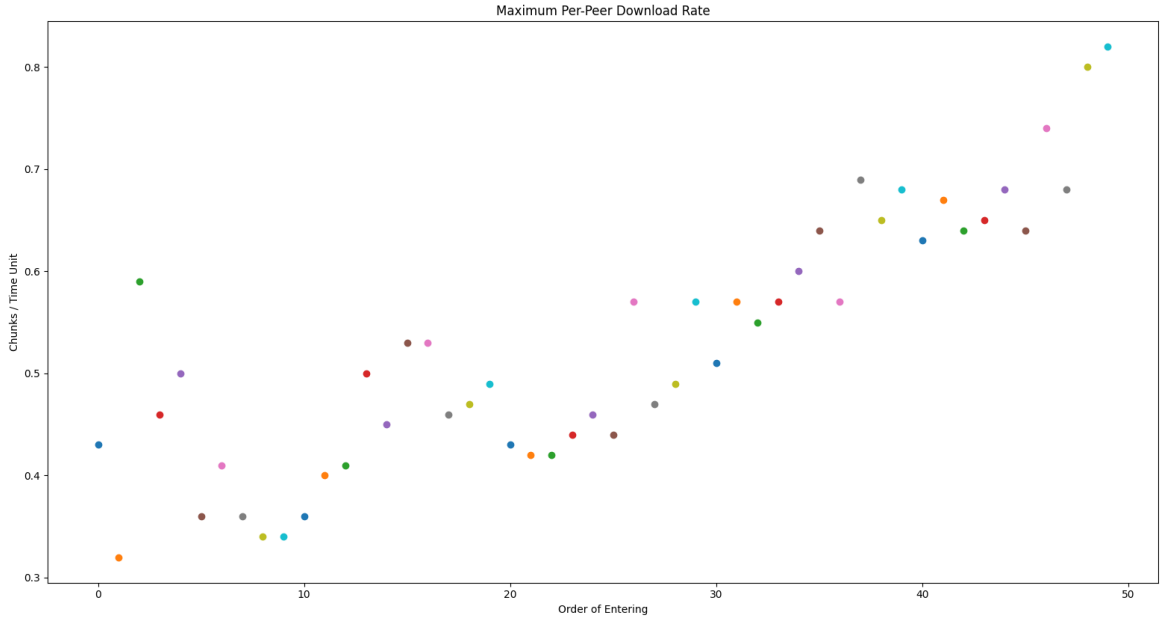


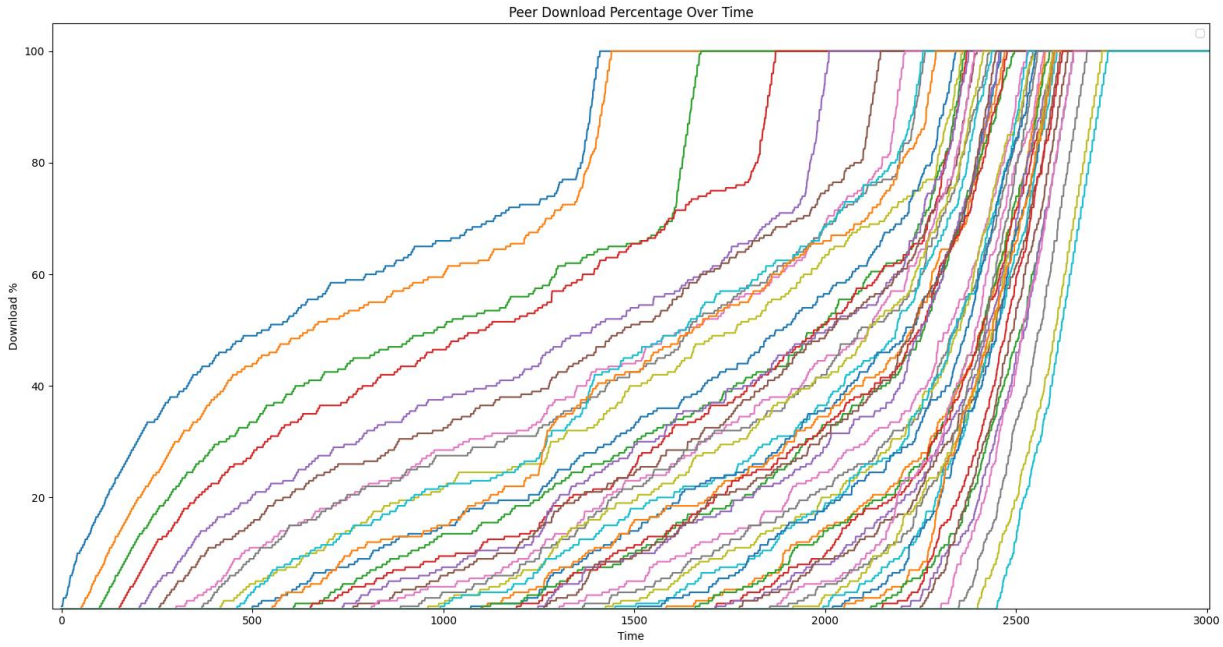


We simulate a Torrent with 200 random bandwidth peers that sequentially enter the tracker at an interval of 200. We assume they do not churn and do not have any chunks to start. We then plotted for each peer their maximum download rates.

From the plot, the max download rate increases as more peers enter. When there are more than roughly 10 peers, the max download rate saturates for all subsequently entering peers due to the altruistic peers that remain to seed. This is simply due to the parameters set for the BitTorrent protocol, namely the rate at which requests can be sent out and the number of rare chunks that can be requested at a time. There must be a finite value for both parameters to prevent over-greediness by one sole peer over the others, congesting the network. The download rate will unlikely saturate as the network will not be congested despite a peer requesting as many chunks as it is missing in one shot.

Torrent with 50 Random-Bandwidth Peers Entering Sequentially at Interval of 50

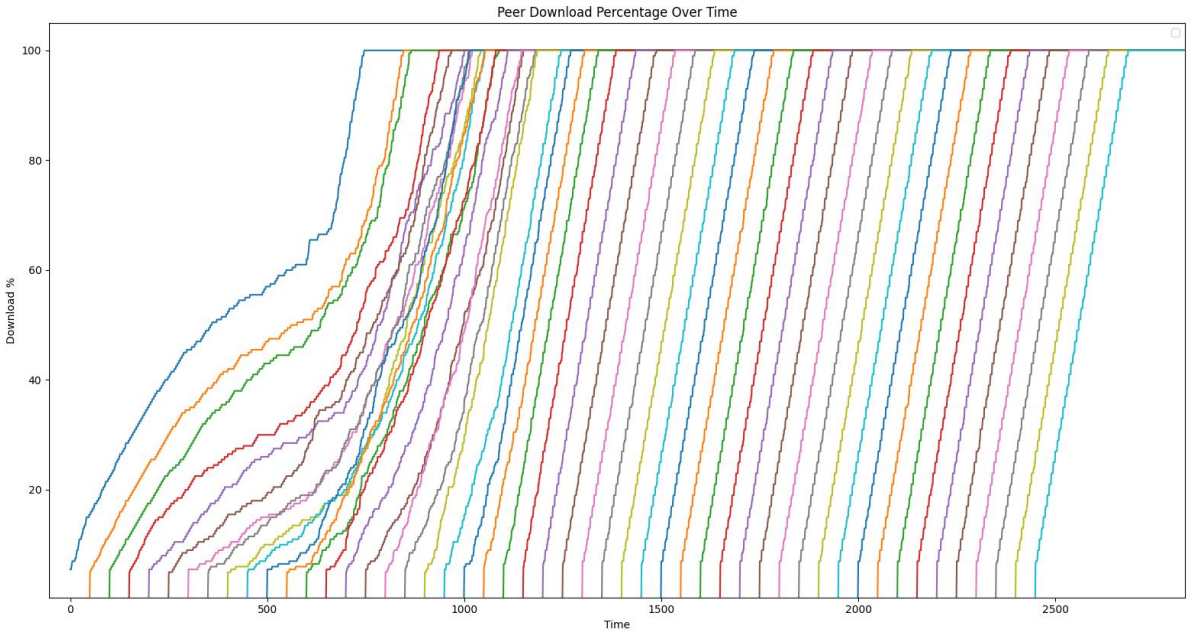


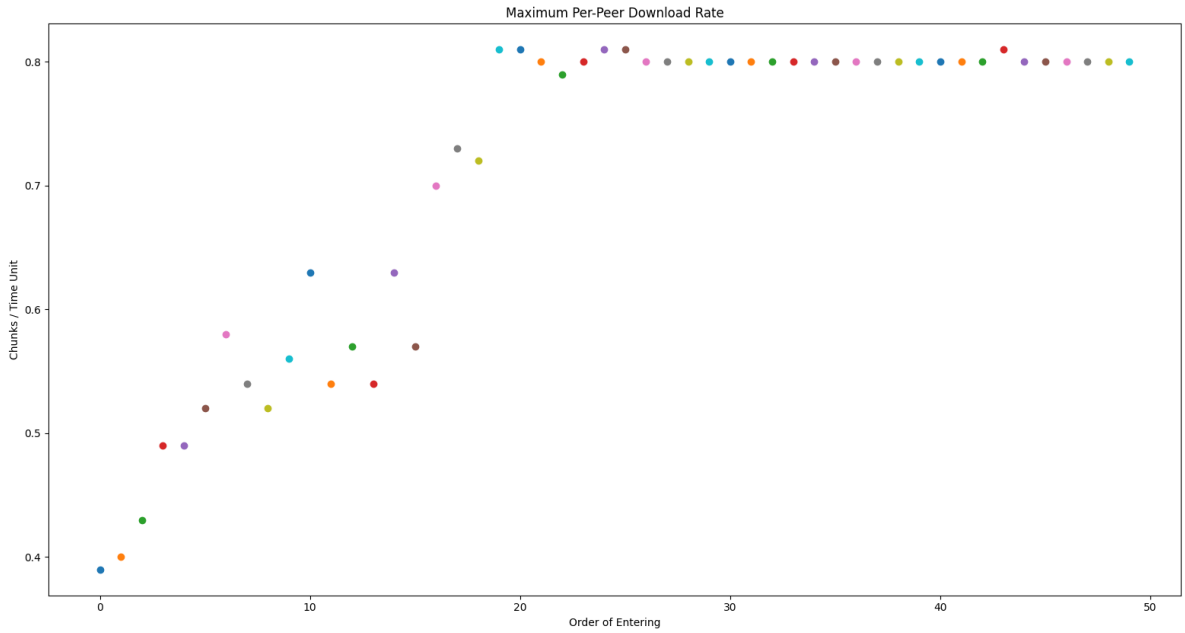


If we decreased the entering interval to 50, we could see the network increasingly struggle to fulfill the requests of all peers. However, after the rapid increase caused by a shift in chunk exchanges from the source to other peers, we can see the true download rate from peer-to-peer exchanges. As more peers enter the Torrent, the max download rate increases towards the saturation value but does not quite reach it due to the overwhelming number of requests experienced by the Torrent.

The initial slow decrease of the download rate is caused by peers entering with no chunks, thereby relying on the source peer. What if they came in with something in their hand?

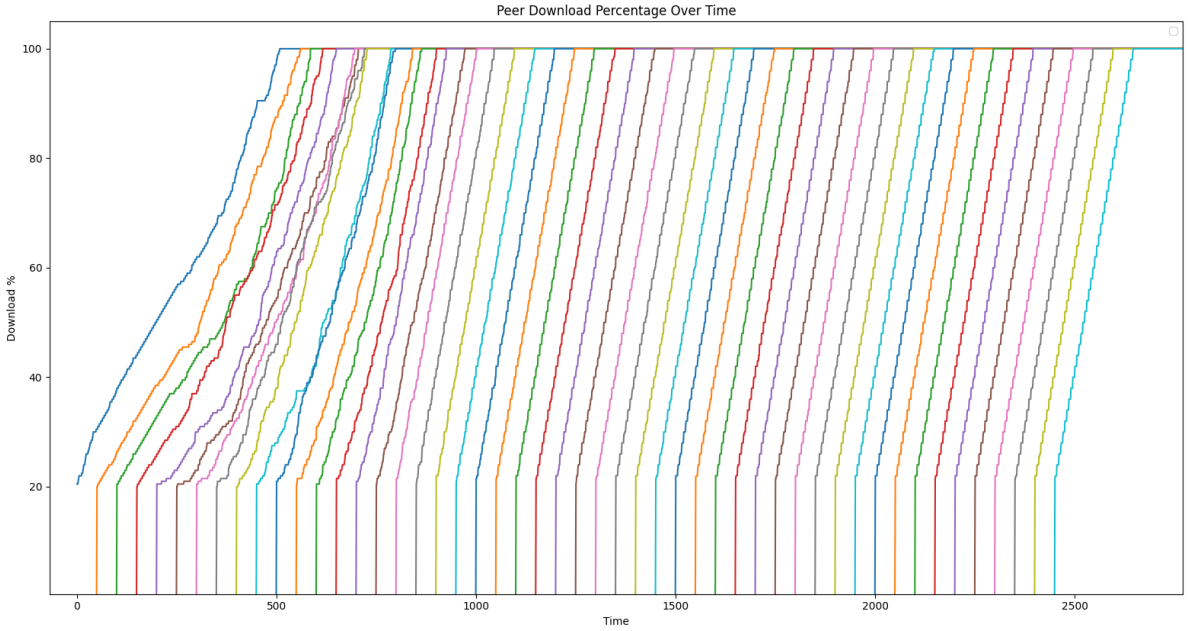
Torrent with 50 Random-Bandwidth Peers Entering Sequentially with 5% at Interval of 50

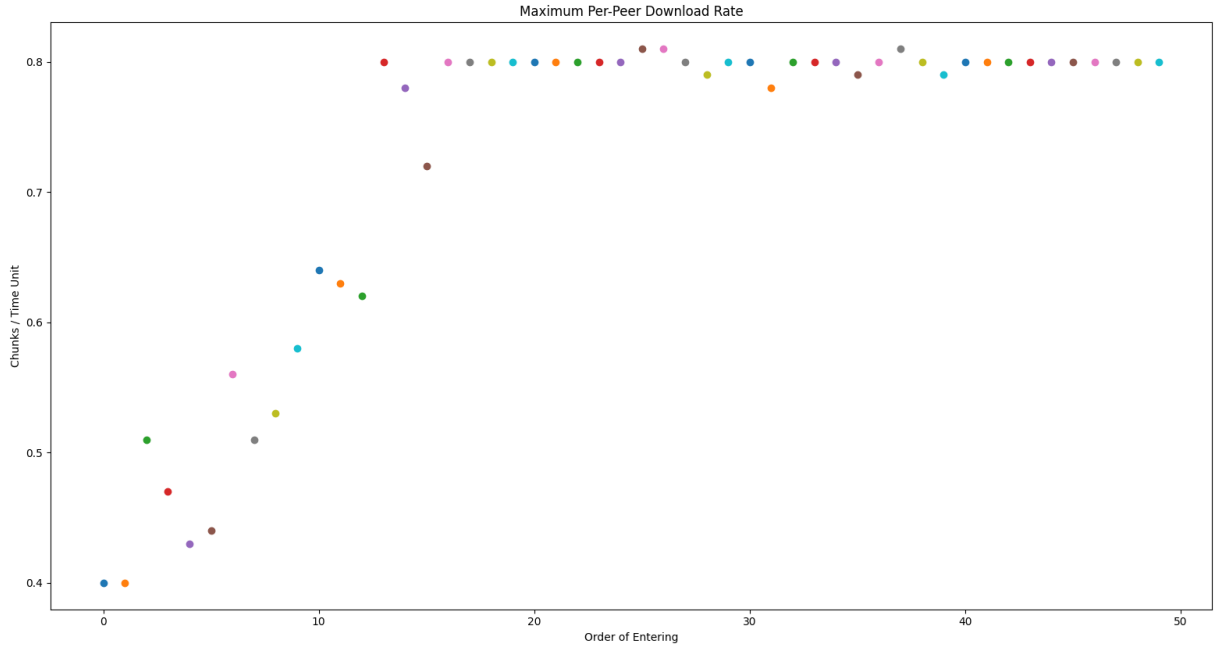




By entering with already 5% of the chunks, the Torrent goes back to saturating after 20 peers – the peers no longer depend on a source peer after a shorter time.

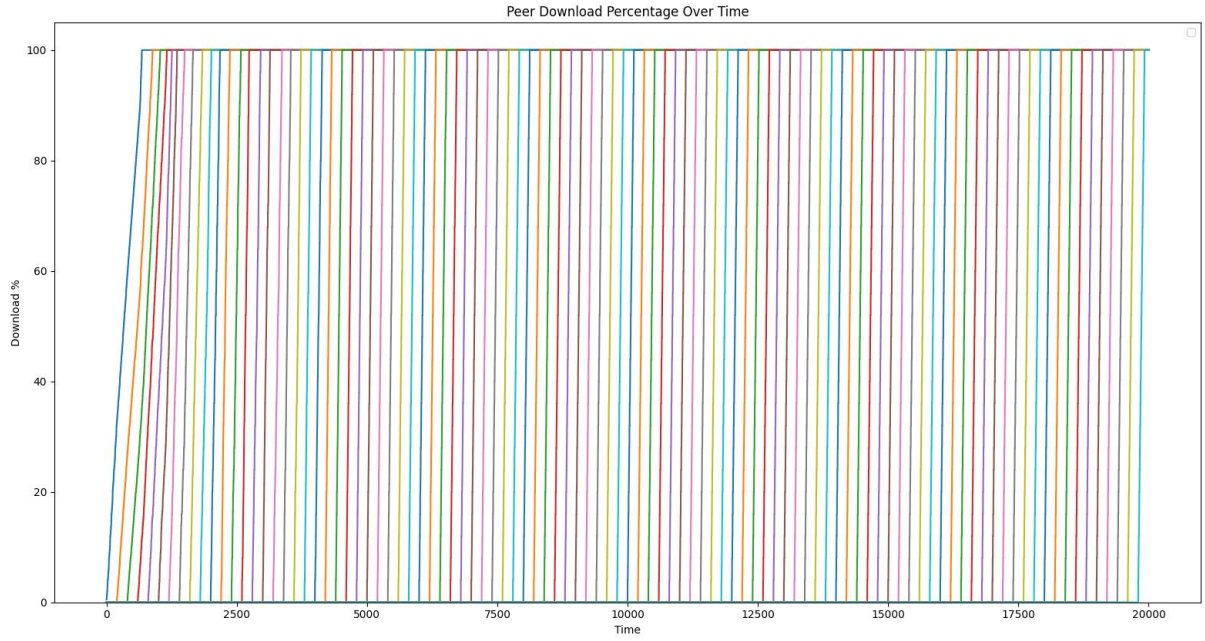
Torrent with 50 Random-Bandwidth Peers Entering Sequentially with 20% at Interval of 50

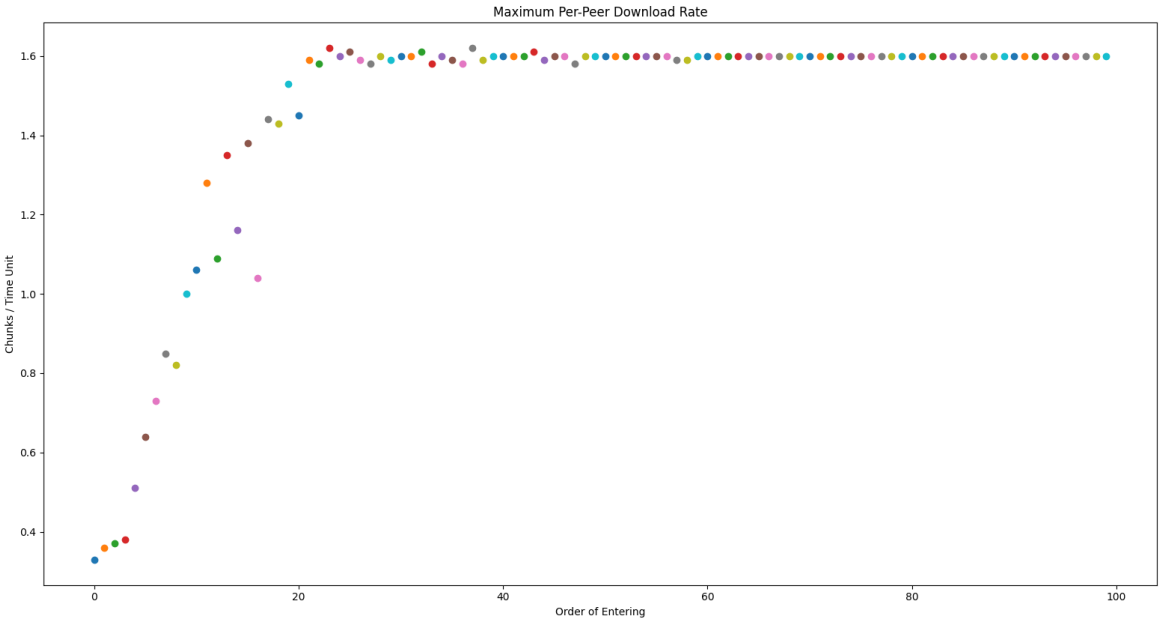




If the peers started out with 20%, the Torrent saturates even quicker. It was previously mentioned that the saturated download rate is a function of how much and often a peer can request chunks – we will investigate this below.

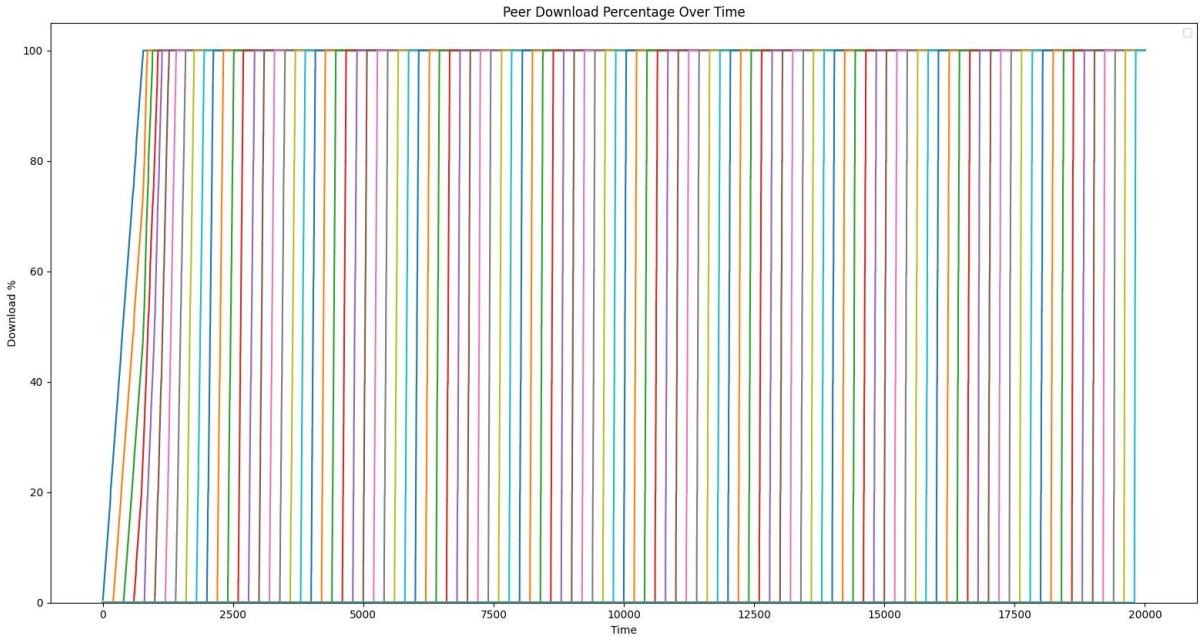
Torrent with 100 Random-Bandwidth Greedy Peers Entering Sequentially at Interval of 50

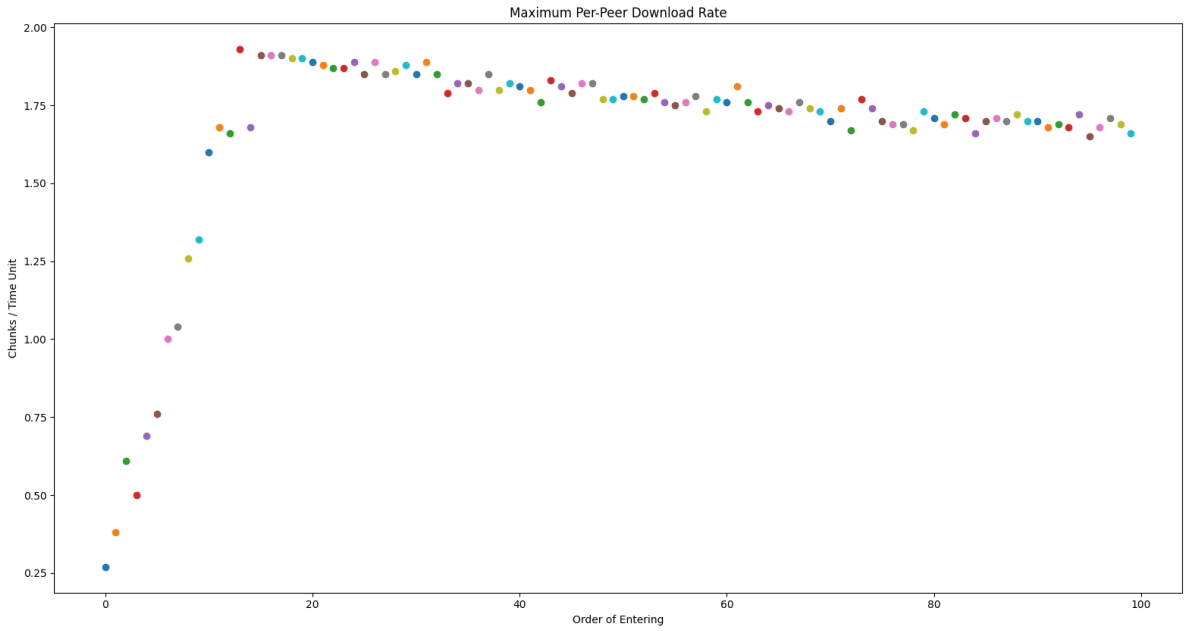




The BitTorrent protocol has been modified such that a peer can request up to 10 rarest chunks at a time instead of 5, a two-fold increase. As a result of increased greediness per peer, the saturation download rate has also increased two-fold, illustrating that the BitTorrent parameters itself can be a bottleneck to homogenize the behavior of the whole Torrent.

Torrent with 100 Random-Bandwidth Greedier Peers Entering Sequentially at Interval of 50

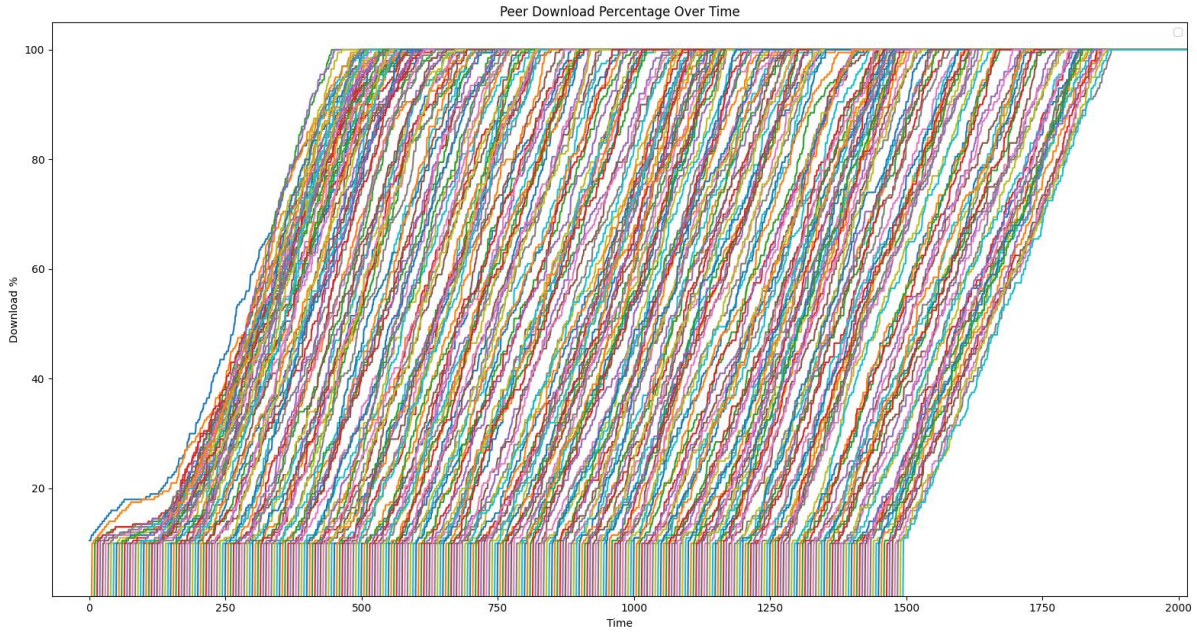


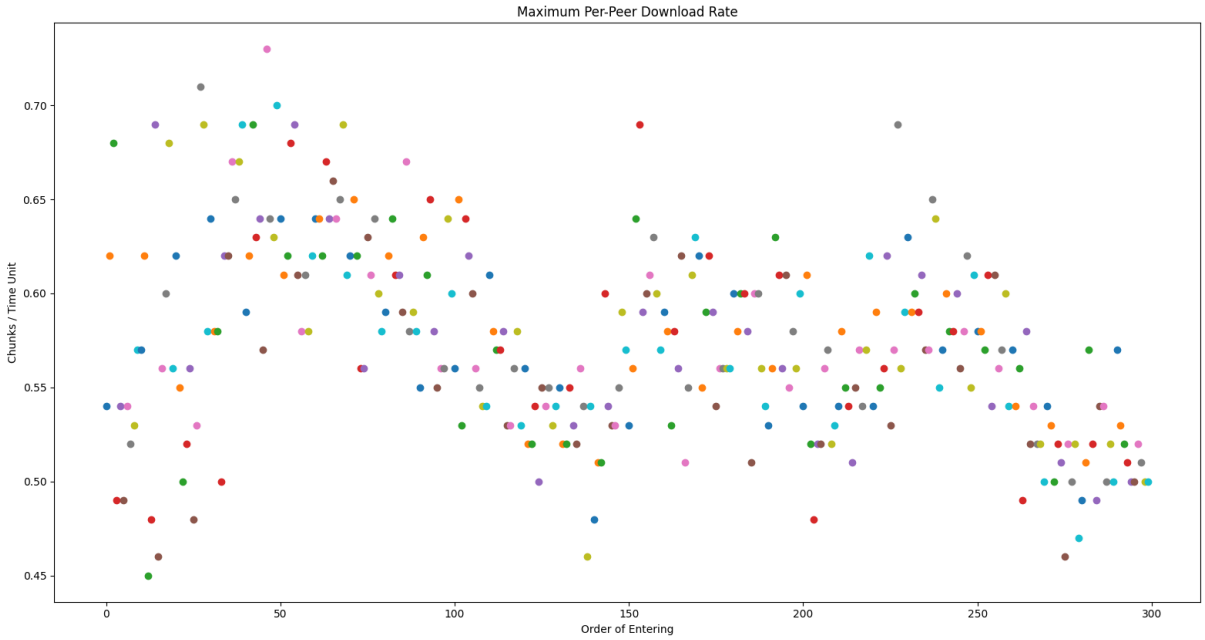


By increasing the rarest chunk request amount by ten, we have made the peers extremely greedy. However, though we see the usual increase in download rate for the first 10 entering peers, the download rate saturates at a value not proportional to the greediness and instead gradually decreases. This can be explained by the finite buffers of each peer such that an overwhelming amount of chunk requests can result in underserving. Effectively, the simulator's modified BitTorrent protocol with a limited request buffer penalizes the entire Torrent for being greedy and

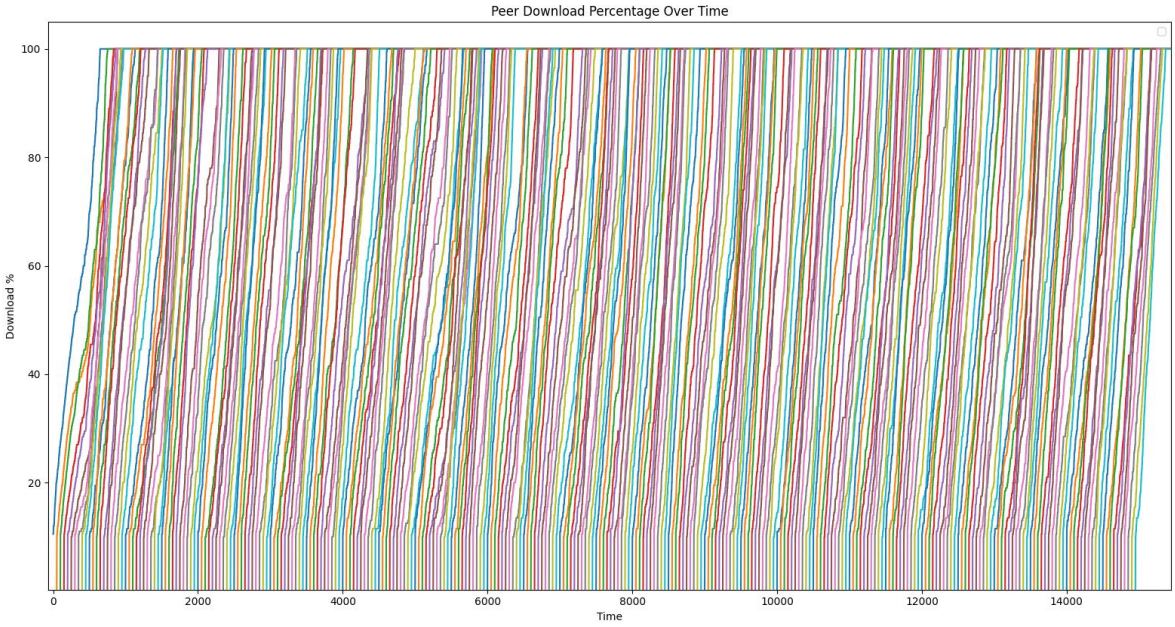
does not scale the download rate with peer count. What would happen to the download rate if now the peers decide to churn?

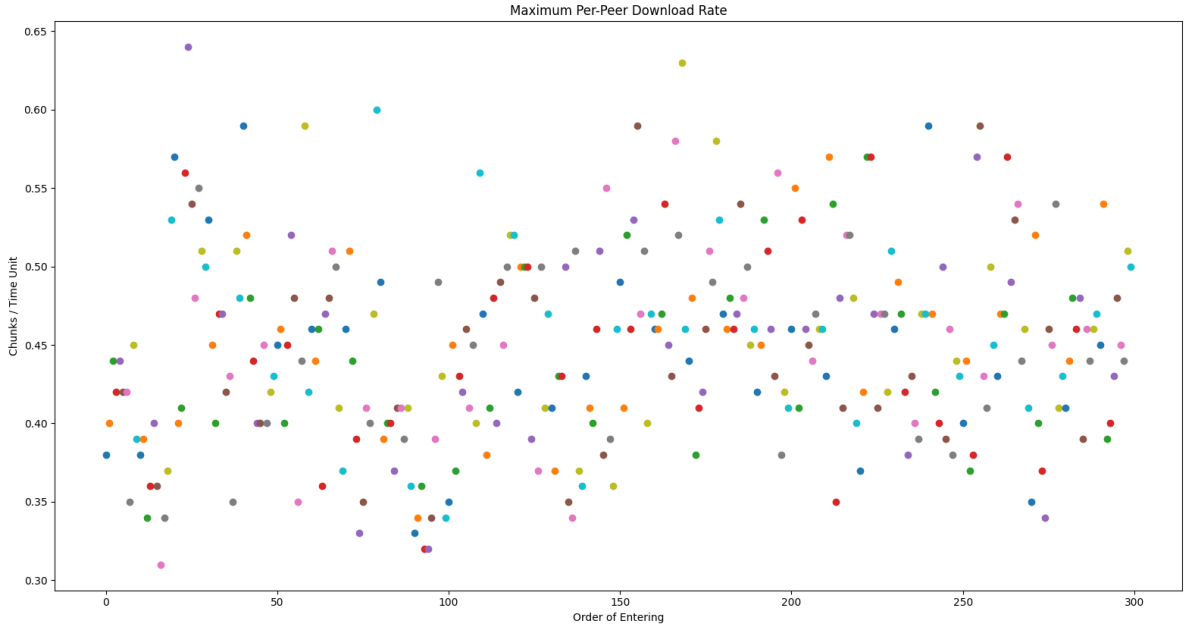
Torrent with 300 Random-Bandwidth, Churning Peers Entering Sequentially at Interval of 5



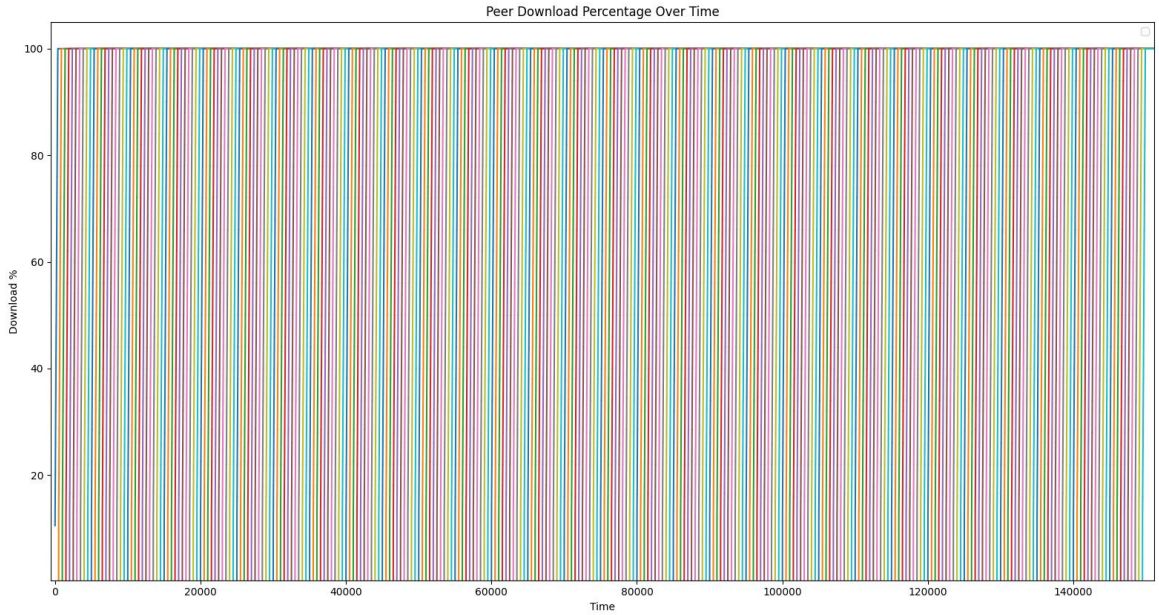


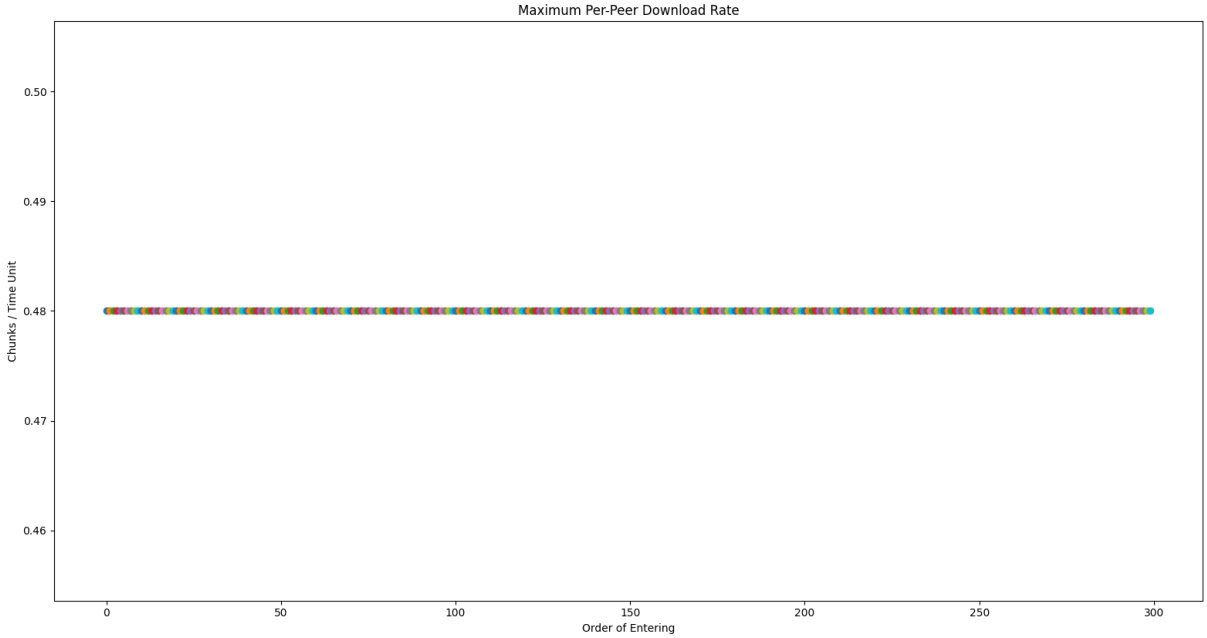
Torrent with 300 Random-Bandwidth, Churning Peers Entering Sequentially at Interval of 50





Torrent with 300 Random-Bandwidth, Churning Peers Entering Sequentially at Interval of 500

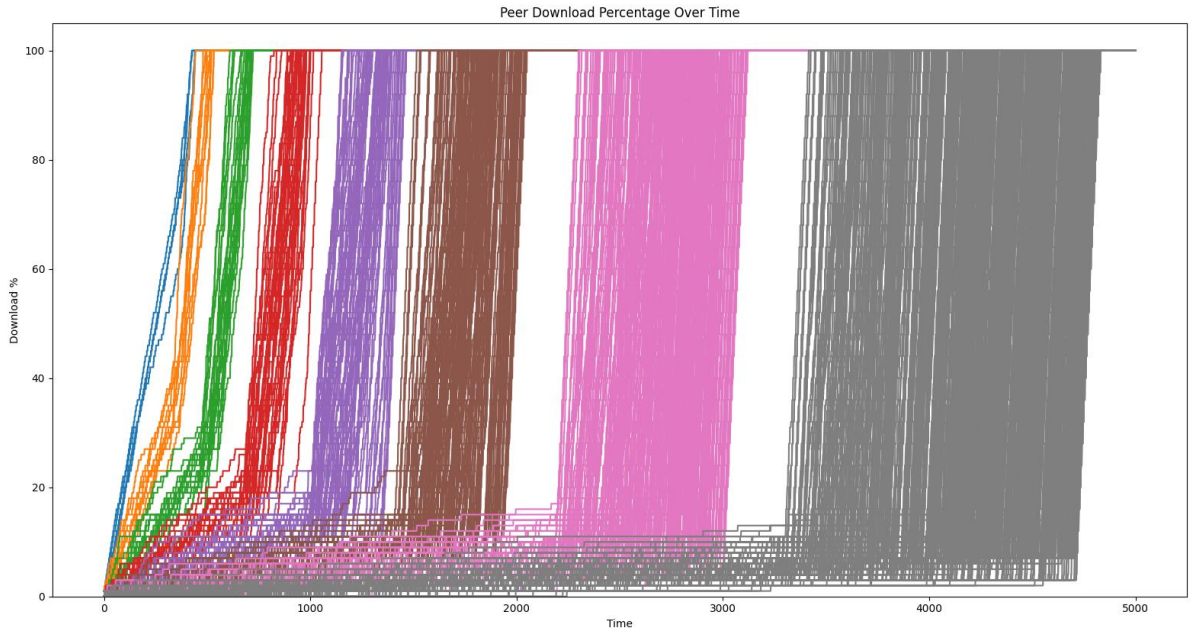


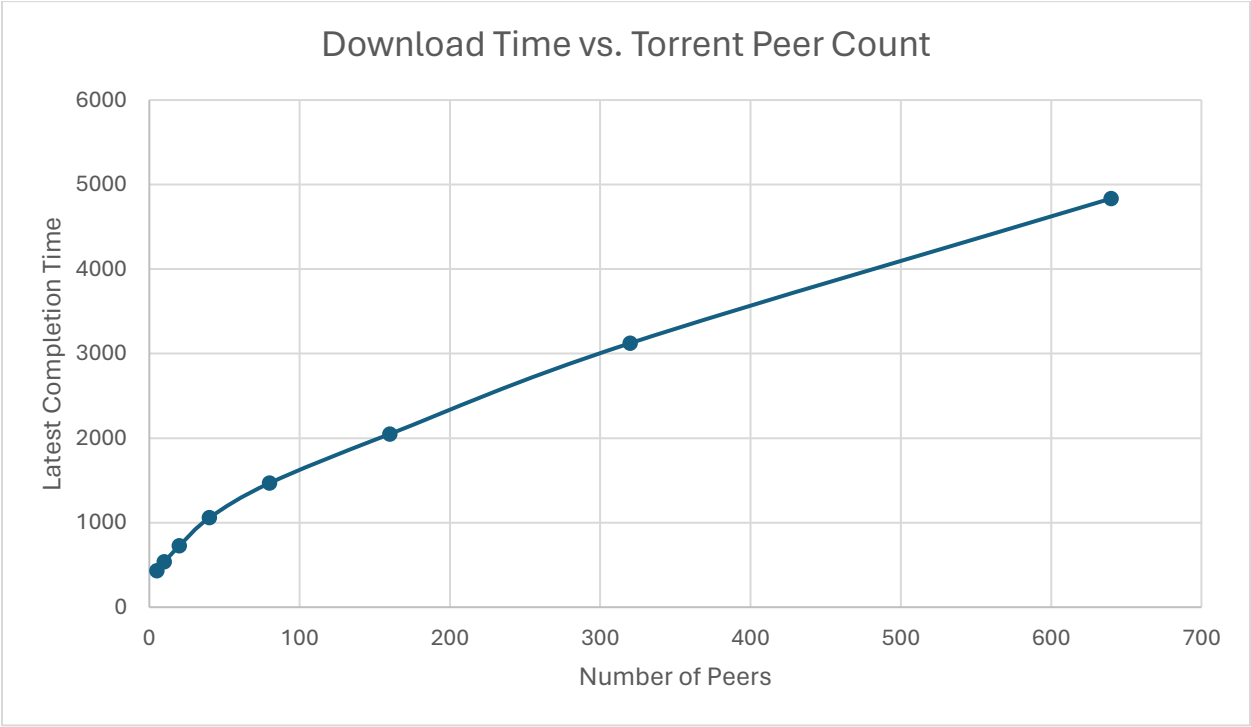


For these three scenarios, we increase the interval at which peers enter the Torrent – this also affects the rate of leaving the Torrent. Because the peers churn, we do not reap the benefits of scalability illustrated by the relatively stable download rates that converge towards the source peer's upload bandwidth the sparser the entering times are. In general, BitTorrent's scalability only works when there exist many participating peers in a Torrent at any given time to facilitate the constant exchange of chunks.

Lastly, we investigate how the file distribution time is scaled in proportion to the number of peers

Torrent with increasing-count Random-Bandwidth Peers





To quantify the distribution time, we simply measure the time upon which the last peer of the Torrent completes downloading. We set off 8 separate Torrents of increasing peer counts – 5, 10, 20, 40, 80, 160, 320, 640 – and measure their completion times. From the plot, we can observe a non-linear increase in the completion time that is inversely proportional to the number of peers. In other words, BitTorrent is very scalable when it comes to file distribution for large peer counts.

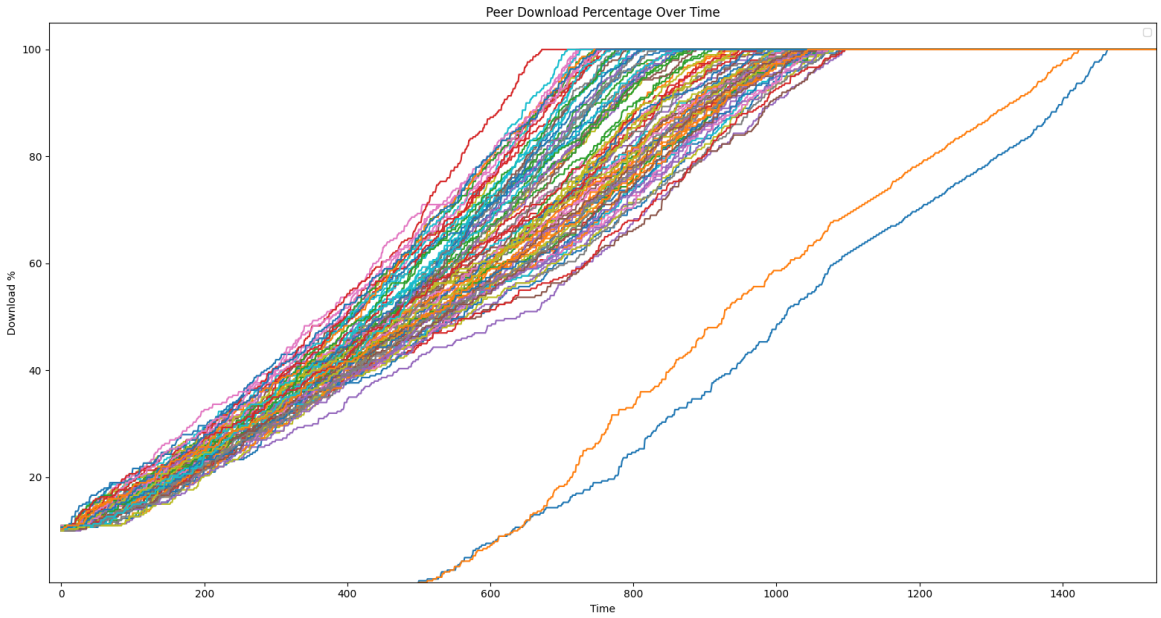
Effectiveness Against Free-Riders

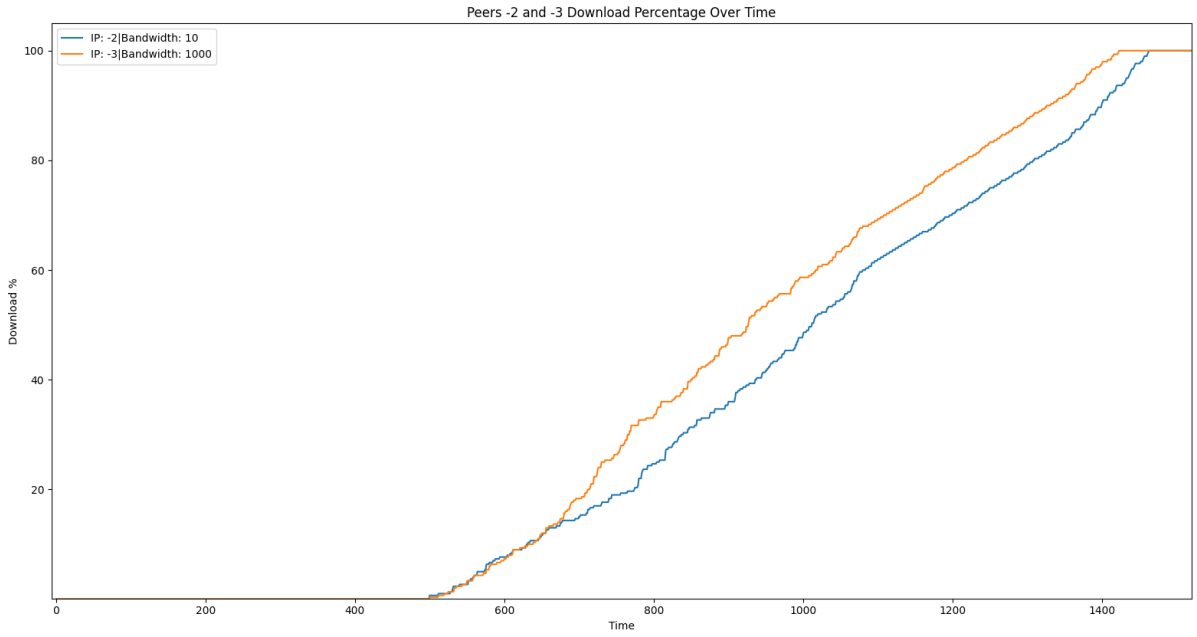
With every herd, there may be bad members that do not contribute equally to solving a problem. In the case of a Torrent, there are so-called “free riders” that can be characterized as peers who do not contribute much to the circulation network of chunks but remain in the tracker to receive chunks unfairly.

The BitTorrent protocol specifies peers to exchange with other peers that give chunks at the highest rates in a tit-for-tat strategy. Though this will effectively drown out the free-riders, it will also discount apparent free-riders that are affected by things out of their control. For example, a peer may have low upload bandwidth due to their ISP or physical location relative to the rest of the Torrent. It is important to not completely ignore such peers and still give them a chance at completing the download. Thus, by introducing optimistic unchokes periodically, the Torrent still provides chunks to free riders without allowing them to climb up the hierarchy of chunk exchange priority.

To investigate the penalty on free riders, we will focus on two injected peers of an ongoing Torrent that contrast in upload bandwidth.

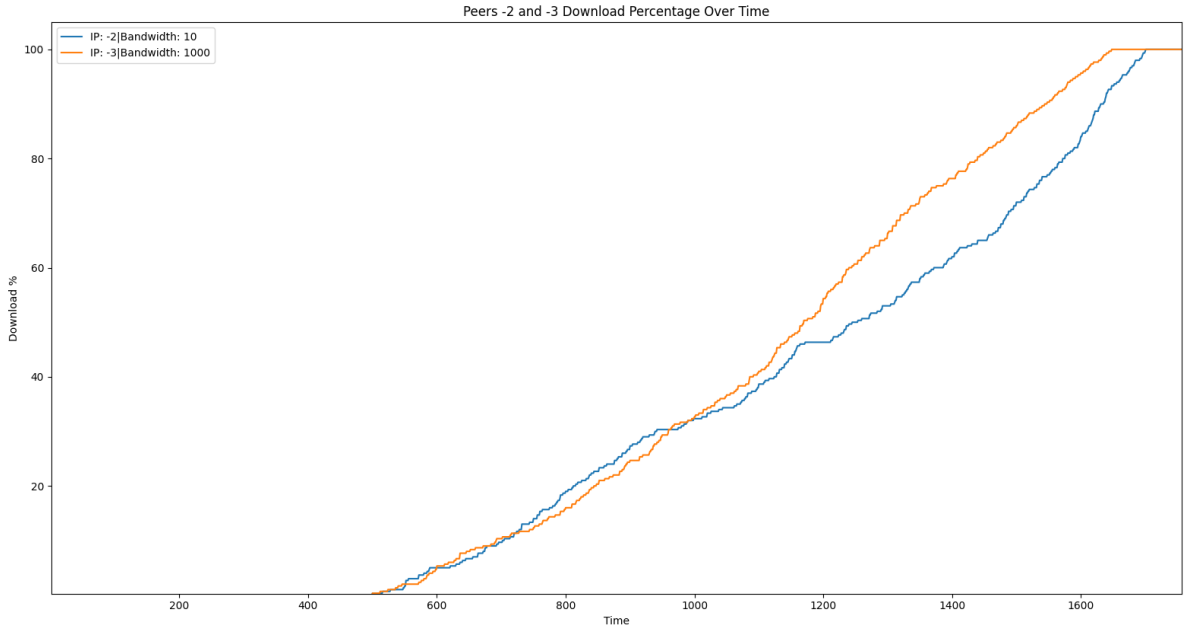
Torrent with 100 random-bandwidth peers





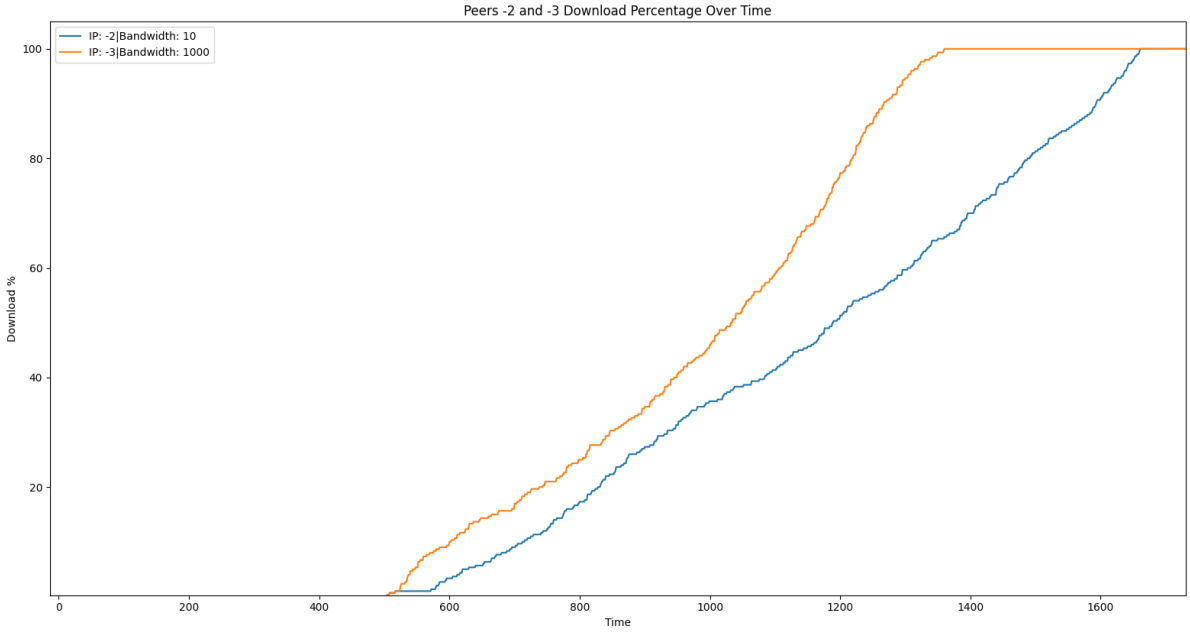
We introduce two new peers into the tracker at time 500: -2 has the minimum bandwidth of 10; -3 has the maximum bandwidth of 1000. As a result of BitTorrent's anti-free-rider mechanisms, the peer with the lower bandwidth has a lower average download rate compared to the higher bandwidth. The initial equality in download rates is caused by both peers not possessing chunks coming into the Torrent, thus they must first only receive what they can get. Later, as they receive chunks that are desired by others, they have a chance to contribute and thus the higher bandwidth peer is recognized and shows up on other peers' top 4. Nonetheless, the lower bandwidth peer still manages to complete its download because of optimistic unchokes. What if we increased the frequency of such unchokes?

Torrent with 100 random-bandwidth peers and twice as many optimistic unchokes



By doing optimistic unchokes more often, the lower bandwidth peer has more chances to receive chunks – this can be seen by the prolonged equality in download rates at the start. However, this phenomenon can only go on for so long before upload bandwidth ultimately prevails and more peers in the Torrent consider the higher bandwidth peer to be part of their top 4. It is a matter of how fast one peer can "show off' it has a superior upload bandwidth.

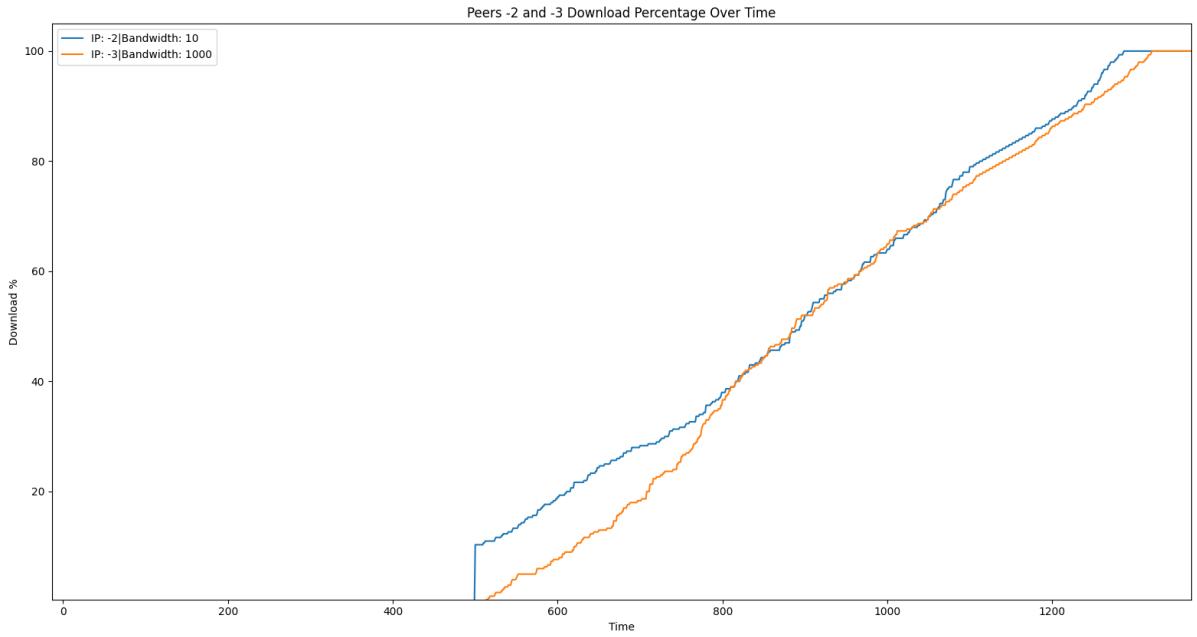
Torrent with 100 random-bandwidth peers and half as many optimistic unchokes



In contrast, if we decreased the frequency of optimistic unchokes by half, the free rider is further drowned out; the higher bandwidth peer's download rate consistently outperforms. The frequency of optimistic unchokes is a parameter to tune such that the rate of incomplete downloads in general is minimized since we still want every peer regardless of bandwidth to eventually receive all the chunks. On the other hand, we also want to penalize free riders, so they do not receive as much as actively contributing nodes.

We have seen the penalty incurred by a free rider's low upload bandwidth, but what if the free rider still has some chunks to contribute?

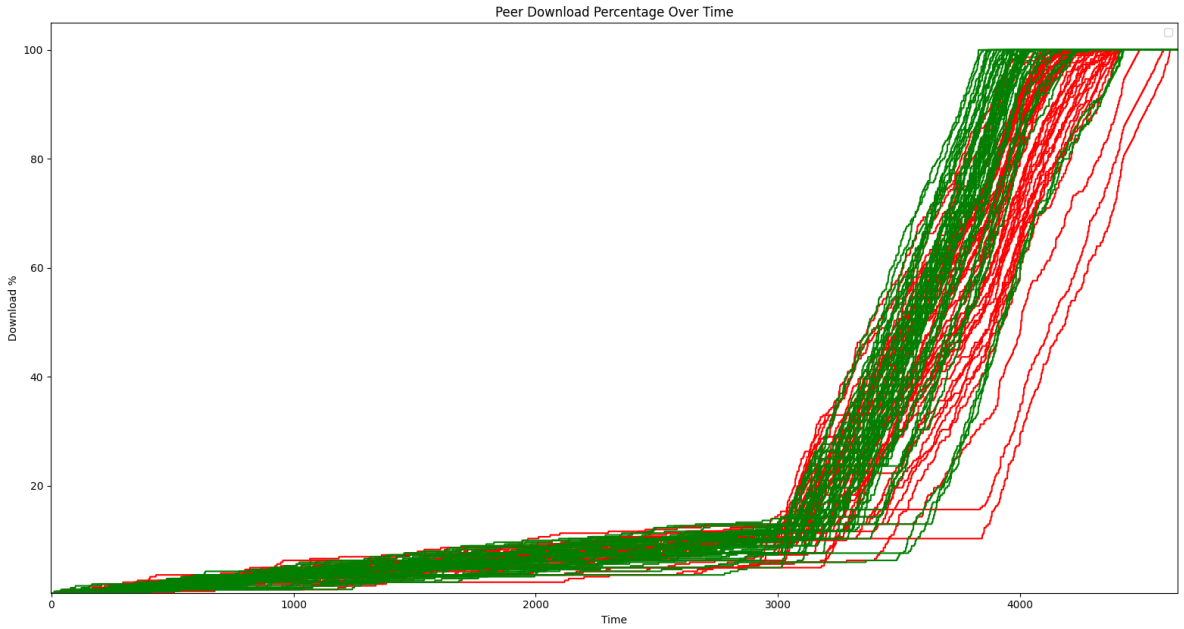
Torrent with 100 random-bandwidth peers with free rider



Although the free rider had a head start compared to the higher bandwidth peer, their download progress eventually converged despite the free rider having something to contribute. This can be explained by its still low upload bandwidth – chunks are useless to a Torrent if they cannot be sent out in a timely manner.

Lastly, we will split a Torrent such that one half are free riders.

Torrent with 100 random-bandwidth peers with half free riders



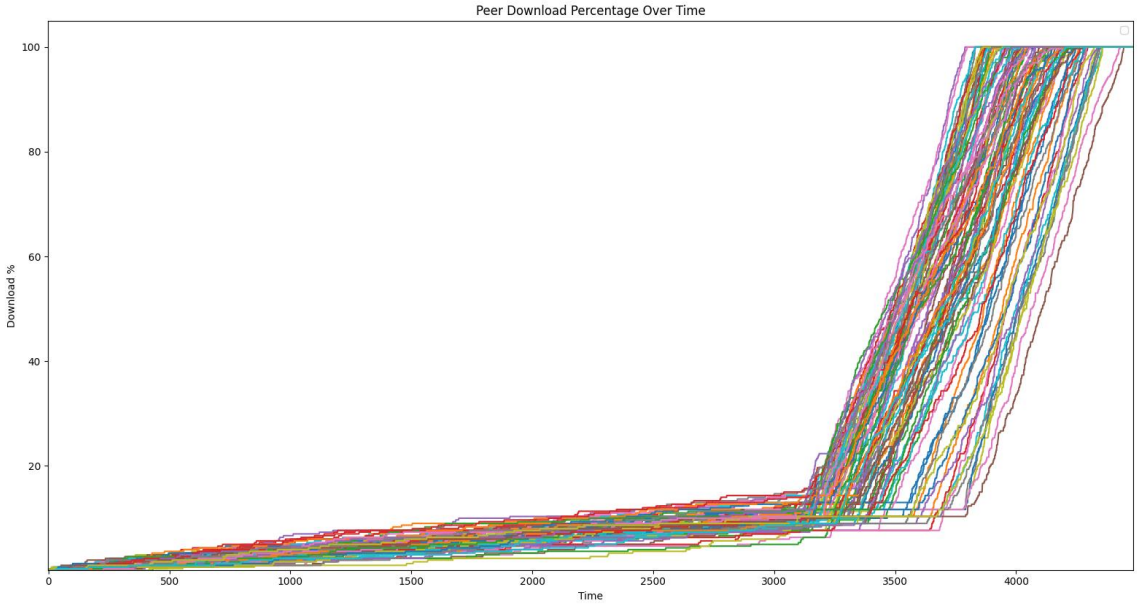
Consistent to previous observations, it is only a matter of time before free riders are drowned out when exchanges become more frequent peer to peer outside of the source peer – free riders can not contribute as fast as other peers and are therefore not considered in top 4 lists aside from the occasional optimistic unchoke. As a result, free riders generally will download at a slower rate.

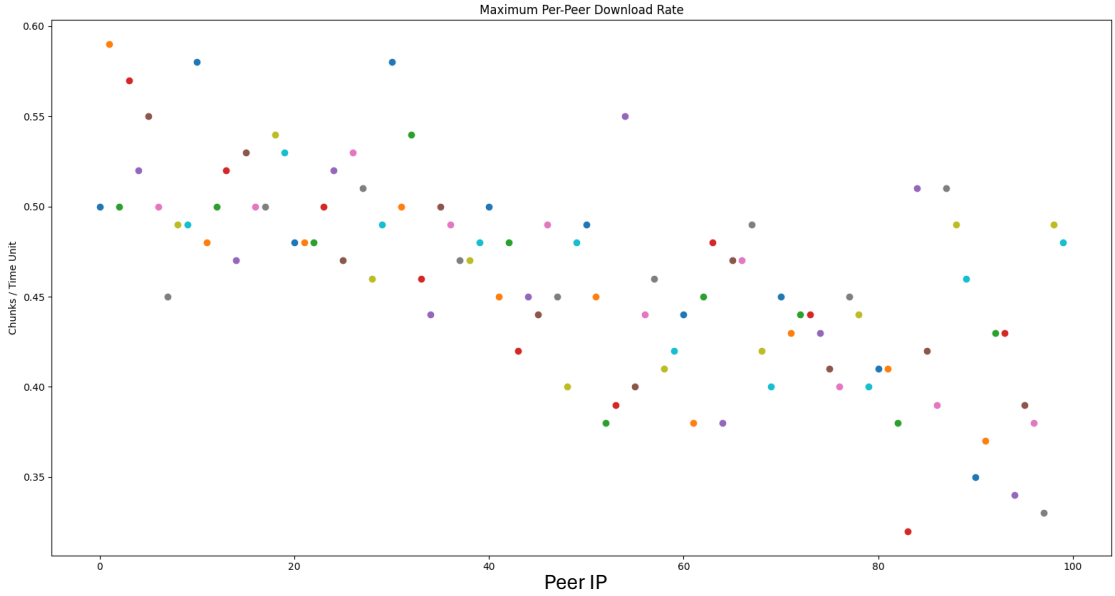
Fairness

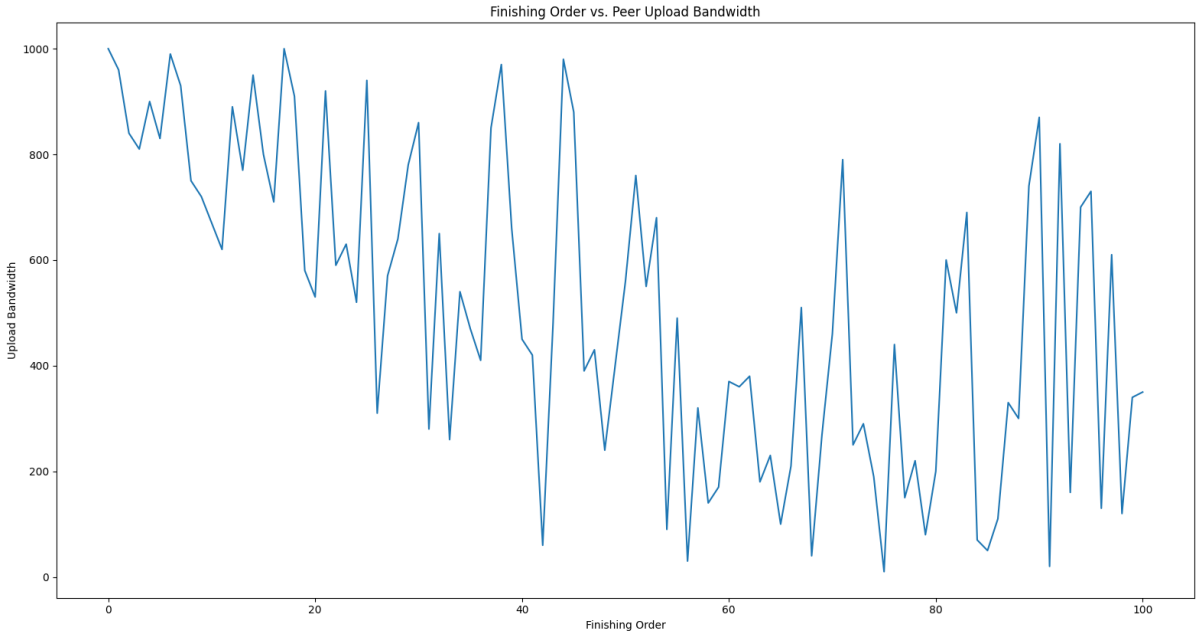
The final aspect of BitTorrent to be investigated is fairness. Due to the diverse conditions of each peer in a Torrent, how do we decide which peers "deserve" a faster download rate? Previously, we have investigated the extreme of free riders who possess the lowest upload bandwidth – BitTorrent consequently reduced their download rates as a penalty.

We now investigate how different upload bandwidths can affect the per-peer download rate as a metric for fairness.

Torrent with source peer and 100 decreasing bandwidth peers



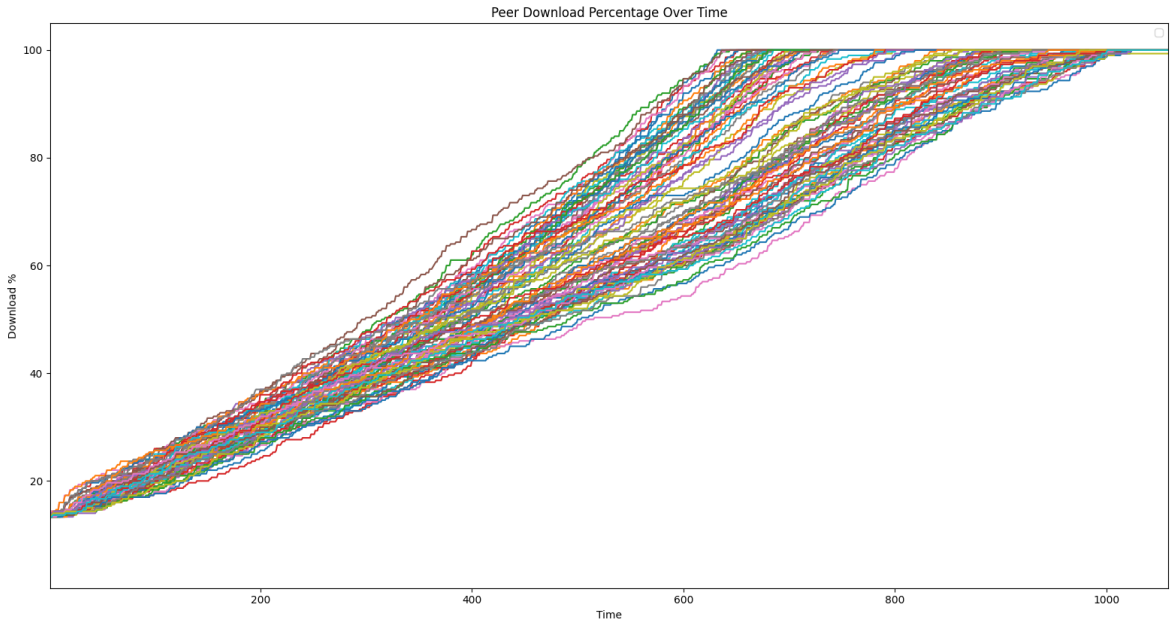


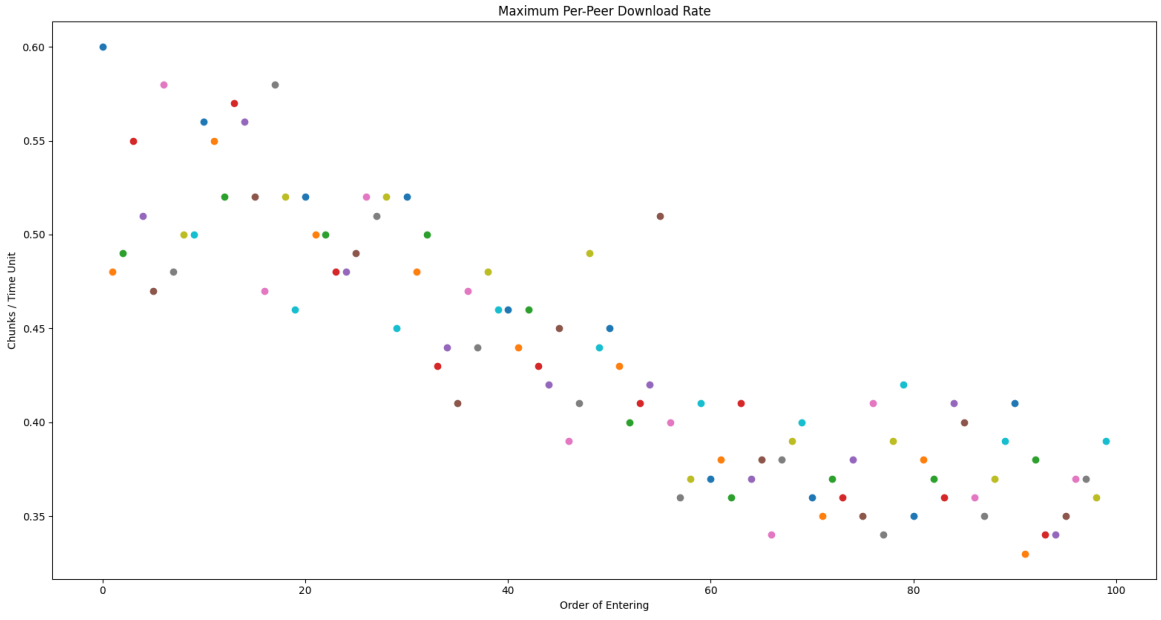


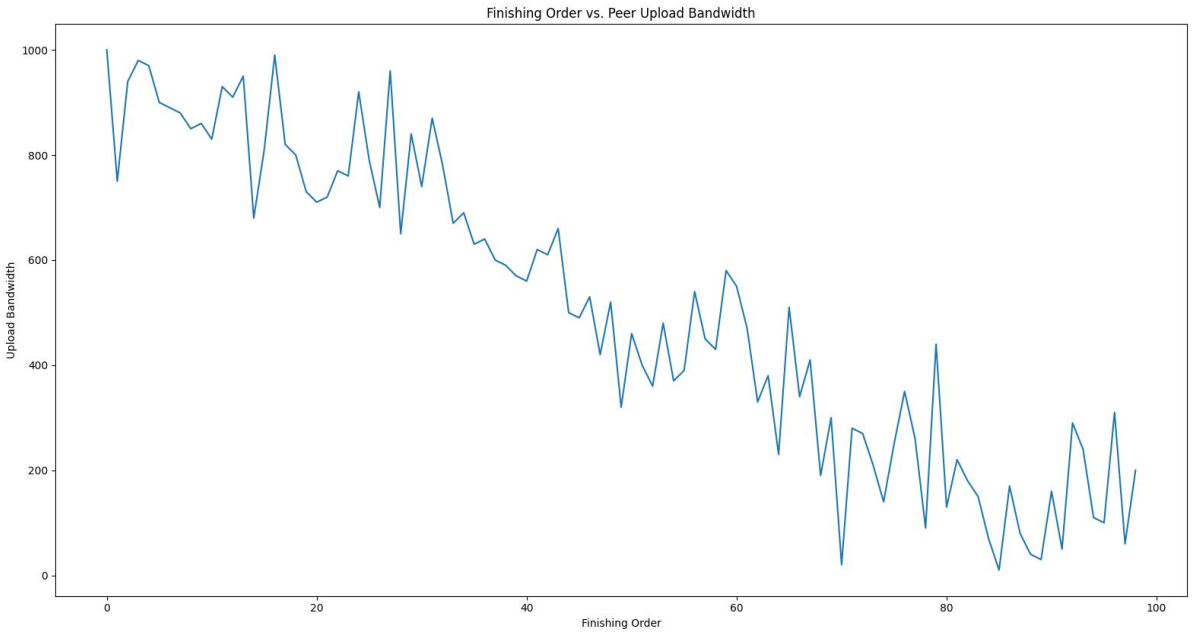
By creating peers with known bandwidths that decrease from the highest bandwidth possible, we can pit them head-to-head in a race to the finish. From the plots of the maximum download rates, there is a correlation with the peer upload bandwidth – the higher the upload bandwidth, the higher the download rate as a reward. Similarly, we can also observe a correlation with the finishing order, where the higher bandwidth peers tend to finish before the lower bandwidth ones.

The existence of the source peer coupled with the empty-handed-start may give lower bandwidth peers a chance to catch up since top 4 peers are not fully evaluated until later into the Torrent, at around time 3000. We can discourage this by removing the source peer entirely and randomly initializing each peer with 15% of the chunks to promote more peer-to-peer communication and top 4 evaluations.

Torrent with no source peer and 100 decreasing bandwidth peers with 15% starting chunks





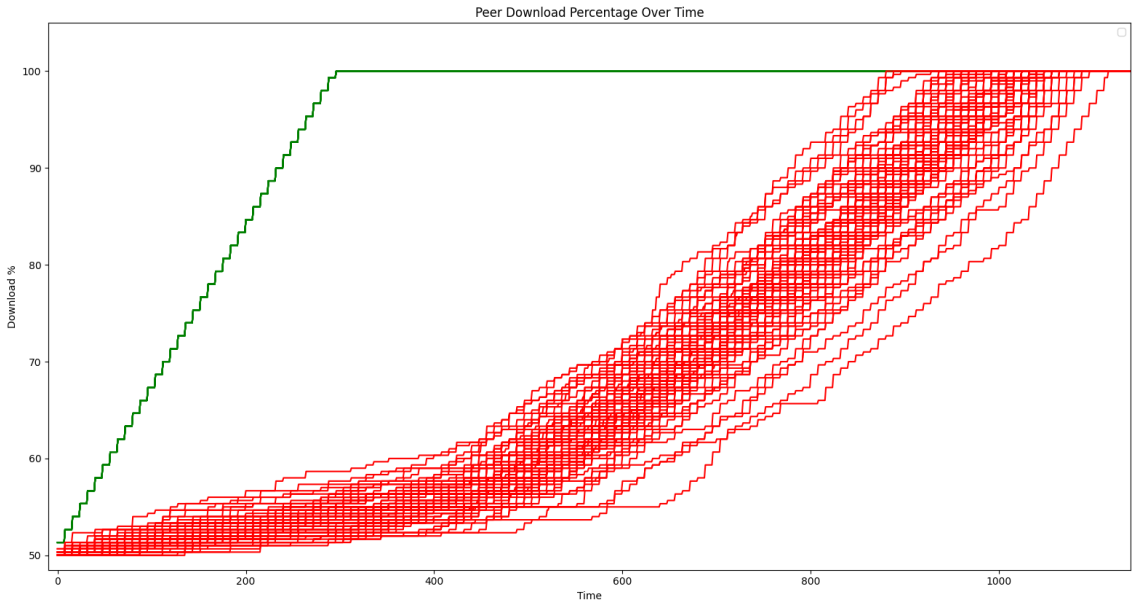


As expected, there is no period of gradual ramp-up in download rate and instead all peers go straight into exchanging with top 4's. As a result, the penalty for fairness' sake it more clearly defined.

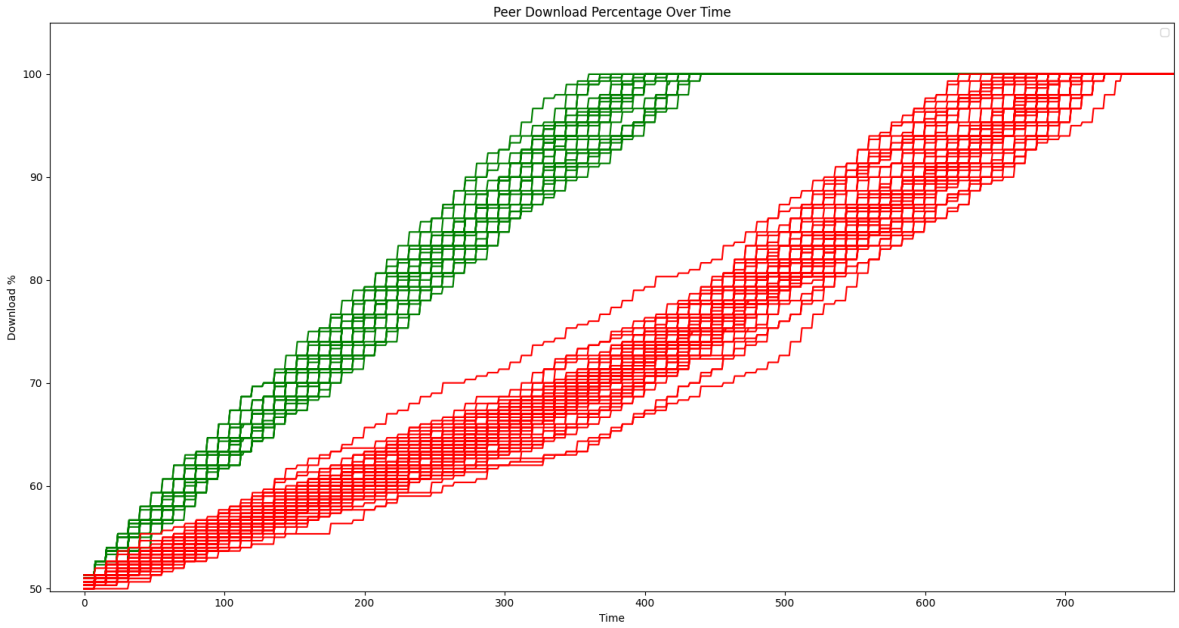
Another metric for fairness would be the composition of chunks that a peer has. Specifically, if a peer entering a Torrent has extremely desirable, rare chunks, it naturally would deserve a higher download rate as it is contributing scarce chunks for the good of the Torrent. To test this, we set all peers to have the same bandwidths but instead vary their starting chunk possessions.

Torrent with no source peer and equal-bandwidth peers

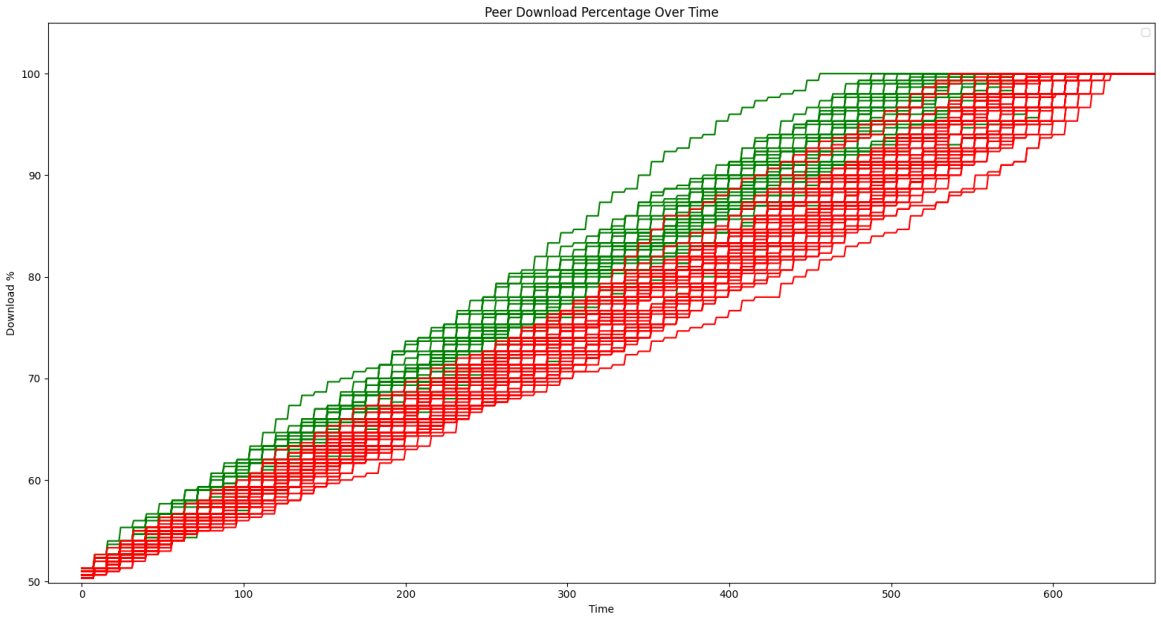
10% of the peers have 50% of the chunks



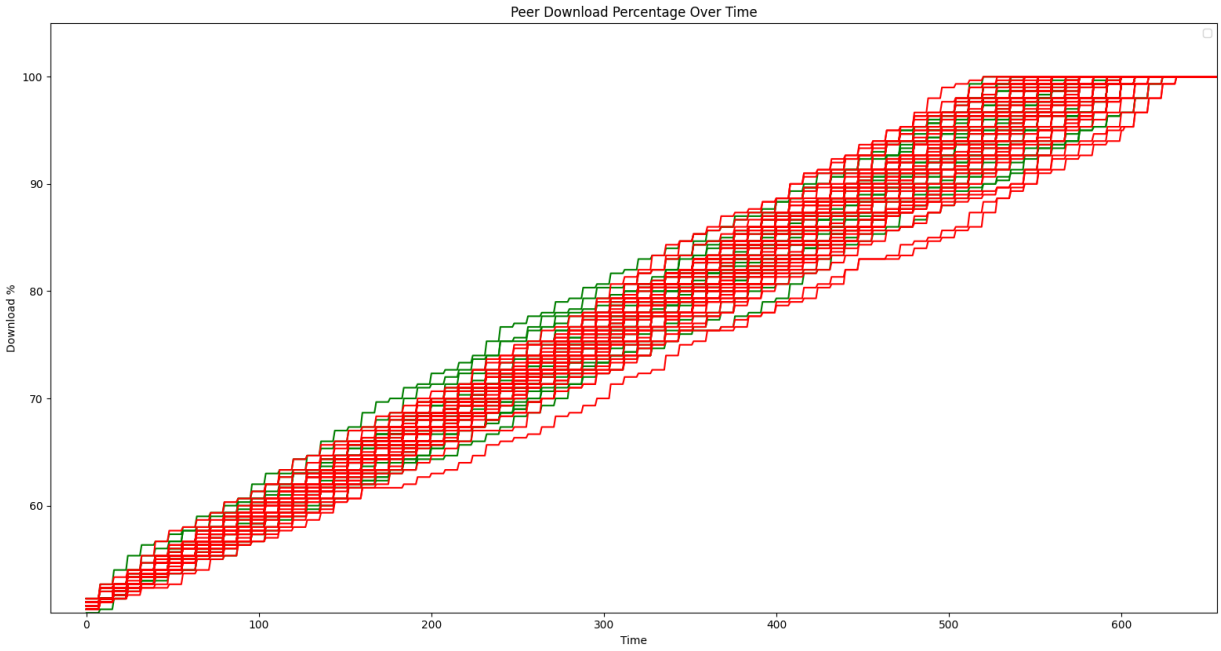
33% of the peers have 50% of the chunks



48% of the peers have 50% of the chunks



50% of the peers have 50% of the chunks



Peers that possess chunks of more rarity will be rewarded with higher download rates. However, if the possession of rare chunks is a common trait in all peers, it averages out such that all peers receive the same download rate. In other words, “if everyone is special then no one is really special.”

Conclusion

A BitTorrent multi-peer simulator has been built to study different behaviors of the P2P protocol. We have run many simulation trials under varying parameters to mimic real-life scenarios on the Internet and have observed well-known benefits of a decentralized file distribution architecture even on a representative model on a smaller scale.