Self-scaling Networks for Content Distribution

Pascal A. Felber, Ernst W. Biersack
Institut EURECOM, 06904 Sophia Antipolis, France
{felber,erbi}@eurecom.fr

***Abstract*— Peer-to-peer networks have often been touted as the ultimate solution to scalability. Although cooperative techniques have been initially used almost exclusively for content lookup and sharing, one of the most promising application of the peer-to-peer paradigm is to capitalize the bandwidth of client peers to quickly distribute large content and withstand flash-crowds (i.e., a sudden increase in popularity of some online content). Cooperative content distribution is based on the premise that the capacity of a network is as high as the sum of the resources of its nodes: the more peers in the network, the higher its aggregate bandwidth, and the better it can scale and serve new peers. Such networks can thus spontaneously adapt to the demand by taking advantage of available resources. In this paper, we evaluate the use of peer-to-peer networks for content distribution under various system assumptions, such as peer arrival rates, bandwidth capacities, cooperation strategies, or peer lifetimes. We specifically try to answer the question: "Do the *self-scaling* and *self-organizing* properties of cooperative networks pave the way for cost-effective, yet highly efficient and robust content distribution?"**

I. Introduction

Cooperative content distribution networks are inherently *self-scalable*, in that the bandwidth capacity of the system increases as more peers arrive: each new peer requests service from, but also provides service to, the other peers. The network can thus spontaneously adapt to the demand by taking advantage of the resources provided by every peer.

As an example of the self-scaling properties of cooperative content distribution, consider the situation where a server must replicate a critical file to a large number of clients, e.g., an antivirus update, to all $100,000$ machines of a large company. Given a file size of 4 MB, and a server (client) bandwidth capacity of 100 Mb/s (10 Mb/s) with 90% link utilization, a classical client/server distribution protocol would distribute the file by iteratively serving groups of 10 simultaneous clients in $u = \frac{32\text{ Mb}}{9\text{ Mb/s}} = 3.55$ seconds. Updating $100,000$ clients would thus necessitate $\frac{100,000}{10}u$, i.e., almost 10 hours.

In contrast, cooperative distribution leverages the bandwidth of the nodes that have already obtained the file, thus dynamically increasing the service capacity of the system as the file propagates to the clients. As each client that has already received the file can serve another client while the server updates 10 new clients, we can compute the number of clients updated at time t as $n(t) = 2n(t-u) + 10 = 2^{\lfloor t/u \rfloor}10 - 10$. Updating $100,000$ clients would thus necessitate less than 1 minute, as can be observed in Figure 1. The exponential increase of peer-to-peer distribution provides a sharp contrast with the linear progression of traditional client/server distribution, and illustrates the self-scaling property of cooperative networks.

II. Cooperative Content Distribution

In order to maximize the participation of each of the peers in the network, large content is typically split into many blocks (or "chunks") that are directly exchanged between the peers—a technique also known as "swarming." The large number and small size of the chunks are key to quickly create enough diversity in the network for each of the peers to be useful to some other peers.

Cooperative networks are usually build incrementally, with joining peers dynamically connecting to existing peers to eventually create complex mesh topologies. In practice, a peer usually knows only a subset of other peers, and actively trades with an even smaller subset. In addition to the actual structure of the mesh (i.e., which and how many neighbors each peers has), two factors are crucial to the global effectiveness of the content distribution process:

- *Peer selection strategy:* which among our neighboring peers will we actively trade with, i.e., serve or request chunks from?
- *Chunk selection strategy:* which chunks will we preferably serve to, or request from, other peers?

The popular BitTorrent [1] tool, which we have studied extensively in [2], empirically selects the peers that offer the best upload and download rates to trade with ("tit-for-tat" strategy). When a new peers joins the system, it initially requests random chunks in order to quickly receive some data and become useful to the system; thereafter, it requests the rarest chunks among those owned by its neighbors, because rare chunks have a higher "trading value" than common chunks.

The main focus of our study is to better understand the potential and the limitations of cooperative networks for content contribution. In particular, we evaluate several peer and chunk selection strategies to determine which ones perform best in various deployment scenarios. For the purpose of our evaluation, we only study the extreme case where each peer knows all other peers (fully-connected mesh) and can potentially trade with any of those peers during its lifetime, although we impose a limit on the number of simultaneous active connections. This assumption allows us to observe the asymptotic behavior of the various cooperative strategies.

A. Deployment Scenarios

In our study, we specifically focus on two deployment scenarios that correspond to real-world applications of cooperative content distribution. In the first scenario, we assume that some critical content need to be quickly replicated on a large number of machines within the private network of a large company. This essentially corresponds to a *push* model where all the peers are known beforehand and distribution stops once the content has been fully replicated on all the machines, which typically have similar connectivity (homogeneous bandwidth).

The second scenario of interest corresponds to the traditional Internet flash-crowd phenomenon, where a large number of clients access almost simultaneously some large popular content. This corresponds to a *pull* model with continuous arrival of the peers. Distribution continues over several peer "generations," with some peers arriving well after the first peers have already left. The clients typically have heterogeneous bandwidth capacities, ranging from dial-up modems to broadband access (asynchronous and synchronous).

B. Notation

We denote by $\mathcal{C}$ the set of all chunks in the file being distributed, and by $\mathcal{D}_i$ and $\mathcal{M}_i$ the set of chunks that peer i has already downloaded and is still missing, respectively (with $\mathcal{M}_i \cup \mathcal{D}_i = \mathcal{C}$ and $\mathcal{M}_i \cap \mathcal{D}_i = \emptyset$). Similarly, $d_i \triangleq |\mathcal{D}_i|/|\mathcal{C}|$ and $m_i \triangleq |\mathcal{M}_i|/|\mathcal{C}|$ correspond to the proportions of chunks that peer i has already downloaded and is still missing, respectively. The function $U(a, b)$ returns a random number uniformly distributed in the interval $[a, b]$.

C. Peer Selection

The peer selection strategy defines "trading relationships" between peers and affects the way the network self-organizes. In our simplified model, we assume that all the peers know one another. When a peer has some chunks available and some free uplink bandwidth capacity, it will use a peer selection strategy to locally determine which other peer it will serve next. In this paper, we propose and evaluate the following peer selection strategies:

- *Random:* A peer is selected at random. This strategy is expected to achieve good diversity in peer connectivity.
- *Least missing:* Preference is given to the peers that have many chunks, i.e., we serve in priority peer j with $d_j \geq d_i$, $\forall i$. This strategy is inspired by the SRPT (shortest remaining processing time) scheduling policy that is known to minimize the service time of jobs [3].
- *Most missing:* Preference is given to the peers that have few chunks (newcomers), i.e., we serve in priority peer j with $d_j \leq d_i$, $\forall i$. The rationale behind this strategy is to evenly spread chunks among all peers to allow them to quickly serve other peers.
- *Adaptive-missing:* Peers that have many chunks serve peers that have few chunks, and vice-versa, with more randomness introduced when download tend to be half complete. A peer i will serve in priority peer j with the lowest rank r_j, computed as:

$$\begin{aligned} r_j^{Rnd} &= U(0,1) \\ r_j^{Det} &= \begin{cases} d_j & : \quad d_i \geq 0.5 \\ m_j & : \quad d_i < 0.5 \end{cases} \\ f &= (1 - |2d_i - 1|)^2 \\ r_j &= f r_j^{Rnd} + (1-f) r_j^{Det} \end{aligned}$$

where r_j^{Rnd} and r_j^{Det} are the random and deterministic ranks of peer j, respectively, and $f \in [0, 1]$ is a weight factor that controls randomness and is maximal when peer i is exactly half-way through the download. This strategy is expected to give good chances to newcomers without artificially slowing down peers that are almost complete.

Although not shown in this paper because of space constraints, we have also experimented with randomized variants of *least missing* and *most missing*, as well as additional strategies that take into account the free bandwidth capacities of the peers.

D. Chunk Selection

The chunk selection strategy specifies which chunks should preferably be traded between the peers. Chunk selection can be performed by the receiver (which requests specific chunks from its neighbors) or by sender (which decides which chunk it will send next on an active connection). With both interaction models, obviously, the chosen chunk must be held by the sender and not by the receiver. In our simplified model, we assume that every peer knows the list of chunks held by its neighbors (i.e., all peers with a fully-connected mesh topology) and that the chunk selection strategy is applied on the sender's side. In this paper, we evaluate the following chunk selection strategies:

- *Random:* The sending peer i selects a chunk $c \in (\mathcal{D}_i \cap \mathcal{M}_j)$ at random among those that it holds and the receiving peer j needs. This strategy ensures good diversity of the traded chunks.
- *Rarest:* The sending peer i selects the rarest chunk $c \in (\mathcal{D}_i \cap \mathcal{M}_j)$ among those that it holds and the receiving peer j needs. Rarity is computed from the number of instances of each chunk held by the peers known to the sender. This strategy is expected to maximize the number of copies of the rarest chunk in the system.

III. Simulation and Evaluation

For the purpose of evaluating cooperative content distribution, we have developed a simulator that models various types of peer-to-peer networks and allows us to observe step-by-step the distribution of large files among all peers in the systems, according to several metrics. Although we have taken extra care to reproduce realistic operating conditions, we have yet made some assumptions in order to simplify and speed up the simulations. In particular, we do not consider failures (peer or network) nor link congestion in any of the experiments, and we do not favor long-running connections overt short connections as real systems usually do. Due to space constraints, we only present here selected results of the simulations of extreme scenarios (little heterogeneity, limited server bandwidth) that best exhibit the differences between the various aforementioned strategies; more moderate scenarios have shown the same general trends, albeit with lower intensity.

A. Methodology and Setup

Our simulator is essentially event-driven, with events being scheduled and mapped to real-time with a millisecond precision. The transmission delay of each chunk is computed dynamically according the link capacities (minimum of the sender uplink and receiver downlink) and the number of simultaneous transfers on the links (bandwidth is equally split between concurrent connections).

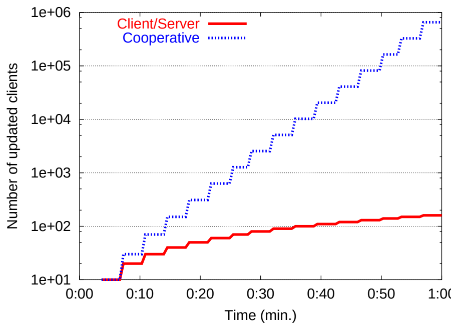

Fig. 1. Scalability of cooperative content distribution: the number of clients that successfully receive a file increases linearly with client/server distribution, and exponentially with cooperative distribution.

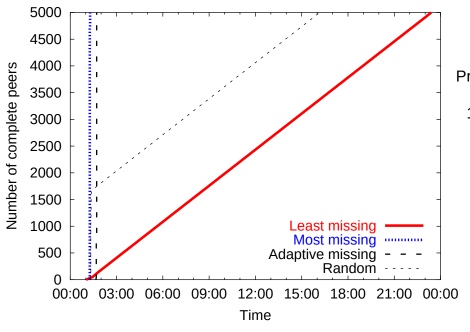

Fig. 2. Completion times for the *random* chunk selection strategy, with simultaneous arrivals, homogeneous and symmetric bandwidth, and selfish peers.

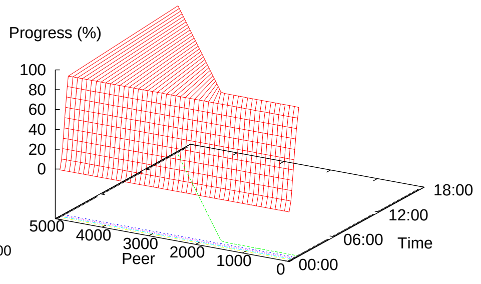

Fig. 3. Download progress for the *random* peer selection strategy, with the *random* chunk selection strategy, simultaneous arrivals, homogeneous and symmetric bandwidth, and selfish peers.

Once a peer i holds at least one chunk, it becomes a potential server. It first sorts its neighboring peers according to the specified peer selection strategy. It then iterates through the sorted list until it finds a peer j that (1) needs some chunks from $\mathcal{D}_i$ ($\mathcal{D}_i \cap \mathcal{M}_j \neq \emptyset$), (2) is not already being served by peer i, and (3) is not overloaded. We say that a peer is overloaded if it has reached its maximum number of connections *and* has less than 128 kb/s bandwidth capacity left. Peer i then applies the specified chunk selection strategy to choose the best chunk to send to peer j. Peer i repeats this whole process until it becomes overloaded or finds no other peer to serve.

Our simulator allows us to specify several parameters that define its general behavior and operating conditions. The most important ones relate to the content being transmitted (file size, chunk size), the peer properties (arrival rates, bandwidth capacities, lifetimes, number of simultaneous active connections), and global simulation parameters (number of initial servers or "origin peers," simulation duration, peer selection strategy, chunk selection strategy). Table I summarizes the values of the main parameters used in our simulations.

B. Simultaneous Arrivals

The chunk selection strategy can have a significant impact on the effectiveness of cooperative content distribution, especially when considering selfish peers. As shown in Figure 2, several of the peer selection strategies need a long time to replicate the file on all clients. First consider that the transmission of all 200 chunks of the file over a 128 kb/s connection requires $\frac{200 \cdot 256 \cdot 8 \text{ kb}}{128 \text{ kb/s}} = 3200$ seconds, i.e., slightly less than one hour. If we could construct a linear chain, with each client receiving the file from the previous peer in the chain and serving it *simultaneously* to the next one, we could theoretically approach this asymptotic limit. In practice, because we only consider the transmission of complete chunks and we share bandwidth capacities between several connections, we expect to experience lower efficiency.

We can explain the low performance of the *least missing* peer selection strategy by the fact that the server will initially only serve the same 5 peers that are closest to completion. These peers will in priority exchange chunks with each other and then slowly propagate some chunks to the other peers, which remain mostly idle because they have no rare chunks to trade. As completed peers leave immediately the system, we essentially have one server (the initial peer) that iteratively serves batches of 5 peers at a time, which explains the low efficiency of the *least missing* strategy. One should note, however, that this strategy minimizes the download time of the first complete peer.

At the other extreme, the *most missing* peer selection strategy tries to make all clients progress simultaneously, thus making them quickly and equally useful to others. This results in a better utilization of the available resources. By "artificially" delaying the departure of the peers, we always keep a large service capacity and ensure that all peers complete approximately at the same time. In the case of simultaneous arrivals, we can observe that the *most missing* strategy minimizes the download time of the last complete peer.

The *random* peer selection strategy is expected to let all peers progress at approximately the same rate, and thus to behave roughly like the *most missing* strategy. We observe, however, that only one third of the peers complete simultaneously and the rest essentially follow the same pattern as the *least missing* strategy. This problem can be tracked down to the *random* chunk selection. Indeed, the chunks that were injected first in the system exist in many instances, while the latter chunks are very rare, with the server doing nothing to correct this imbalance. Most of the peers quickly reach near completion, as shown in Figure 3, but many require much time to obtain the few missing chunks (often just one) that are only held by the origin server.

The *adaptive missing* strategy is interesting because it seems

Parameter	**Value**
Chunk size	256 kB
File size	200 chunks (i.e., 51.2 MB)
Peer arrival rate	
Simultaneous (push)	5000 peers at t_0
Continuous (flash-crowd)	Poisson with rate $\lambda = \frac{1}{2.5 \text{ s}}$
Peer bandwidth (downlink/uplink)	
Homogeneous, symmetric	100% peers: 128/128 kb/s
Homogeneous, asymmetric	100% peers: 512/128 kb/s
Heterogeneous, asymmetric	50% peers: 512/128 kb/s
Peer lifetime	
Selfish	Disconnects when complete
Altruistic	Remains 5 minutes online
Active connections per peer	5 inbound and 5 outbound
Number of origin peers	1 (bandwidth: 128/128 kb/s)
Duration of simulation	12 h or more
Peer selection strategy	*Varies*
Chunk selection strategy	*Varies*

TABLE I
PARAMETERS USED IN THE SIMULATIONS.

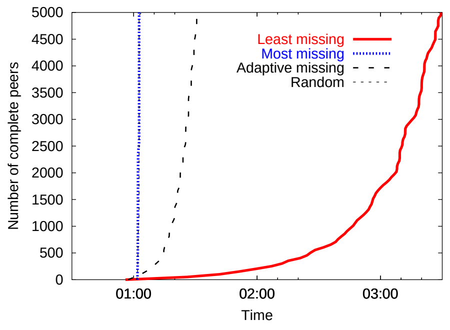

Fig. 4. Completion times for the *rarest* chunk selection strategy, with simultaneous arrivals, homogeneous and symmetric bandwidth, and selfish peers.

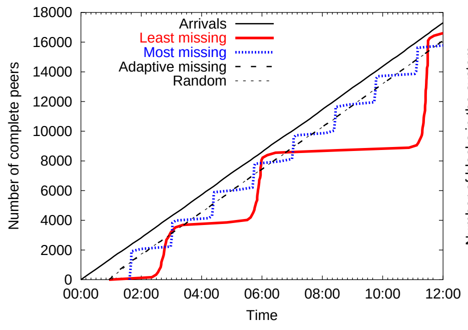

Fig. 5. Completion times for continuous arrivals, with the *rarest* chunk selection strategy, homogeneous and asymmetric bandwidth, and altruistic peers.

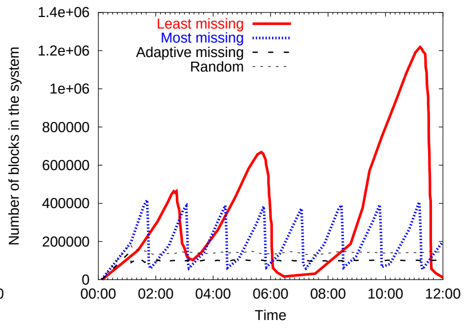

Fig. 6. Chunk capacity of the system, with the *rarest* chunk selection strategy, homogeneous and symmetric bandwidth, and altruistic peers.

to inherit some of the good properties of each of the extreme *least missing* and *most missing* strategies. It initially quickly and evenly replicates blocks in the system and, at the same time, does not artificially prevent near-complete peers to finish their download.

When switching to the *rarest* chunk selection strategy, we observe in Figure 4 significant performance improvements, particularly for the *random* peer strategy that becomes as efficient as *most missing*, and the *least missing* strategy that shows a seven-fold improvement. In contrast to the *random* chunk selection strategy, we do not experience the pathological situation where the origin sequentially serves the rare missing chunks to almost-complete peers.

C. Continuous Arrivals

In the case of continuous arrivals and asymmetric bandwidth ($512/128$ kb/s ADSL) with moderately altruistic peers, we observe in Figure 5 that the *random* and *adaptive missing* peer selection strategies keep up with the arrival rate of the clients, with the latter looking empirically better initially. The *most missing* strategy delays the completion of a first batch of clients, before following the same slope as the arrivals but with notable steps. Finally, the *least missing* strategy shows an odd behavior: the number of complete peers is slow to "take off," then makes a big step to overtake all other strategies, then stalls again for a longer period of time before another even higher step, and so on. To better understand this behavior, consider that the origin peer will iteratively serve groups of 5 peers until they complete their download. The peers of a group will exchange chunks with each other in priority, but also slowly propagate some chunks to other less-complete peers, which will quickly disseminate them among all remaining peers (they cannot indeed serve more-complete peers as the *least missing* strategy would require, because they only have blocks that the more-complete peers also hold). Therefore, we have few peers that complete very fast, and a large majority of peers that progresses slowly but steadily and eventually complete all together.

We can better understand the behavior of the peer selection strategies by considering the chunk capacity of the system with respect to time, shown in Figure 6. The *random* and *adaptive missing* strategies maintain a nearly constant number of chunks in the system. We can note that the latter looks more efficient than the former in this deployment scenario, as it achieves the same completion rate with a lower average chunk capacity. The *most missing* strategy creates a higher chunk capacity by delaying peers until the first batch completes, which corresponds to the sharp drop of chunk capacity. Thereafter, the capacity oscillates with a constant period, driven by the batches of peers that progress and complete together. Finally, the *least missing* strategy exhibits the highest volatility in chunk capacity. The system traverses phases during which it builds an extremely large chunk capacity, and then completely empties it by letting almost all peers terminate simultaneously. Interestingly, the frequency and amplitude of the oscillations increase over time. This corresponds to the steps that we have observed in Figure 5.

IV. Conclusions and Open Issues

The main objective of this paper was to assess the potential of, and make a case for, cooperative content distribution. Based on our preliminary study, it appears that the *self-scaling* and *self-organizing* properties of peer-to-peer networks do indeed offer the technical capabilities to quickly and efficiently distribute large or critical content to huge populations of clients. Cooperative distribution techniques capitalize the bandwidth of every peer to dramatically increase the service capacity of the system. The efficiency of these techniques does, however, depend on many factors. In particular, the chunk and peer selection strategies directly impact the delay experienced by the clients and the global throughput of the system. We did not clearly identify a "best" strategy, as each of them offers various trade offs and may prove most adequate for specific deployment scenarios. Further investigations will be necessary to answer the many open questions raised by our study. In particular, we did not take into account failures nor the *churn* of the system, and it is not clear how such networks behave in the face of malicious or uncooperative clients.

References

[1] B. Cohen, "Incentives to build robustness in BitTorrent," Tech. Rep., `http://bitconjurer.org/BitTorrent/bittorrentecon.pdf`, May 2003.

[2] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five months in a torrent's lifetime," in *Proceedings of the 5th Passive and Active Measurement Workshop*, Apr. 2004.

[3] L.E. Schrage, "A proof of the optimality of the shortest remaining service time discipline," *Operations Research*, vol. 16, pp. 670–690, 1968.