UNIVERSITÀ DEGLI STUDI DI ROMA "TOR VERGATA"

FACOLTA' DI INGEGNERIA

DOTTORATO DI RICERCA IN

Ingegneria delle Telecomunicazioni e Microelettronica

CICLO DEL CORSO DI DOTTORATO: XII

Peer-to-Peer multimedia communication

by

Lorenzo Bracciale

Supervisor

Prof. Stefano Salsano

Coordinator

Prof. Giuseppe Bianchi

Maggio 2010

Summary

Peer-to-Peer (P2P) systems have been invented, deployed and researched for more than ten years and went far beyond the simple file sharing applications. In P2P networks, participants organize themselves in an overlay network that abstracts from the topological characteristics of the underlying physical network. Aim of these systems is the distribution of some kind of resources like contents, storage, or CPU cycles. Users, therefore, play an active role so that they can be considered as client and server at the same time, for the particular service that is provided through the P2P paradigm.

The success of Peer-to-Peer systems is given by the clear benefits of this kind of architecture. Among the most important advantages that come for free with the P2P paradigms we have the lack of a single point of failure as well as the cheaper services that can be offered if users contribute to the provisioning of the service.

Nowadays, indeed, P2P traffic represents a big share of the whole Internet traffic. Besides conventional file-sharing P2P applications, recently P2P streaming systems emerge as a new way to distribute multimedia streams among peers. This technology is made possible by the advances in terms of access bandwidth capacity which connects users to the Internet backbone. Goal of this dissertation thesis is to study these systems, and give contributes in their performance evaluation. The analysis will aim to evaluate the achieved performance of a system and/or the performance bounds that could be achievable.

In fact, even if there are several proposals of different systems, peer-to-peer streaming performance analysis can be considered still in its infancy and there is still a lot of work to do. To this aim, the main contributes of this dissertation thesis are i) the derivation of a theoretical delay bounds for P2P streaming system ii)

the creation of an algorithm that exploits the new insights that come out from the theoretical study iii) the performance evaluation of this algorithm using an ad-hoc simulator, expressly tailored to reproduce the characteristics of the real-world P2P streaming systems, composed by hundred thousands of intermittently connected users.

The dissertation is organized as follows.

In chapter 1 there is a survey of the state of art of p2p streaming systems, including both "structured" systems where connections in the overlay are almost fixed by a strategy, and "unstructured" where connections and partnerships are setted up in a random way.

Chapter 2 focuses on delay bounds in chunk based p2p streaming systems i.e. those systems that choose to split the multimedia content in pieces called chunks whose size is typically bigger than an IP packet. This chapter presents a novel theoretical bound, its asymptotic closed form expression and the demonstration that it is attainable.

Chapter 3 deals with a practical algorithm called O-Streamline based on the theoretical work presented in Chapter 2. Differently from the theoretical bound that has been derived under the strong hypotheses of the presence of a global and centralized vision of the whole network and of the absence of peer churn, O-Streamline works in a distributed manner but it is designed to achieve a delay that is very close to the theoretical bound.

Chapter 4 presents a performance analysis of O-Streamline by means of simulations. These simulations are performed using a novel overlay streaming simulator called "OPSS", expressly tailored to evaluate performance of large scale networks composed by hundred thousands nodes like it happens for real deployed applications. Simulations in a still network and in the case of peer churn show the effectiveness of the O-Streamline algorithm: the achieved delays are very close to the theoretical bound.

Conclusions end the whole dissertation thesis.

In Appendix A, there is a list of my publications related both to P2P multimedia systems, and to other research topics tackled during the Ph.D.

Contents

Chapter 1

State of Art of Peer-to-Peer Multimedia Systems

1.1 Introduction

Overlays and Peer-to-Peer (P2P) networks, initially developed to share files among different users, have gone a long way beyond that functionality. The first P2P network dates back to 1999, when an 18-year-old college dropout named Shawn Fanning stayed awake 60 hours writing the source code of a program for sharing and swapping music files over the Internet, through a centralized index server. That program was Napster [1], the first wide-used P2P file sharing application. From then on, even if a lawsuit filed by the Recording Industry Association of America (RIAA) forced Napster to shut down the file sharing service of digital music, P2P paradigm usage has continued to grow.

In a Peer to Peer network, users ("peers") plays the role of client and server, being both producer and consumer of the service the network offers. In this way, the burden of providing a specific service (e.g. buying servers, bandwidth capacity, network devices) is shared among users and the content provider, making cheaper and easier to setup such kind of network.

Moreover this kind of networks offer implicitly an highly desirable fault-tolerance, self-organization and massive scalability properties.

Each P2P application set up an overlay network that is optimized for the offered

service. We can distinguish between "pure" peer-to-peer systems in which the entire network consists solely of equipotent peers. There is only one routing layer, as there are no preferred nodes with any special infrastructure function. Otherwise we can have "Hybrid" peer-to-peer systems that rely on such infrastructure nodes, often called supernodes. Another characteristic of a peer to peer network is the structure. We have structured peer-to-peer networks where connections in the overlay are almost fixed by a strategy. On the contrary, in unstructured peer-to-peer networks often connections and partnerships are setted up in a random way
All these strategies has their pro and cons. Typically, structured P2P networks obtain better performance at steady-state, but when the amount of peers that leave or join the network increases (peer churn) they could suffer of instability or they should need and high volume of traffic to keep the network consistent to the given structure.

P2P overlays have been deployed in many different application areas.

When users share CPU cycles, they make distributed computing possible. The SETI@home project [2] uses for example a virtual supercomputer composed of large numbers of Internet-connected computers, with the goal of analyzing radio telescope signals from space and detecting intelligent life outside Earth. The Compute Against Cancer [3] program takes advantage from Frontier, a distributed computing platform by Parabon Computation, to understand and reduce the side effects of chemotherapy, study the structure and behaviour of cancer cells and create better ways to screen new cancer drugs.

P2P systems can be used also to share disk storage allows for creating distributed file systems. The authors of [4] propose Cooperative File System (CFS), a peer-to-peer completely decentralized read-only storage system that provides efficiency, robustness and load-balance of file storage and retrieval.

Skype[5] is probably the best examples of how P2P technology may be exploited in the consumer market. Skype is a software application that allows users to make voice calls over the Internet that today has more than 521 millions of users (20,365,656 concurrent Skype users were online November 9, 2009) with a revenue

of more than 185 millions of USD.

P2P file sharing networks are perhaps the most popular P2P application. In this case peers share their own files and, once contents of interest are localized in the network, direct connections are established between peer requesting content and peer storing it to make file transfer possible.

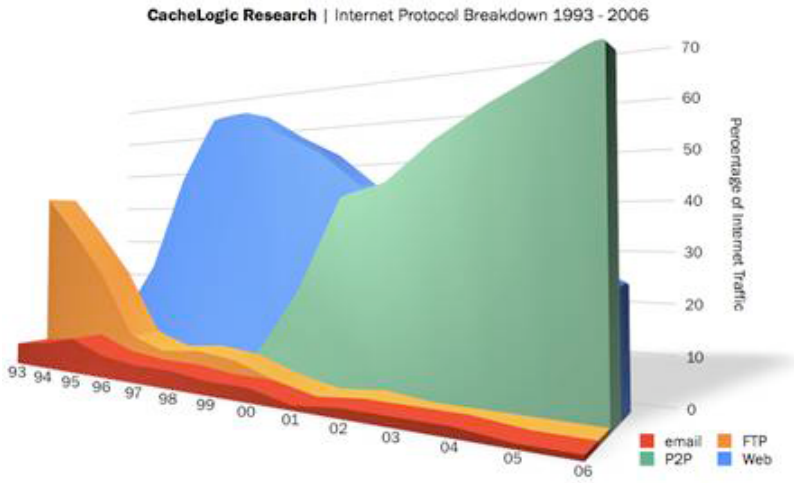

Figure 1.1. Peer to Peer traffic represents a big share of the whole Internet traffic

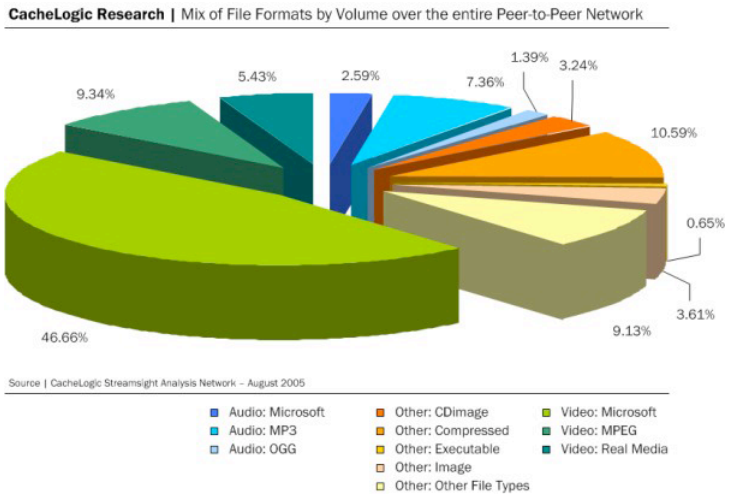

Figure 1.2. Percentage of P2P traffic per type of file

According to measurement studies in Figure 1.1, P2P file sharing traffic dominates Internet traffic. Moreover Figure 1.2 pointed also out a user demand shifting from music files towards video files. BitTorrent [6][7] is undoubtedly the most widespread P2P file sharing application. Gnutella [8][9], KazaA [10][11][12], eDonkey [13] were others example of wide used P2P software for file sharing, at least in the past. All of them have several accidents with law and ended with their closure, while the open source counterpart of eDonkey, eMule, keeps instead on existing by eliminating the presence of any central server that can get in trouble with authorities for the copyright issues. Nowadays eMule, like other P2P programs, uses a completely distributed searching system based on a structured organization on peers called Distributed Hash Table (DHT) .

DHTs assign uniform random NodeIDs from a large space of identifiers to the overlay peers. Data objects or *values* are distinguished with an unique identifiers called *keys*, chosen from the same identifier space. Keys are mapped by the overlay network protocol to a unique live peer in the overlay network so that there is at least one peer in the system that knows where data is (the node with ID nearer to the key). Each peer maintains a small routing table consisting of its neighboring peers' NodeIDs and IP addresses. Lookup queries or messages are forwarded across overlay paths to peers in a progressive manner. In other words, queries are always routed to the peer with the NodeID that is closest to the key in the identifier space. Naturally, different DHT-based systems will have different organization schemes for the data objects, key space and routing strategies. However, regardless of the organization schemes, DHT-based systems may guarantee that any data object can be located in a number of hops that grows on average sub-linearly (usually logarithmically) with the number N of network nodes. Moreover, the underlying network path between two peers can be significantly different from the path on the DHT-based overlay network. Therefore, the lookup latency in DHT-based P2P overlay networks can be quite high and could adversely affect the performance of the applications running over it. Differently from unstructured ones, structured P2P networks can efficiently locate rare items since key-based routing is scalable. They however incur significantly higher overheads than unstructured P2P networks for popular content because all keys are distributed in the same way, regardless from the popularity of the data they refer to.

Recently, the speedup of the users available access bandwidth paved the way to P2P overlay networks aimed to provide video streaming to all the participant. Among the motivation of the success of such kind of applications, there were the explosive growth of multimedia services and applications and the the several difficulties in which incurs the deployment of IP multicast. Even if IP multicast has originally been introduced with the purpose of offering point-to-multipoint content distribution services, many deployment issues have still to be solved. As argued in [14], IP multicast calls for multicast-capable routers able to maintain per group state information, which seriously limit its scalability when groups grow up. Second, IP multicast is a best effort service, and providing higher level features such as reliability, congestion control, flow control, and security has been shown to be more difficult than in the unicast case. Finally, IP multicast requires changes at the infrastructure level, and this slows down the deployment pace.

Due to this, more and more researchers are investigating application level multicast as solution to stream multimedia audio and video content from a source to a large number of end users. This approach consists of end hosts, which according to peer-to-peer (P2P) paradigm auto-organize themselves in an overlay network out of unicast tunnels across participating overlay nodes. Relaying data among overlay nodes allows then the multicast service.

Overlay multicast distribution trees represent the most natural way of extending IP level multicast to application level. To name a few, NICE [15], ZIGZAG [16], CoopNet [17], SplitStream [18] are tree-based P2P streaming systems. However, while tree-based topologies are well suited to dedicated IP multicast routers, they could suffer from re-configurability problems in presence of high churn rate of P2P nodes. In reason of this, overlay mesh-based and unstructured topologies have also been proposed. CoolStreaming/DONet [19] and GridMedia [20] offer examples of the latter approach.

The main disadvantage to using a P2P approach is that it is hard to provide delivery guarantees because peer failures, departures, and disconnections are common.

Goal of all the P2P multimedia streaming systems is to provide data to all the participant of the overlay, with some time constraint given by the streaming nature of the service: usually data must arrive to the user as soon as possible, but no later than a deadline (sometimes called "playback deadline") that is when this data must be used by the user streaming application to play the part of the video data contains.
To achieve this goal, a plethora of very different distribution algorithms have been proposed up to now. These algorithms make specific and different assumptions and choices about the main aspects of a P2P real time streaming system, such as overlay topology, scheduling process and upload strategy.
Sometimes the multimedia flow is divided in pieces called "chunks", and goal of the network is to organize peer nodes in an overlay distribution network to relay chunks. Before summarize the different approaches known in literature and in "real-world" applications, I want to classify the different kind of choices operated by all these approaches.

First categorization regards the overlay topology. It is possible to distinguish among:

- tree-based solutions, such as NICE [15] and ZIGZAG [16], where nodes are organized in a tree and content is recursively spread from the parent node (the streaming source) to its child nodes until all peers are reached;

- mesh-based solutions, such as CoolStreaming [19], Gridmedia [20] or PRIME [21], where each node randomly establishes overlay connections with other nodes and sends a received chunk only to neighbors still missing that chunk, in such a way that each chunk is streamed on a different tree;

- forest-based solutions, such as CoopNet [17] or SplitStream [18], where chunks and nodes are logically organized in a finite set of groups and distribution trees, in such a way that each distribution tree includes all nodes and is used to distribute a single group of chunks. Nodes are interior nodes only in one tree and leaf nodes in all the remaining trees. The idea is to exploit the upload capacity that leaf nodes do not use in separate distribution trees, thus

increasing the overall transmission capacity of the overlay network.

As regards the scheduling process, it is possible to distinguish among:

- push-based algorithms, such as [15] and [16], where supplier nodes decide which chunks will be served to which neighbors;
- pull-based algorithms, such as [19] or [21], where scheduling decisions are taken at receivers and a chunk is transmitted only if a receiver requests that chunk;
- hybrid push-pull algorithms, like [20], where chunks are requested in pull mode at start up and relayed in push mode in the immediate following phase.

Moreover, different local scheduling policies have been proposed: for instance, giving priority to the chunks with more stringent playback deadline [19], to the rarest chunks [22], to the neighbors with the highest upload/download capacity [19][23].
As regards the upload strategy, when the same chunk has to be uploaded to more than one child node, it can be transmitted "in series", e.g. starting the transmission towards a second child node after completion of the chunk upload to the first child, or "in parallel", i.e. sending the same chunk to more than one child node at the same time.

In what follows I will survey the most relevant peer to peer streaming systems.

1.2 Structured P2P Streaming Systems

1.2.1 ZigZag

ZIGZAG [16] deals with the problem of one source towards multiple destinations with consideration of network condition. The goals are to minimize the E2E delay, to manage user dynamicity and to keep the overhead traffic as small as possible to achieve scalability.
To realize this objective, ZIGZAG organizes receivers into a hierarchy of bounded-size clusters and builds the multicast tree based on that. The connectivity of this tree is enforced by a set of rules, which guarantees that the tree always has a height O($log_k N$) and a node degree O(k^2), where N is the number of receivers and k a

constant. The proposed approach helps in reducing the number of processing hops processing to avoid the network bottleneck. At the same time ZIGZAG handles the effects of network dynamics requiring a constant amortized control overhead.
Peers are organized in a multi-layer hierarchy of clusters. The cluster size is upper bounded by $3k$ so that if it has to split because of oversize, the two new cluster are big enough to do not risk to as peers leave.

1.2.2 SplitStream

In *SplitStream* the stream is stripped into k *stripes* to be distributed across a forest of overlay multicast trees. The construction of the overlay multicast trees is based on Scribe [24]. Scribe is an application-level group communication system built on top of Pastry [25], which is a DHT-based self-organizing, structured, P2P overlay network. The key idea is the following: each multicast group is associated with a pseudo-random Pastry key, and the corresponding multicast overlay tree is formed by the union of the Pastry routes from each group member to the root node[1] for that group Pastry key.

In more detail, *SplitStream* uses a separate Scribe multicast tree for each of the k stripes. A fundamental property of the *SplitStream* overlay trees is that they are interior-node-disjoint trees. In other words, *SplitStream* nodes are organized into overlay trees in such a way that each node is interior node in at most one tree and leaf node in the remaining trees. To realize these trees, it uses the Pastry mechanism to forward message: each Pastry node indeed forward each message to node whose nodeId share a progressively longer prefix with the message's key. Since each Scribe multicast tree is formed by the union of the routes from all members to the groupId, nodeIds of all the interior nodes share some number of digits with the tree groupId. Therefore, by choosing groupId that differs in the most significant digit, we are obtaining trees with disjoined sets of interior nodes. The number of stripes determine the incoming bandwidth usage, each node is at least expected to forward k. If a SplitStream node has exceeded its outgoing capacity and receives a

[1]The root node for a Pastry key is the node with the identifier that is numerically closest to the key.

request, one child is dropped according to given criteria. This child node first tries to attach to one of its former siblings. Otherwise, it searches the "Spare capacity group" for a node that is able to provide to it the requested stripe.

1.2.3 CoopNet

Coopnet is an application level multicast system developed to overcome the so called "flash crowd" scenario, where an unexpected increase of demand for a popular content, causes an unsustainable bandwidth request at a server (as happened to the MSNBC streaming server on Sep 11, 2001).
In this sense, a Peer-to-Peer based approach could solve this problem, in reason of its self-scaling propriety, so that the system scale well with the users grows without requiring any new infrastructure. On the other hand, peer churn can make the service highly unreliable. *CoopNet* aims to become resilient to the transience of peers by utilizing redundancy in network paths and redundancy in data. To achieve this goal, the CoopNet protocol anchors itself at a central server that manages multiple distribution trees of peers. The server also encodes data using Multiple Description Coding (MDC) to provide redundancy in data and the encoded data is then distributed to clients using the different distribution trees . MDC is constructed so that any subset allows the client to reconstruct the video so that if a node fails, the user might still get video.

CoopNet protocol is quite simple because it relies on a centralized server. Nodes inform the server when they join and leave they indicate available bandwidth and the delay coordinates so basically the server maintains the trees and nodes could monitor loss rate on each tree and seek new parent(s) when it gets too high.

1.2.4 PTree

The inspiration for PTree was taken by Splitstream. The distribution architecture of *PTree* (formalized in [65]) organizes peers in k overlay trees, in such a way that: i) each tree includes all peers and ii) each peer is interior node in at most one tree and leaf node in all the remaining trees. The root nodes of the k trees are children of

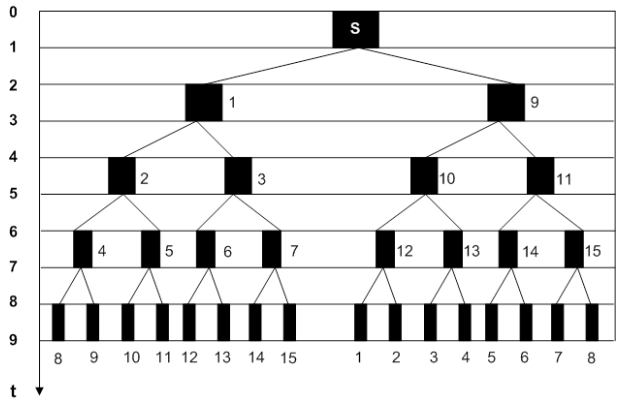

Figure 1.3. PTree distribution architecture; 15 nodes network.

the source. The multimedia flow is divided into chunks and the source always serves k different chunks to its children by means of parallel transmissions; each interior node does the same and serves each received chunk to its k children by means of parallel transmissions. Figure 1.3 illustrates such distribution architecture for the case $k = 2$ and a network of 15 nodes. In this figure the source is denoted with an "S" and nodes are progressively indexed starting from 1. Note that the distribution patterns in figure 1.3 actually relate to the first two chunks and repeat themselves with period $k = 2$. To achieve an homogeneous node operation it has been assumed that the source starts to transmit the first 2 chunks only when both are available. This happens at time unit $t = 1$, i.e. after the time required to transmit a chunk at the rate B. The source ends transmitting the first 2 chunks at time unit $t = 3$. Since at this time two new chunks will be available, it starts transmitting these 2 chunks, and so on.

1.3 Unstructured P2P Streaming Systems

1.3.1 DONET/COOLStreaming

CoolStreaming/DONet [19] is a live media streaming system which constructs a random overlay mesh to distribute the stream segments between the participating overlay nodes. Chunk availability in the node buffer is represented by bit vector called Buffer Map (BM), where a bit 1 and 0 indicate that a segment is respectively

available or unavailable. Each node learns about chunk availability by periodically exchanging its BM with the BMs of its partners, which are the neighbors in the overlay mesh. DONet is built on a pull approach, i.e. scheduling decisions are taken at receivers and chunk transmissions start only if a receiver requests that chunk from a supplier neighbor. Specifically, the proposed heuristic scheduling algorithm gives priority to the chunks with more stringent playback deadline and to supplier neighbors with the highest bandwidth. So the algorithm calculates the number of potential suppliers for each segment and, starting from the segments with only one potential supplier, it selects the supplier with the highest bandwidth and enough available time in case of multiple suppliers.

1.3.2 GridMedia

GridMedia [20] is an unstructured P2P live media streaming system which tries to overcome the limitation of the DONet pull approach to achieve a minor delay. It is based on a push-pull approach that consists in requesting stream packets in pull mode at start up and having nodes relaying stream packets in push mode (e.g. without explicit request) in the immediate following phase.

The same authors as [20] focus in [22] on the optimal streaming scheduling problem in data-driven overlay networks. The optimal streaming scheduling problem aims at addressing how each node optimally decides from which neighbor to request which block, and how it allocates its limited outbound bandwidth to every neighbor, in order to maximize the throughput. This scheduling problem is formulated as a classical min-cost network flow problem and two resolution strategies are considered. The first one is a global optimal solution which assumes a centralized knowledge of all network state, the second one is an heuristic algorithm which is fully distributed and calls for only local information exchange.
Gridmedia was adopted by CCTV to broadcast the CCTV Spring Festival Gala 2005 through Internet and attracted more than 500.000 users all over the world during that night, with the maximum amount of concurrent users reaches as high as 15.239.

1.3.3 PPLive

PPLive [26] is by far the most popular video streaming client. Both protocol and application is proprietary but thanks to some reverse engineering work such [27] we are able to better understand its behaviour. PPlive aims to construct a mesh but it relies also on the help of some servers/supernodes. It exhibits a very aggressive behaviour both in contacting new peers than in using node bandwidth to forward video content. Because of the nature of the streaming whose chunks have hard deadlines, PPLive introduces some buffers (and therefore some delays) to have enough time to react to node failures and to smooth out the jitter. In particular we can distinguish between two buffers: one is managed by PPLive, the second by the media player. A downside of such an architecture is the long startup delay. In PPLive we can consider two types of delay: i) the interval between channel selection and media display (10 to 15 seconds), and; ii) the playback time, required for fluent playback in spite of jitter (10 to 15 more seconds). The time lag between nodes may range up to about one minute.
On January 28, 2006 PPLive delivered a very popular TV program in China, hosting over 200K users, at data rates to users between 400 and 800Kbps, reaching an aggregate bit-rate of 100Gbps.

1.3.4 Prime

In PRIME [21] participating peers form a randomly connected and directed mesh, where all connections are congestion controlled. The incoming and outgoing degrees of individual peers are determined by maximizing the utilization of the incoming and outgoing access link bandwidth. The content is encoded with Multiple Description Coding (MDC) which enables each peer to maximize the delivered quality by pulling a proper number of descriptions. With regard to the content delivery mechanism, PRIME combines push advertisements by parents with pull requests by child peers. The packet scheduling mechanism at child peers selects the packets to be requested according to a global pattern of content delivery that minimizes the probability of content bottleneck among peers. Such pattern consists of the diffusion and the swarming phases. The diffusion phase relates to the new stream segments that have been advertised by parents during the last scheduling event. The swarming phase

relates to the packets that have already been received and are within the playout buffer. Specifically, for each received packet within the playout buffer, the scheduling algorithm determines the number of missing descriptions according to the target delivered quality and assigns each identified packet to a parent that can provide it. The stream source is instead required to implement optimized mechanisms to minimize the potential overlap among the packets delivered to different children. Performance of PRIME has been evaluated via packet-level *ns* [47] simulations. The number of simulated nodes ranges from 100 to 500. Access links are considered symmetrical and are assigned a bandwidth of 700 Kbps and/or 1.5 Mbps respectively in the heterogeneous and homogeneous scenario. The stream has 10 descriptions and each description has bit rate of 160 Kbps.Performance metrics related to delivered quality, content bottleneck occurrence during the diffusion and swarming phase, bandwidth bottleneck, playout buffer capacity and average path length of delivered packets are presented.

Chapter 2

Delay Bounds in Chunk Based Peer-to-Peer Streaming Systems

2.1 Introduction

In this chapter we focus on *chunk-based* systems, where, similarly to most file-sharing P2P applications, the streaming content is segmented into smaller pieces of information called chunks. Chunks are elementary data units handled by the nodes composing the network in a store-and-forward fashion. A relaying node can start distributing a chunk only when it has completed its reception from another node. While the solutions based on multicast overlay trees usually organize the information in form of small IP packets to be sequentially delivered across the trees and can not be regarded as chunk-based, some data-driven solutions, like the ones proposed in [19] [21] [56], may be regarded as chunk-based. A characterizing feature of the chunk-based approach is that, in order to reduce the per-chunk signalling burden, the chunk size is typically kept to a fairly large value, greater than the typical packet size.

In this chapter we raise some very basic and foundational questions on chunk-based systems: what are the theoretical performance limits, with specific attention to delay, that *any* chunk-based peer-to-peer streaming system is bounded to? Which fundamental laws describe how performances depend on network parameters such as the available bandwidth or system parameters such as the number of nodes a

peer may at most connect to? And which are the system topologies and operations which would allow to approach such bounds?

Surprisingly enough, according to the best of our knowledge and our literature survey, these questions have never been directly posed before, and answered, for chunk-based systems (the only references somewhat related to these issues are [57] for the file sharing case, and [58] for the streaming case). The aim of this chapter is to answer these questions. The answer is completely different from the case of systems where the streaming information, optionally organized in sub-streams, is continuously delivered across overlay paths (for a theoretical investigation of such class of approaches refer to [59] and references therein contained). As we will show, in our scenario the time needed for a chunk to be forwarded across a node significantly affects delay performance.

In more detail, we focus on the ability to reach the greatest possible number of nodes in a given time interval (this will be later on formally defined as "stream diffusion metric") or equivalently the ability to reach a given number of nodes in the smallest possible time interval (i.e. absolute delay). We derive analytic expressions for the maximum asymptotic stream diffusion metric in an homogeneous network composed of stable nodes whose upload bandwidth is the same (for simplicity, multiple of the streaming rate).

With reference to such homogeneous and ideal scenario, we show how this bound relates to two fundamental parameters: the upload bandwidth available at each node, and the number of neighbors a node may deliver chunks to. In addition, we show that the serialization of chunk transmissions and the organization of peer nodes into multiple overlay unbalanced trees allow to achieve the proposed bound. This suggests that the design of real-world applications could be driven by two simple basic principles: i) the serialization of chunk transmissions, and ii) the organization of chunks in different groups so that chunks in different groups are spread according to different paths.

This chapter is organized as follows. Section 2.2 surveys the state of the art related to delay in p2p streaming systems. Section 2.3 shows a new discovered bound for streaming. Section 2.4 shows that the bound is also attainable. In section 2.5 there is a performance evaluation of the optimal scheduling solution respect to other kind of simple scheduling strategies. Finally in the section 2.6 there is

a comparison with other literature proposed systems. The work presented in this chapter is realized in team work with professors Giuseppe Bianchi, Nicola Blefari Melazzi, Stefano Salsano and colleagues Francesca Lo Piccolo and Dario Luzzi.

2.2 Related work

The literature abounds of papers proposing practical and working distribution algorithms for P2P streaming systems; however very few theoretical works on their performance evaluation have been published up to now. As a matter of fact, due to the lack of basic theoretical results and bounds, common sense and intuitions and heuristics have driven the design of P2P algorithms so far.

The few available theoretical works mostly focus on the flow-based systems, as they have been defined in subsection 2.3.1.1. In such case, a fluidic approach is typically used to evaluate performance and the bandwidth available on each link plays a limited role with respect to the delay performance, which ultimately depend on the delay characterizing a path between the source node and a generic end-peer. This is the case in [59] and [60]. Moreover, there are also other studies that address the issue of how to maximize throughput by using various techniques, such as network coding [61] or pull-based streaming protocol [62].

This work differs from the previously cited ones mainly because it focuses on chunk-based systems, for which discrete-time approaches are most suitable than fluidic approaches. Surprisingly enough, according to the best of our knowledge and our literature survey, there is only one work [58] where chunk-based systems are theoretically analyzed. In more detail, the author of [58] derives a minimum delay bound for P2P video streaming systems, and proposes the so called *snow-ball* streaming algorithm to achieve such bound. Like the theoretical bound presented in this chapter, the bound in [58], that is expressed in terms of delay in place of stream diffusion metric, can be achieved only in case of serial chunk transmissions and it is equivalent to the one that we found as a particular case when $k \to \infty$. However, the assumptions under which such bound has been derived in [58] are completely different. In fact, with reference to a network composed of $N = 2^l$ nodes excluding the source node, the proposed *snow-ball* algorithm for chunk dissemination requires

that i) the source node serves each one of the $N = 2^l$ network nodes with different chunks, ii) nodes other than the source serve l different neighbors. In other words, the resulting overlay topology is such that i) the source node is connected to all the N network nodes, ii) nodes other than the source have $log_2 N$ overlay neighbors. Due to this, our approach may be definitely regarded as significantly different from the one in [58]. Differently from [58], we indeed consider the case of limited overlay connectivity among nodes and we show that organizing nodes in a forest-based topology allows to achieve performance very close to the ones of the snow-ball case.

2.3 Delay Bound in Chunk Based Peer-to-Peer Streaming Systems

This section addresses the following foundational question: what is the maximum theoretical delay performance achievable by an overlay peer-to-peer streaming system where the streamed content is subdivided into chunks? When posed for chunk-based systems, and as a consequence of the store-and-forward way in which chunks are delivered across the network, this question has a fundamentally different answer with respect to the case of systems where the streamed content is distributed through one or more flows (sub-streams). To circumvent the complexity emerging when directly dealing with delay, we express performance in term of a convenient metric, called "stream diffusion metric" defined in section 2.3.3. We show that it is directly related to the end-to-end minimum delay achievable in a P2P streaming network. In a homogeneous scenario, we derive a performance bound for such metric, and we show how this bound relates to two fundamental parameters: the upload bandwidth available at each node, and the number of neighbors a node may deliver chunks to. In this bound, presented in 2.3.3, k-step Fibonacci sequences do emerge, and appear to set the fundamental laws that characterize the optimal operation of chunk-based systems. The contribute in the theory of n-step Fibonacci sums for which a prior reference result was missing is presented in section 2.3.2.

2.3.1 Motivation

Goal of this section is to clarify why P2P *chunk-based* streaming systems have significantly different performance issues with respect to streaming systems, where the information content continuously flows across one or more overlay paths or trees. Unless ambiguity occurs, such systems will be referred to as, with slight abuse of name, *flow-based* systems. More precisely, we will show that i) theoretical bounds derived for the flow-based case may not be representative for chunk-based systems, and new, *fundamentally different*, bounds are needed, ii) the methodological approaches which are applicable in the two cases are completely diverse, and fluidic approaches may be replaced with inherently discrete-time approaches where, as shown later on, k-step Fibonacci series and sums enter into play.

2.3.1.1 Delay in flow-based systems

We recall that "flow-based" system denotes a stream distribution approach where the streaming information, possibly organized in multiple sub-streams, is delivered with continuity across one or more overlay network paths. Clearly, in the real IP world, continuous delivery is an abstraction, as the streaming information will be delivered in the form of IP packets. However, the small size of IP packets yields marginal transmission times at each node. As such, the remaining components that cause delay over an overlay link (propagation and path delay because of queueing in the underlying network path) may be considered predominant. We can conclude that the delay performances of flow-based systems ultimately depend on the delay characterizing a path between the source node and a generic end-peer. More specifically, if we associate a delay figure to each overlay link, then the source to destination delay depends on the sum of the link delays: the transmission times needed by the flow to "cross" a node may be neglected, or, more precisely, they play a role only because the 'crossed" nodes compose the vertices of the overlay links, whose delays dominate the overall delay performance.

As a consequence, the delay performance optimization becomes a minimum path cost problem, as such addressed with relevant analytical techniques. If we further assume that the network links are homogeneous (i.e. characterized by the same delay), then the problem of finding a delay performance bound is equivalent to

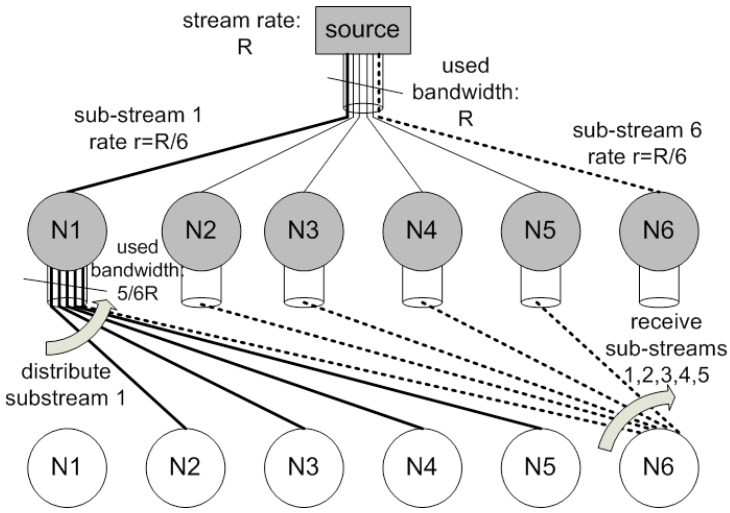

Figure 2.1. Tree depth optimization in flow-based systems. A tree depth equal to 2 can be achieved by i) splitting the stream in a number of sub-streams equal to the number of network nodes N, ii) delivering each sub-stream to a different node, and iii) letting each node i replicate and deliver the i-th sub-stream to the remaining $N-1$ nodes.

finding what is the minimum depth of the tree (or multiple trees) across which the stream is distributed. This problem has been thoroughly addressed in [59], under the general assumption that a stream may be subdivided into sub-streams (delivered across different paths), and that each node may upload information to a given maximum number of children. For instance, if we assume no restriction on the number of children a node may upload to, then it is proven in [59] that a tree depth equal to two is always sufficient. This is indeed immediate to understand and visualize in the special case of all links with a "sufficient" amount of available upload bandwidth - see figure 2.1 for a constructive example[1].

At this stage, it should be clear that, in the context of flow-based systems, as long as some feasibility conditions are met (see e.g. [60]), the bandwidth available on each link plays a limited role with respect to the delay performance achievable. This is clearly seen by looking again at figure 2.1: if for instance we double the

[1]In this example, the amount of available upload bandwidth is "sufficient" in the sense that the source node has a bandwidth at least equal to the stream bit rate R, while each peer node has a bandwidth at least equal to $(N-1) \cdot R/N$, being N the number of peer nodes composing the overlay. As shown in [59] the same result holds under significantly less restrictive assumptions on the available bandwidth.

bandwidth available on each link, the delay performances do not change (at least until the source is provided with a large enough amount of bandwidth to serve all peers in a single hop).

2.3.1.2 Delay in chunk-based systems

Chunk-based systems have a key difference with respect to flow-based systems: the streaming information is organized into chunks whose size is significantly greater than IP packets. Since a peer must complete the reception of a chunk before forwarding it to other nodes (i.e. chunks are delivered in a store-and-forward fashion), the obvious consequence is that delay performance are mostly affected by the chunk transmission time. Thus, in terms of delay performance, the behaviour of chunk-based systems is opposite to the one of flow-based systems. Not only chunk transmission times cannot be neglected anymore with respect to link-level delays (propagation and underlying network queueing), but also we can safely assume that in most scenarios any other delay component at the link-level has negligible impact when compared with the chunk transmission times. This consideration can be restated as: the delay performances of chunk-based systems do not depend on the sum of the delays experienced while travelling over an overlay link, but depend on the sum of the delays experienced while *crossing a node*.

From a superficial analysis, one might argue that the overall delay optimization problem does not change. In fact, the transmission delay of a chunk at a given node could be attributed to the overlay link over which the chunk is being transmitted, and, also in this case, the optimization could be stated as a minimum path cost problem.

However, a closer look reveals that this is not at all the case. The reasons are manifold and can be illustrated with the help of figure 2.2. In this figure, and consistently throughout the chapter, we rely on the following notation. C is the chunk size (in bit); R_{bps} is the streaming constant bit rate (in bps). $T = C/R_{bps}$ is the chunk "inter-arrival" time at the source, being such arrival process a direct consequence of the segmentation into chunks done at the source: a new chunk will be available for delivery only when C information bits, generated at rate R_{bps}, are accumulated (see top of figure 2.2). U_{bps} is the available upload bandwidth, assumed

to be the same for all network nodes, including the source (homogeneous bandwidth conditions). $U = U_{bps}/R_{bps}$ is the normalized upload bandwidth of each node with respect to the streaming bit rate. For simplicity, we consider the case of U integer greater or equal than 1, i.e. U_{bps} being either equal or a multiple of R_{bps}. The *minimum* transmission time for a chunk is equal to $T^* = C/U_{bps} = T/U$; this is true only if the whole upload bandwidth U_{bps} is used to transmit *a single chunk to a single node*. Moreover, we rely on the common simplifying assumption, in overlay P2P systems, that the only bandwidth bottleneck is the uplink bandwidth of the access link that connects the peer to the underlying network (the downlink bandwidth is considered sufficiently large not to be a bottleneck - this is common in practice, due to the large deployment of asymmetric access links - e.g., ADSL).

The first reason why the overall delay optimization problem can not be stated as a minimum path cost problem in the case of chunk-based systems is the sharing of the available upload bandwidth U_{bps} across multiple overlay links. As a consequence, i) it is not possible to *a priori* associate a constant delay cost to overlay links originating from a given node, ii) the delay experienced while transmitting a chunk depends on the fraction of the bandwidth that the node is dedicating to such transmission. For instance, figure 2.2 shows that the source node is transmitting a given chunk in parallel to two nodes; as such, the transmission delay is $C/(U_{bps}/2)$. If the source were transmitting the chunk only to node 1, this delay would be halved.

The second reason is that the transmission time may not be the *only* component of the overall chunk delivery delay. This is highlighted for the case of node N1. After receiving chunk 1, node N1 adopts the strategy of *serializing* the delivery of chunk 1 to nodes N4 and N5. On the one side, in both cases the chunk will be transmitted in the same time, namely C/U_{bps}; this is the minimum transmission time for a chunk, as all the available bandwidth is always dedicated to a single transmission. On the other side, the time elapsing between the instant at which the chunk is available at node N1 and the instant at which the chunk is received by node N5 is greater than the transmission time, as it includes also the time spent by node N1 while transmitting the chunk to node N4.

The third and final aspect which characterizes chunk-based systems in a *streaming* context is that there is a tight constraint which relates the number of peer nodes that can be *simultaneously* served and the available upload bandwidth. If

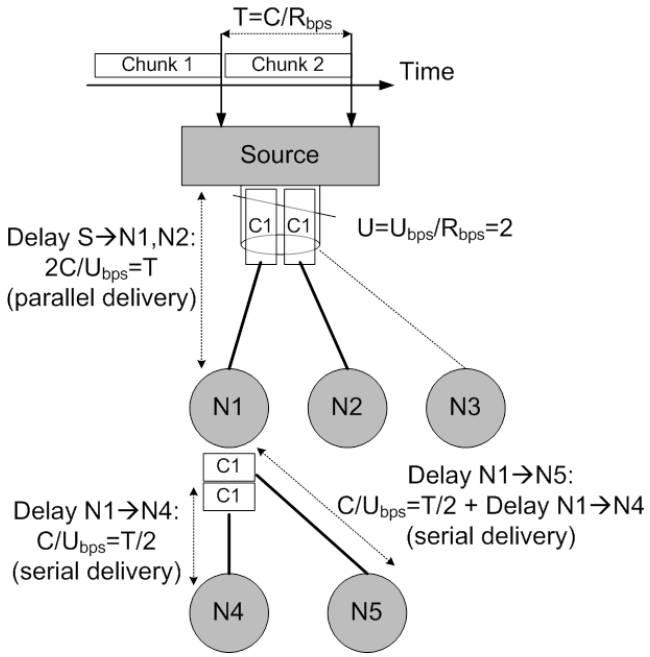

Figure 2.2. Delay components and constraints in chunk-based systems.

we look back flow-based systems in figure 2.1, we see that only practical implementation issues may impede the source node to arbitrarily subdivide the stream into sub-streams, and the tree depth may be indeed trivially optimized by using as many sub-streams as the number of nodes in the network. On the contrary, in chunk-based systems, the number of nodes that can be served is no more a "free" parameter, but it is tightly constrained by the stream rate and the available upload bandwidth. This fact can be readily understood by looking at the source node in the example illustrated in figure 2.2. Due to their granularity, new chunks are available for delivery at the source node every $T = C/R_{bps}$ seconds. Hence, in order to keep the distribution of chunks balanced (i.e., to avoid introducing delays with respect to the time instant at which chunks are available at source and to privilege specific chunks by giving them extra distribution time), the source node must complete the delivery of every chunk before the next new chunk is available for the delivery (i.e. within T seconds). This implies that the source node cannot deliver a single chunk to more than U nodes, being $U = U_{bps}/R_{bps}$ the ratio between the upload bandwidth

and the streaming rate[2].

2.3.2 Mathematical background: some results on k-step Fibonacci Sums

Before stating the bound, we need to provide some preliminary notation and the mathematical background.

Let $F_k(i)$ be the k-step Fibonacci sequence defined as follows:

$$F_k(i) = \begin{cases} 0 & \text{if } i \leq 0 \\ 1 & \text{if } i = 1 \\ \sum_{j=1}^{k} F_k(i-j) & \text{if } i > 1 \end{cases} \tag{2.1}$$

Let $S_k(n)$ be a new sequence defined as the sum of the first n non-null terms of the k-step Fibonacci sequence, i.e.,

$$S_k(n) = \begin{cases} 0 & \text{if } n \leq 0 \\ \sum_{i=1}^{n} F_k(i) & \text{if } n > 0 \end{cases} \tag{2.2}$$

While k-step Fibonacci series have been extensively investigated in related literature, to the best of our knowledge, very few results are available on the sum of k-step Fibonacci series. Since these sums play a fundamental role in our analysis, we derive some new key results concerning them.

Lemma 1 Recursive expression for k-step Fibonacci Sums. *Let $S_k(n)$, with $n \geq 1$, be the sum of the first n terms of a k-step Fibonacci sequence as defined in (2.1). Then, $S_k(n)$ may be recursively computed as:*

$$S_k(n) = 1 + \sum_{i=1}^{k} S_k(n-i) \quad \forall n \geq 1 \tag{2.3}$$

The proof is based on the mathematical induction. Condition (2.3) is immediately verified for $n = 1$. Hence, let us assume that condition (2.3) holds for all indices up

[2]A similar conclusion can be drawn for other nodes as well. Moreover, we remark that this conclusion holds even when chunks are serially delivered, like in the case of node N1.

to n. By applying such condition to $S_k(n)$ and by using the k-step Fibonacci series definition (2.1), it is straightforward to prove that (2.3) holds also for $n+1$:

$$\begin{aligned} S_k(n+1) &= S_k(n) + F_k(n+1) = \\ &= 1 + \sum_{i=1}^{k} S_k(n-i) + \sum_{i=1}^{k} F_k(n+1-i) = \\ &= 1 + \sum_{i=1}^{k} [S_k(n-i) + F_k(n-i+1)] = 1 + \sum_{i=1}^{k} S_k(n+1-i) \end{aligned} \tag{2.4}$$

Lemma 2 Relation between k-step Fibonacci Sums and k-step Fibonacci Series. *The following general relation holds*

$$S_k(n) = \frac{\sum_{i=1}^{k}(i+1-k)F_k(n+i)}{k-1} - \frac{1}{k-1} \tag{2.5}$$

We also observe that the well known result $S_2(n) = F_2(n+2)-1$, relative to the sum of traditional Fibonacci series (i.e., $k = 2$), is a special case of equation (2.5) (achievable for $k = 2$).

The proof requires some algebraic elaboration. We start by reformulating the linear recurrence (2.1) as the following difference equation:

$$F_k(i) + \sum_{j=1}^{k-1} F_k(i+j) - F_k(i+k) = 0 \quad \forall i \geq 1 \tag{2.6}$$

Since this equality holds for any $i \geq 1$, it holds also for the sum

$$\sum_{i=1}^{n} \left\{ F_k(i) + \sum_{j=1}^{k-1} F_k(i+j) - F_k(i+k) \right\} = 0 \quad \forall n \geq 1 \tag{2.7}$$

In addition, $S_k(n) = \sum_{i=1}^{n} F_k(i)$ and the following algebraic manipulations may be

performed on the left-hand member:

$$
\begin{aligned}
&S_k(n) + \sum_{j=1}^{k-1} \sum_{i=1+j}^{n+j} F_k(i) - \sum_{i=1+k}^{n+k} F_k(i) = \\
&= S_k(n) + \sum_{j=1}^{k-1} \left(\sum_{i=1}^{n} F_k(i) + \sum_{i=n+1}^{n+j} F_k(i) - \sum_{i=1}^{j} F_k(i) \right) + \\
&\quad - \sum_{i=1}^{n} F_k(i) - \sum_{i=n+1}^{n+k} F_k(i) + \sum_{i=1}^{k} F_k(i) = \\
&= (k-1)\, S_k(n) + \sum_{i=1}^{k-1} (k-i) F_k(n+i) - \sum_{i=1}^{k} F_k(n+i) + \\
&\quad - \sum_{i=1}^{k-1} (k-i) F_k(i) + \sum_{i=1}^{k} F_k(i) = (k-1)\, S_k(n) + \\
&\quad + \sum_{i=1}^{k} (k-i-1) F_k(n+i) - \sum_{i=1}^{k} (k-i-1) F_k(i)
\end{aligned}
$$

Using the last elaboration and then solving equation (2.7), we achieve

$$
S_k(n) = \frac{\sum_{i=1}^{k} (i+1-k) F_k(n+i)}{k-1} - \frac{\sum_{i=1}^{k} (i+1-k) F_k(i)}{k-1} \tag{2.8}
$$

Equation (2.5) is now proven by noting that the numerator of the second term can be simplified to 1, taking into account that $F_k(1) = 1$ *and* $F_k(i) = 2^{i-2} \quad \forall i : 2 \leq i \leq k+1$.

Lemma 3 Exact non recursive expression for $S_k(n)$. *We now derive a "Binet-like" exact expression for* $S_k(n)$. *As a starting point, we recall that an exact expression has been derived in [63] for the k-step Fibonacci sequence* $F_k(n)$. *This expression, which generalizes the historical Binet's Formula derived for the case of* $k = 2$, *has been conveniently expressed in [64] as*

$$
F_k(n) = \sum_{j=1}^{k} \frac{\phi_{k,j}^{n}}{Q_k(\phi_{k,j})} \tag{2.9}
$$

where $\phi_{k,j}$, $j \in (1,k)$ *are the k (real and complex) roots of the characteristic polynomial*

$$
P_k(x) = x^k - x^{k-1} - x^{k-2} - \cdots - x - 1 = \frac{x^{k+1} - 2x^k + 1}{x - 1} \tag{2.10}
$$

and $Q_k(x)$ is the following sequence of polynomials

$$\begin{aligned} Q_2(x) &= -1+2x \\ Q_3(x) &= -1+4x-1x^2 \\ Q_4(x) &= -1+6x+0x^2-1x^3 \\ \vdots \quad &\vdots \quad \vdots \qquad \vdots \qquad \vdots \\ Q_k(x) &= -1+2(k-1)x+\sum_{i=2}^{k-1}(k-i-2)x^i \end{aligned} \tag{2.11}$$

Thanks to the key relation provided in Lemma 2, we can now substitute the exact expression of $F_k(\cdot)$ (2.9) in (2.5), thus obtaining:

$$S_k(n) = \frac{\sum_{i=1}^{k}(i+1-k)\sum_{j=1}^{k}\frac{\phi_{k,j}^{n+i}}{Q_k(\phi_{k,j})}}{k-1} - \frac{1}{k-1} =$$

$$= \sum_{j=1}^{k}\frac{\phi_{k,j}^{n}}{(k-1)Q_k(\phi_{k,j})}\sum_{i=1}^{k}(i+1-k)\phi_{k,j}^{i} - \frac{1}{k-1} \tag{2.12}$$

Now,

$$\sum_{i=1}^{k}(i+1-k)\phi_{k,j}^{i} = \frac{\phi_{k,j}}{\phi_{k,j}-1}\left[k-1+\frac{1-2\phi_{k,j}^{k}+\phi_{k,j}^{k+1}}{\phi_{k,j}-1}\right] \tag{2.13}$$

The last fraction in (2.13) vanishes, as this is the characteristic polynomial (2.10) computed for one of its roots. Hence, expression (2.12) simplifies to the final expression:

$$S_k(n) = \sum_{j=1}^{k}\frac{\phi_{k,j}}{(\phi_{k,j}-1)Q_k(\phi_{k,j})}\phi_{k,j}^{n} - \frac{1}{k-1} \tag{2.14}$$

Lemma 4 Approximate closed form expression for $S_k(n)$. *The exact expression derived in the prior lemma is not handy, as it requires to handle all the complex roots of the characteristic polynomial (2.10). However, such roots are known to satisfy an important property [63]: only one root has module greater than 1. This root (obviously real) is hereafter referred to as k-step Fibonacci constant ϕ_k. For $k = 2$ it is the most known golden ratio $(1+\sqrt{(}5))/2 = 1.61803$; for growing k, it rapidly tends to the value 2 ($\phi_2 = 1.61803$,$\phi_3 = 1.83929$,$\phi_4 = 1.92756$,$\phi_5 = 1.96595$,$\phi_6 = 1.98358$). Since all the other real and complex roots have modulus*

lower than 1, their contribution in either (2.9) and (2.14) rapidly becomes negligible as the index n grows. As a consequence, the following approximate expression holds:

$$S_k(n) \approx \frac{\phi_k}{(\phi_k - 1)Q_k(\phi_k)}\phi_k^n - \frac{1}{k-1} \tag{2.15}$$

We remark that this expression asymptotically converges to the exact (integer) sequence, and the approximation becomes negligible (within the unit) even for small values of n. For the convenience of the reader, table 2.1 reports the first few values for both the Fibonacci constants and the terms $Q_k(\phi_k)$ which are required to compute (2.15).

k	2	3	4	5	6
ϕ_k	1.61803	1.83929	1.92756	1.96595	1.98358
$Q_k(\phi_k)$	2.23607	2.97417	3.40352	3.65468	3.80162

Table 2.1. k-step Fibonacci constants and $Q_k(\phi_k)$ values

Lemma 5 Derivation of $S_\infty(n)$. *A possibility would be to obtain this as the limit of expression (2.15) for $k \to \infty$[3]. However, there is another trivial alternative way to derive $S_\infty(n)$. It suffices to recognize that $F_\infty(n) = 2^{n-2}$ for $n > 1$, and $F_\infty(1) = 1$, so that*

$$S_\infty(n) = \sum_{i=1}^{n} F_\infty(i) = 1 + \sum_{i=2}^{n} 2^{n-2} = 2^{n-1} \tag{2.16}$$

2.3.3 Stream diffusion metric: a delay-related fundamental bound

Let $\mathcal{P}$ be the set of all peers which compose a P2P streaming network, and let $P = |\mathcal{P}|$ be the cardinality of such network. Let $p \in \mathcal{P}$ be a generic peer in the network. Since the streamed information is organized into subsequently generated chunks, p is expected to receive all these chunks with some delay after their generation at the

[3] *The computation of this limit is not straightforward because of the tight and non trivial dependence of parameters ϕ_k and $Q_k(\phi_k)$ on index k. A way to circumvent this problem is to algebraically transform (2.15) into a function of the only variable ϕ_k and then take the limit for $\phi_k \to 2$. This is possible by exploiting the known property $k = -\log_{\phi_k}(2 - \phi_k)$ related to Fibonacci constants. Details are omitted for reasons of space.*

source. Let us define with $d(c,p)$ the specific interval of time elapsing between the generation of chunk c ($c = 1,2,3,\cdots$) at the source, and its completed reception at peer p. In most generality, different chunks belonging to the stream may be delivered through different paths. This implies that $d(c,p)$ may vary with the chunk index c. Let

$$D(p) = \max_{c} d(c,p)$$

be the maximum delay experienced by peer p among all possible chunks.

To characterize the delay performance of a whole P2P streaming network, we are interested in finding the maximum of the delay experienced across all peers composing the network, i.e.:

$$D\left(\mathcal{P}\right) = \max_{p\in\mathcal{P}} D(p)$$

We refer to this network-wide performance metric as *absolute network delay*. However, for reasons that will be clear later on, this performance metric does not yield to a convenient analytical framework. Thus, we introduce an alternative delay-related performance metric, which we call *stream diffusion metric*. This is formally defined as follows:

$$N(t) = |\mathcal{P}_t| \qquad \text{where} \qquad \mathcal{P}_t = \{p \in \mathcal{P} : D(p) \leq t\}$$

In plain words, $N(t)$ is the number of peers that may receive each chunk in at most a time interval t after its generation at the source.

The most interesting aspect of the stream diffusion metric $N(t)$ is that it can be conveniently applied also to networks composed of an infinite number of nodes (for such networks, obviously, the absolute network delay $D\left(\mathcal{P}\right)$ would be infinite). Moreover, for finite-size networks, it is straightforward to derive the absolute network delay from the stream diffusion metric. Since $N(t)$ is a non-decreasing monotone function of the continuous time variable t and it describes the number of peers that may receive the whole stream within a maximum delay t, for a finite size network composed of P peers the value of t at which $N(t)$ reaches P is also the maximum delay experienced across all peers. The formal relation between the absolute network delay and the stream diffusion metric is hence

$$D\left(\mathcal{P}\right) = \min\{t : N(t) = P\}$$

2.3.3.1 The bound on $N(t)$

Let us assume that propagation delays and queueing delays experienced in the underlying physical network because of congestion are negligible with respect to the minimum chunk transmission time $T^* = C/U_{bps}$, namely the time needed to transmit a chunk by dedicating, to such transmission, *all* the upload capacity of a node. In what follows, we measure the time using, as time unit, the value T^* above defined.

We can now state the following theorem on the upper bound of $N(t)$.

Theorem 1 *In a P2P chunk-based streaming system where all peer nodes have the same normalized upload capacity $U = U_{bps}/R_{bps}$ (assumed integer greater or equal than 1) and k overlay neighbors to delivery chunks to, the stream diffusion metric is upper bounded by*

$$\overline{N}(t) = \sum_{j=1}^{U} S_k(t - j + 1) \tag{2.17}$$

for integer values of t (i.e. multiple of T^*) while, for non integer values of t, $\overline{N}(t) = \overline{N}(\lfloor t \rfloor)$ must be considered.

Proof We preliminarily point out that the minimum amount of time elapsing between the time instant at which a peer receives a chunk and the time instant at which *all* its k neighbors receive the same chunk is lower bounded by k (or equivalently $k \cdot C/U_{bps} = k \cdot T^*$ seconds).

This is a trivial consequence of the fact that the node must replicate and deliver the chunk k times, and hence the amount of time to transmit $k \cdot C$ bits using an upload capacity U_{bps} cannot be lower than the ratio $k \cdot C/U_{bps} = k \cdot T^*$; it is equal to the latter value whenever a work-conserving scheduling discipline is employed.

We now introduce two best-case assumptions representative of an upper bound. First, let us assume that every node in the network, with the obvious exception of the source node (which is feeded with a new chunk every $T = U \cdot T^*$ second), is given sufficient time to deliver a chunk to all its k neighbors. This is equivalent to assume that the next chunk to be delivered by the same node will not arrive before k time units. To avoid misunderstanding, we remark that this implicitly implies that *not all* the chunks received by a node must be further forwarded: section 2.4.3 will discuss what this operatively implies in terms of chunk distribution across the

whole network. Secondly, let us assume that all the k neighbors of a given peer did not receive the considered chunk from other peers.

In order to prove the theorem, with regard to the generic non-source peer X_0, we introduce the *relative stream diffusion metric* $N_{X_0}(t)$, defined as the number of peer nodes which receive a chunk either directly or indirectly (through a neighbor, or a neighbor of a neighbor, etc) from X_0 with a delay lower than or equal to t. We observe that such a delay is evaluated with respect to the time at which peer X_0 completes the download of the considered chunk. Let us include in the count of nodes also the peer X_0 itself. By construction, i) $N_{X_0}(t) = 0 \; \forall t < 0$, as peer X_0 has not yet received the chunk and hence it has not yet started to distribute it further, ii) $N_{X_0}(0) = 1$, as the only peer which can receive the chunk with a null delay is X_0 itself, iii) $N_{X_0}(t)$ is a monotone non decreasing function of t.

Now, let X_1,X_2,$\cdots$,X_k be the neighbors of node X_0, and let d_i be the time interval after which a neighbor node X_i receives the chunk from node X_0. For $t > 0$, the following "pseudo-recursion" holds:

$$N_{X_0}(t) = 1 + \sum_{i=1}^{K} N_{X_i}(t - d_i) \;\; \forall t \geq 0 \tag{2.18}$$

where $N_{X_i}(t - d_i)$ is the relative stream diffusion metric starting from the neighbor X_i at time $t - d_i$, and we use the term "pseudo-recursion" to underline that a neighbor node, in general, may deliver the chunk to its neighbors using a different chunk transmission discipline than node X_0.

The time intervals d_1,d_2,$\cdots$,d_k depend on how the upload bandwidth of node X_0 is allocated to the k transmissions of the considered chunk towards the k neighbors. Let us assume, non restrictively, that $d_1 \leq d_2 \leq \cdots \leq d_k$. Then, it is trivial to prove that $d_i \geq i$, and that equality is achieved if and only if the transmission of chunks is serialized[4]. Based on this, and observing that by construction the relative stream diffusion metric starting from any node is a monotone non decreasing function,

[4] As a sketch of the proof, note that under the non restrictive assumption $d_1 \leq d_2 \leq \cdots \leq d_k$, the only way to achieve $d_i = i$ for a generic value $1 \leq i \leq k$ is to dedicate, in the considered time period i, all the available upload capacity to the i transmissions towards the neighbors X_1,X_2,$\cdots$,X_i and hence defer the transmission of the chunk towards the remaining $k - i$ neighbors after the time interval i has elapsed. However, this applies to all i, starting from the case $i = 1$. Hence, the only scheduling rule which satisfies the equality $d_i = i \forall i$ is the serial transmission.

we can conclude that, for every X_i in equation (2.18), $N_{X_i}(t-d_i) \leq N_{X_i}(t-i)$. Moreover, if we define with $\overline{N_X}(t)$ an upper bound on the relative stream diffusion metric starting from *any* possible node, then, by definition, $N_{X_i}(t-i) \leq \overline{N_X}(t-i)$. Hence, the right part of (2.18) is bounded by:

$$1 + \sum_{i=1}^{K} N_{X_i}(t-d_i) \leq 1 + \sum_{i=1}^{K} \overline{N_X}(t-i) \quad \forall t \geq 0 \tag{2.19}$$

In homogeneous conditions, the bound on the relative stream diffusion metric is independent of the specific non-source node which diffuses the stream. Hence we can conclude that the upper bound can be computed as the solution of the following recurrence:

$$\overline{N_X}(t) = 1 + \sum_{i=1}^{K} \overline{N_X}(t-i) \quad \forall t \geq 0 \tag{2.20}$$

By using the Lemma 1, and setting as applicable initial conditions $\overline{N_X}(t) = 0$ for $t < 0$ and $\overline{N_X}(0) = 1$, we can conclude that for integer values of t

$$\overline{N}_X(t) = S_k(t+1) \tag{2.21}$$

where $S_k(n)$ is the k-step fibonacci sum as defined in (2.2). We also observe that $\overline{N_X}(t)$ may change value only when t is integer, so that for non-integers values of t $\overline{N_X}(t) = \overline{N_X}(\lfloor t \rfloor)$.

Theorem 1 is now readily proven by considering that the stream diffusion metric $N(t)$ differs from the just found relative stream diffusion metric because it represents the number of nodes which have received a chunk starting from the source node S (rather than a generic node X), and not including the source node in the count. Since i) in the time interval elapsing between the arrival of a chunk and the arrival of the following chunk, the source node can deliver the chunk to at most U neighbors, and since, in analogy with what above demonstrated, ii) the most efficient chunk transmission policy is to serially transmit the chunk to the U considered nodes, we can finally conclude that, for any integer value of t

$$\overline{N}(t) = \sum_{i=1}^{U} \overline{N_X}(t-i) = \sum_{i=1}^{U} S_k(t-i+1) \quad \forall \geq 0 \tag{2.22}$$

2.3.3.2 Asymptotic closed form expressions for the bound on $N(t)$

Thanks to the asymptotic expression of k-step Fibonacci Sums, which has been derived in 2.3.2 , equation (2.17) can be more conveniently expressed in the following asymptotic closed form:

$$\begin{aligned}\overline{N}(t) = \sum_{j=1}^{U} S_k(t-j+1) \approx \sum_{j=1}^{U} \frac{\phi_k \cdot \phi_k^{t-j+1}}{(\phi_k - 1)Q_k(\phi_k)} + \\ - \sum_{j=1}^{U} \frac{1}{k-1} = \frac{\phi_k^2(1-\phi_k^{-U})}{Q_k(\phi_k)(\phi_k-1)^2} \cdot \phi_k^t - \frac{U}{k-1}\end{aligned} \tag{2.23}$$

where i) ϕ_k represents the so said k-step Fibonacci constant and it is the only real root with modulo greater than 1 of the characteristic polynomial $P_k(x) = x^k - x^{k-1} - x^{k-2} - \cdots - x - 1$ of the k-step Fibonacci sequence, and ii) $Q_k(x)$ is a suitable polynomial already introduced in 2.3.2.

For the convenience of the reader, the first few values of the Fibonacci constants are $\phi_2 = 1.61803$,$\phi_3 = 1.83929$,$\phi_4 = 1.92756$,$\phi_5 = 1.96595$,$\phi_6 = 1.98358$, while the first few values of the terms $Q_k(\phi_k)$ are $Q_2(\phi_2) = 2.23607$,$Q_3(\phi_3) = 2.97417$,$Q_4(\phi_4) = 3.40352$,$Q_5(\phi_5) = 3.65468$,$Q_6(\phi_6) = 3.80162$.

The derived bound explicitly accounts for the fact that each node at most can feed k neighbors. If this restriction is removed, we obtain a more simple and immediate expression

$$\overline{N}(t) = \sum_{j=1}^{U} S_\infty(t-j+1) = \sum_{j=1}^{U} 2^{t-j} = 2^t(1-2^{-U}) \tag{2.24}$$

We can compare the results given by the approximation formula and the exact solution. Figure 2.3 shows the time $T(n)$ needed by n peer nodes to complete the download of the 100-th chunk as a function of the number n of peer nodes and for two values of the number k of parents/children. We observe that the asymptotic approximate expression tends to be exact for large values of t but is fairly accurate already for very small values of t.

The same behavior can be observed in figure 2.4, which shows the maximum number of peer nodes $N(t)$ that can complete the download of the 100-th chunk as a function of the time t and for two values of the number k of parents/children.

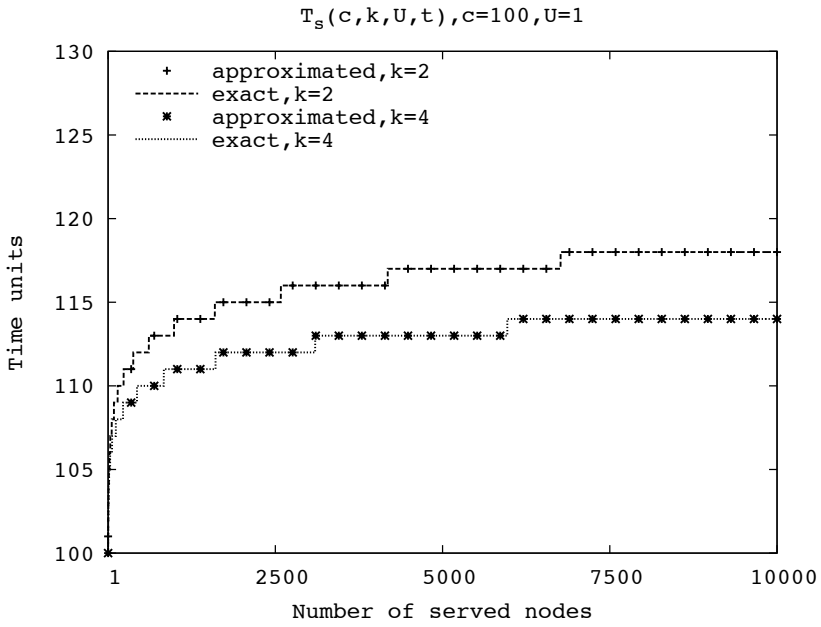

Figure 2.3. Asymptotic and exact evaluation of the time units needed by n peer nodes to complete the download of the 100-th chunk as a function of the number n of peer nodes, and for two values of k.

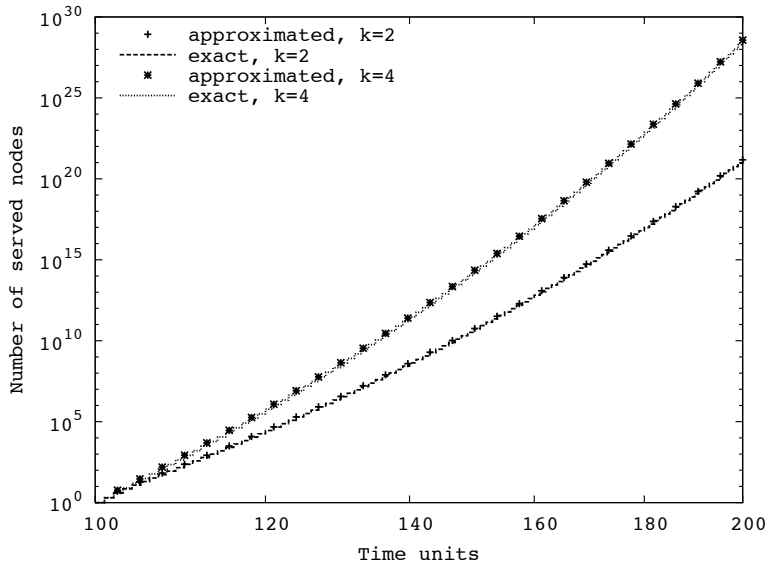

Figure 2.4. Asymptotic and exact evaluation of the maximum number of peer nodes that can complete the download of the 100-th chunk as a function of the time t, and for two values of k.

2.4 Attaining the bound

The provided bound offers only limited insights on how chunks should be forwarded across the overlay topology. Specifically, the bound clearly suggests that delay performances are optimized only if chunks are serially delivered towards the neighbor nodes, but does not make any assumption on which specific paths the chunks should follow, or in other words, which overlay topologies should be used. We now show that, to attain the performance bound, peer nodes have to be organized according to i) an overlay unbalanced tree if $k = U$, ii) multiple overlay unbalanced trees if $k > U$ and multiple of U

When the number of neighbor nodes k is equal to the normalized upload capacity U, the source node can deliver each chunk to *all* its k neighbors before a new chunk arrives. As such, the source node can repeatedly apply a round-robin scheduling policy during the time interval $T = UT^*$, which elapses between the arrivals of consecutive chunks. Specifically, in the first T^* seconds it can send a given chunk to a given node, say peer N_1, then send the chunk to peer N_2, and so on until peer N_k. If this policy is repeated for every chunk, the result is that any neighbor of the source also receives a new chunk every $T = UT^*$ seconds. Hence, each neighbor of the source may apply the same scheduling policy with respect to its neighbors, and so on. As a consequence, every node in the network receives chunks from the same parent, and in the original order of generation: in other words, chunks are delivered over a tree topology.

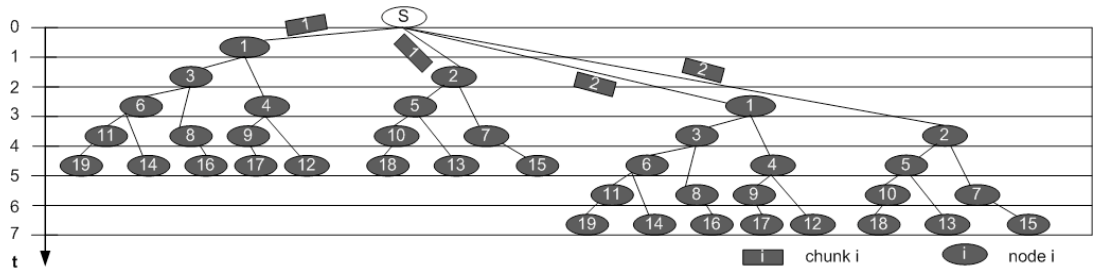

Figure 2.5. Overlay tree resulting in the case $k = U = 2$.

The operation of the above described chunk distribution mechanism is depicted in figure 2.5, which refers to the case $U = k = 2$ and a network composed of 19 nodes. In this figure the source is denoted with an "S". The nodes and the chunks

are progressively indexed starting from 1. Going from the upper part of the figure to its lower part, we see how the first two chunks are progressively distributed starting from the source; the time since the start of the transmission, measured in time units, until time instant $t = 7$ is reported on the left side of the figure. The tree on the left hand side of the figure distributes the first chunk, while the tree on the right hand side of the figure distributes the second chunk. In more detail, since the first chunk is assumed to be available for transmission at the source at time instant $t = 0$, the source starts transmitting the first chunk to node 1 at $t = 0$ and after finishing this transmission, i.e at $t = 1$, it sends the first chunk to node 2, in series. In its turn, node 1 sends the first chunk first to node 3 and then to node 4, in series, and so on. Likewise, node 2 sends the first chunk first to node 5 and then to node 7, in series, and so on. As regards the second chunk, the source starts transmitting it to node 1 at time $t = 2$, exactly when that chunk is available for the transmission. After finishing transmitting the first chunk to node 1, the source sends the same chunk to node 2, in series. In their turn, node 1 and 2 distribute the second chunk in same manner as the first chunk, i.e. sending the second chunk in series first to nodes 3 and 5 respectively, and then to nodes 4 and 7 respectively.

It is to be noted that, even if two distribution trees are depicted in figure 2.5, actually there is only one distribution, which repeats itself for each chunk with period $k = U = 2$. In other words, a given node receives all chunks through the same path. It is also interesting to note that the tree formed in figure 2.5 is unbalanced in terms of number of hops. For instance, the first chunk reaches node 19 at time $t = 5$ after crossing nodes 1,3,6 and 11. Conversely, the same chunk reaches node 15, again at time $t = 5$, after crossing nodes 2 and 7. The unbalancing in terms of number of hops is a consequence of the fact that the proposed approach achieves equal-delay source-to-leaves paths, and that the time in which a chunk waits for its transmission turn at a node (because of serialization) contributes to such path delay.

We are now in condition to evaluate the stream diffusion metric $N(t)$. To this end, let us introduce $n(i)$ as number of new nodes that complete the download of a chunk exactly i time units after the generation of that chunk at the source node, in such a way that $N(t)$ can be assessed according to the equation $N(t) = \sum_{i=1}^{t} n(i)$. With reference to figure 2.5, $n(1) = 1$ (node 1), $n(2) = 2$ (nodes 2 and 3), $n(3) = 3$ (nodes 4, 5 and 6), $n(4) = 5$ (nodes 7, 8, 9, 10 and 11), $n(5) =$

8 (nodes 12, 13, 14, 15, 16 and 17). Thus, $N(t) = 19$, which is equal to the performance bound $\overline{N}(t)$ evaluated at $t = 5$. To generalize the evaluation of $n(i)$, we observe that only the nodes which have completed the download of a chunk exactly after $i-1$,$i-2$,$i-3$,$\cdots$,$i-k$ since the generation of that chunk have still children to be served, whereas nodes that have completed the download of that chunk with a delay less than $i-k$ have already served all their k children. As a consequence, if we set $n(0) = 1$ to take the children served by the source into account, it results $n(i) = n(i-1) + n(i-2) + \cdots + n(1) + n(0)$ for $i \leq U$ and $n(i) = n(i-1)+n(i-2)+\cdots+n(i-k+1)+n(i-k)$ for $i > U$. It is then easy to evaluate the sequence $n(i)$ for a given $k = U$ and to verify that $n(i) = F_k(i+1)$ and consequently $N(t) = \sum_{i=1}^{t} F_k(i+1)$. Easy algebraic manipulations allow to turn the last equality into $N(t) = \sum_{j=1}^{k} S_k(t-j+1)$, which guarantees the matching between the stream diffusion metric of the described chunk distribution mechanism and the performance bound $\overline{N}(t)$ for each value of t.

2.4.1 Case $k > U$ and multiple of U: unbalanced multiple trees

When $k > U$, the source cannot deliver a chunk to all its k neighbors, but only to a subset of U peers. Hence, in principle, it might distribute chunks through the same tree as discussed before, and hence every peer in the network would use only U neighbors out of the available k. However, the provided bound assures that performance in the case $k > U$ are better than in the case $k = U$. For instance, if $U = 2$, the case $k = 4$ outperforms the case $k = 2$ as follows:

t	1	2	3	4	5	6	7	...
$\overline{N}(t)$,$k=2$	1	3	6	11	19	32	53	...
$\overline{N}(t)$,$k=4$	1	3	6	12	24	47	91	...

2.4.2 Case $k = U$: unbalanced tree

A thorough general explanation of how to design a mechanism which attains the bound in the case $k > U$ and multiple of U is complex (for reasons that will emerge later on). Hence, we limit ourselves to show how the bound may be achieved through

the simple example depicted in figure 2.6, which refers to the case $U = 2$ and $k = 4$ and a network composed of 24 nodes.

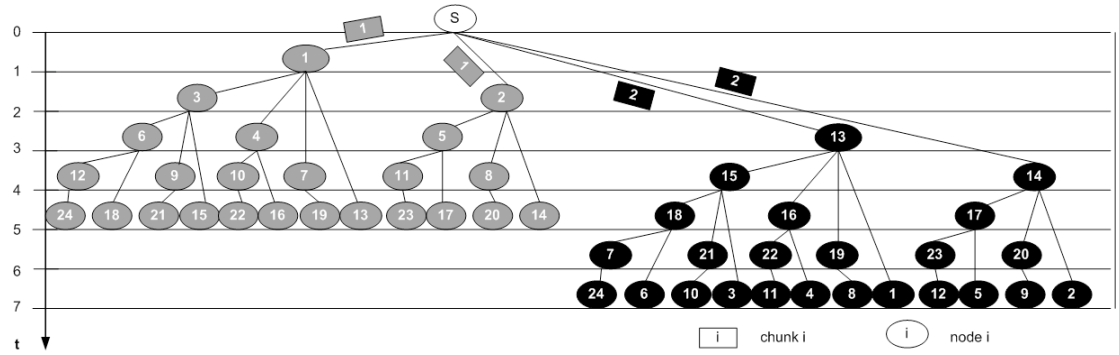

Figure 2.6. Chunk distribution over multiple trees for the case $U = 2$,$k = 4$.

The notation in this figure is the same as in figure 2.5. As in the case $k = U = 2$, at time $t = 0$ the source node receives chunk #1 and serially delivers it to nodes 1 and 2. However, with respect to the case $k = U = 2$, at time $t = 2$, when the source node receives chunk #2, instead of sending it again to nodes 1 and 2, it delivers that chunk to the remaining two neighbors (nodes 13 and 14). This process is repeated for the subsequent chunks, and specifically the odd-numbered chunks are serially delivered to nodes 1 and 2, while the even-numbered ones are serially delivered to nodes 13 and 14. As a consequence of this operation of the source, each neighbor of the source i) receives directly from the source only half chunks, ii) receives a new chunk from the source every 4 time units. As such, neighbors of the source have the necessary extra time to deliver each chunk they receive from the source to all their $k = 4$ neighbors. The same holds for the remaining peer nodes. For instance, with regard to chunk #1, node 1 serves that chunk to all its four neighbors (nodes 3, 4, 7 and 13) in series. Node 2 serves instead chunk #1 only to three neighbors (nodes 5, 8 and 14) out of four available, since all nodes in the network have already received chunk #1 at $t = 5$ and there are no nodes to be served. In their turn, all nodes that have been served by nodes 1 and 2, transmit chunk #1 to their neighbors (unless their neighbors have already received that chunk) in series, and so on, until all nodes in the network receive chunk #1. This allows delivering chunk #1 to 24 nodes in 5 time units, instead of the previous 19 nodes.

It is to be noted that chunks are now distributed by means of two distinct

unbalanced trees, the left one for odd-numbered chunks and the right one for even-numbered chunks, which repeat themselves with period $k = 4$. In general, the number of distribution trees is k/U, where we use the assumption that k is integer multiple of U.

We are now in condition to evaluate the stream diffusion metric $N(t)$. As in the case $k = U$, let us introduce $n(i)$ as number of new nodes that complete the download of a chunk exactly i time units after the generation of that chunk at the source node, in such a way that $N(t)$ can be assessed according to the equation $N(t) = \sum_{i=1}^{t} n(i)$. With reference to figure 2.5 and to the left hand side tree, $n(1) = 1$ (node 1), $n(2) = 2$ (nodes 2 and 3), $n(3) = 3$ (nodes 4, 5 and 6), $n(4) = 6$ (nodes 7, 8, 9, 10, 11 and 12), $n(5) = 12$ (nodes 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 and 24). The amounts $n(i)$ take on the same values even in the right hand side tree. Thus, $N(t) = 24$, which is equal to the performance bound $\overline{N}(t)$ evaluated at $t = 5$. To generalize the evaluation of $n(i)$, we observe that, if $i \leq U$, the source is still serving a given chunk; otherwise, the source is already serving the next chunk. In addition, only the nodes which have completed the download of a chunk exactly after $i-1$,$i-2$,$i-3, \cdots, i-k$ since the generation of that chunk have still children to be served, whereas nodes that have completed the download of that chunk with a delay less than $i-k$ have already served all their k children. As a consequence, if we set $n(0) = 1$ to take the children served by the source into account, it results $n(i) = n(i-1)+n(i-2)+\cdots+n(1)+n(0)$ for $i \leq U$, $n(i) = n(i-1)+n(i-2)+\cdots+n(2)+n(1)$ for $U < i \leq k$ and $n(i) = n(i-1) + n(i-2) + \cdots + n(i-k+1) + n(i-k)$ for $i > U$. It is then easy to evaluate the sequence $n(i)$ for a given pair of k and U values and to verify that $n(i) = F_k(i) + F_k(i-1) + \cdots + F_k(i-U+1)$ and consequently $N(t) = \sum_{i=1}^{t} \sum_{j=1}^{U} F_k(i-j+1)$. Easy algebraic manipulations allow to turn the last equality into $N(t) = \sum_{j=1}^{k} S_k(t-j+1)$, which guarantees the matching between the stream diffusion metric of the described chunk distribution mechanism and the performance bound $\overline{N}(t)$ for each value of t.

Before concluding the description of the case $k > U$ and multiple of U, we finally observe that a peer node needs to be part of all the k/U trees in order to properly receive the full stream. This leads to a complex issue which we call the "tree intertwining problem", that is: how nodes should be placed in every tree so that the different role of a node in every considered tree does not lead to sharing

the node's upload capacity among the different trees (and hence to performance impairments with respect to the bound's prediction, or even congestion). This can be more easily illustrated through the following example. Let us first consider node 5. In the left (odd-numbered) tree, node 5 is in charge of serving two neighbors, namely 11 and 17. If node 5 were used by the right (even-numbered) tree in place of node 15, it would also have to forward even-numbered chunks to three additional neighbors, thus breaking the assumption that a node has at most $k = 4$ neighbors. The problem is actually more complex, as we can understand by considering the following second case. In the odd-numbered tree, node 2 has to serve three nodes, namely nodes 5, 8, and 14. At a first glance, we might conclude that node 2 can be also used by the even-numbered tree provided that it is placed in a position of the tree that requires the node to serve only a single node. However, this is not the case. In fact, let us assume to replace node 7 in the even-numbered tree with node 2. This implies that node 2 would be required to deliver an even-numbered chunk to node 24 at every time instant $t = 6 + 4n$. However, node 2 is required by the left tree to deliver an odd-numbered chunk at instants of time $t = 2 + 4n$,$t = 3 + 4n$, and $t = 4 + 4n$. Thus, since $6 + 4n = 2 + 4(n + 1)$, node 2 should simultaneously deliver an odd-numbered chunk to node 5, and an even-numbered chunk to node 24, which would not allow reaching the bound.

Unfortunately, the "intertwining problem" for unbalanced trees can not be solved by letting interior nodes of a given tree play the role of leaves in the remaining trees[5]. However, we proved in 2.4.3 that i) the tree-intertwining problem can be solved via exhaustive search for arbitrary U and k and for any network size for which the bound $\overline{N}(t)$ is attainable, and that ii) there exists a constructive approach which allows finding one of the many possible solutions without relying on exhaustive search.

2.4.3 The "tree intertwining" problem

To understand the tree intertwining problem, we propose the following example. Let us consider our distribution algorithm in case of $k = 3$, $U = 1$ and a network of 28

[5]This is instead the solution when parallel transmission and, as consequence, balanced trees are used [65], being trivial to show that the number of leaves in a tree of fan-out k is greater than $(k - 1)$ times the number of non-leaf nodes.

nodes. In such a case, peer nodes are organized into 3 distribution trees, which are denoted as $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$. $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$ repeat themselves with period 3 in such a way that

- $T_{3,1}$ is used to transmit chunks 1,4,7,... and in general all chunks c with $c \bmod 3 = 1$;
- $T_{3,2}$ is used to transmit chunks 2,5,8,... and in general all chunks c with $c \bmod 3 = 2$;
- $T_{3,3}$ is used to transmit chunks 3,6,9,... and in general all chunks c with $c \bmod 3 = 0$.

Figure 2.7 shows how the distribution trees $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$ allow distributing the first 3 chunks in the time interval (0,8). This figure has the same structure and meaning as figure 2.6. Going from the left part of the figure to the right part, we see how the first three chunks are progressively distributed starting from the source till they reach all nodes. The difference with respect to figure 2.6 is that now $U = 1$,$k = 3$ and thus we have three distribution trees. In addition, peer nodes are differently indexed: peer nodes are indexed starting from 1 in each overlay tree and the subscript 1, 2 or 3 allows to distinguish between $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$. In other words, we do not use the same index to denote the same node in each overlay tree, but different indexes denote the same node in different distribution trees. As we will understand later on in this section, the reason for this change is that using the indexing of figure 2.6 would imply that we have already solved the problem.

From figure 2.7, we can observe that for each overlay tree $T_{3,i}$ with $i = 1,2,3$ it is possible to classify the 28 peer nodes as follows:

- 4 nodes, namely 1_i,2_i,3_i and 4_i, $i = 1,2,3$, transmit for 3 consecutive time units. We refer to these nodes as "class 3" nodes.
- 4 nodes, namely 5_i,6_i,7_i and 8_i, $i = 1,2,3$, transmit for 2 consecutive time units. We refer to these nodes as "class 2" nodes.
- 7 nodes, namely 9_i,10_i,...,15_i, $i = 1,2,3$, transmit for only 1 time unit. We refer to these nodes as "class 1" nodes.

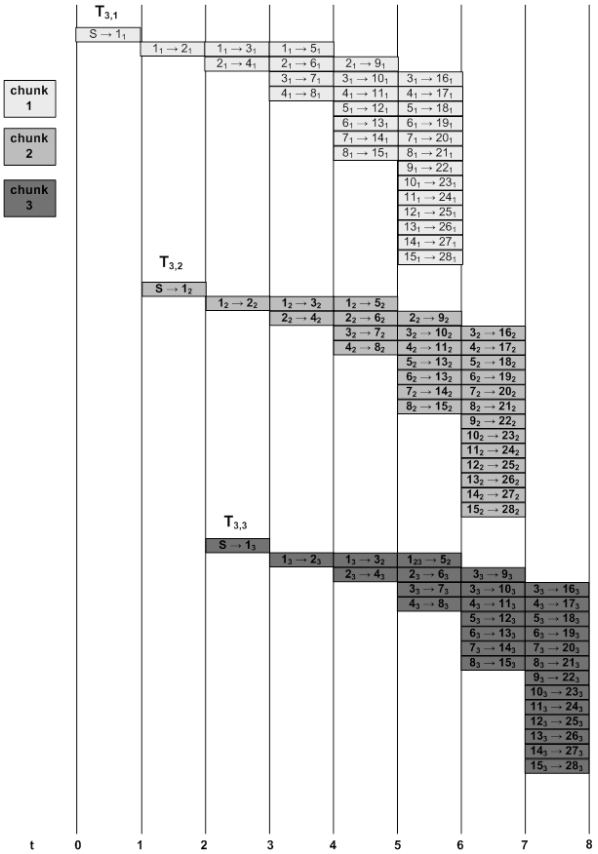

Figure 2.7. Overlay trees $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$ in a network of 28 peer nodes: distribution process of the first 3 chunks in the time interval (0,8)

- 13 nodes, namely 16_i,17_i,...,28_i, i = 1,2,3, do not transmit at all. We refer to these nodes as "class 0" nodes.

Thus, we define "nodes of class j" the set of nodes that transmit for j consecutive time units. In addition, as $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$ repeat themselves with period 3, we observe that, even if each distribution tree includes the whole set of 28 nodes, a node that belongs to a given class in a given distribution tree can belong to the same or to another class in another distribution tree. For instance "class 3" nodes in $T_{3,1}$ transmit for three consecutive time units and then start again, thus they do not have idle times to transmit also in $T_{3,2}$ and $T_{3,3}$. As a consequence, "class 3" nodes in $T_{3,1}$ have to be "class 0" nodes in $T_{3,1}$ and $T_{3,3}$.

The tree intertwining problem consists in assigning peer nodes to classes in each distribution tree in such a way that no conflict occurs in the uplink capacity. By

no conflict in the uplink capacity we mean that the node must be able to transmit at full rate to a single node. We will now present a solution of the tree intertwining problem for the case depicted in figure 2.7.

The solution of the tree intertwining problem is given by showing what is the role played by the nodes in each distribution tree. For instance, node indexed as 1 in tree $T_{3,1}$, which is a "class 3" node has to become a "class 0" node in $T_{3,2}$ and in $T_{3,3}$. Thus, to avoid conflicts, this node will have to become one of the node indexed from 16 to 28 in $T_{3,2}$ and in $T_{3,3}$, i.e. one of the nodes that do not have to relay the received chunk further on. The complete solution for our example is given in table 2.2. In this table each row describes the role played by each node in each tree. For instance in the first row we see that node 1 of class 3 in tree $T_{3,1}$ becomes node 16 of class 0 in tree $T_{3,2}$ and then becomes node 16 of class 0 in tree $T_{3,3}$. Another example is that of node 5 of class 2 in tree $T_{3,1}$ that becomes node 9 of class 1 in tree $T_{3,2}$ and then becomes node 20 of class 0 in tree $T_{3,3}$. In fact, given the periodicity of 3 time units of the distribution trees, node 5 in $T_{3,1}$ transmits for two time units and before transmitting again in the same tree it has to wait for a time unit. Thus, it can play the role of a "class 1" node in tree $T_{3,2}$ and of a "class 0" node in tree $T_{3,1}$. Of course several solutions are possible, and table 2.2 presents only one of them.

How to generalize the solution of the intertwining problem for each (U,k) is not a simple task. In the next two subsections we provide a "constructive" demonstration: given a k value, we will describes how to assign nodes to classes in such a way that the k distribution trees may be intertwined without conflicts in the uplink capacity. The demonstration is limited to the case of $U = 1$, to which the more general case $U > 1$ may be traced. Before the constructive demonstration that will be presented in subsection 2.4.3.2 we must first formalize the concept of "node class" in subsection 2.4.3.1.

2.4.3.1 Node classes

As introduced before, with respect to a given distribution tree, each node can be classified as belonging to a class $0,1,\cdots,k$, being k the number of parents/children. Specifically, a node in class k uses all its available uplink capacity for the transmission to its k children in the considered tree, while a node in class i, $0 \leq i < k$,

$T_{3,1}$		$T_{3,2}$		$T_{3,3}$	
Node index	Node class	Node index	Node class	Node index	Node class
1_1	3	16_2	0	16_3	0
2_1	3	17_2	0	17_3	0
3_1	3	18_2	0	18_3	0
4_1	3	19_2	0	19_3	0
5_1	2	9_2	1	20_3	0
6_1	2	10_2	1	21_3	0
7_1	2	11_2	1	22_3	0
8_1	2	12_2	1	23_3	0
9_1	1	20_2	0	5_3	2
10_1	1	21_2	0	6_3	2
11_1	1	22_2	0	7_3	2
12_1	1	23_2	0	8_3	2
13_1	1	13_2	1	13_3	1
14_1	1	14_2	1	14_3	1
15_1	1	15_2	1	15_3	1
16_1	0	1_2	3	24_3	0
17_1	0	2_2	3	25_3	0
18_1	0	3_2	3	26_3	0
19_1	0	4_2	3	27_3	0
20_1	0	5_2	2	9_3	1
21_1	0	6_2	2	10_3	1
22_1	0	7_2	2	11_3	1
23_1	0	8_2	2	12_3	1
24_1	0	24_2	0	1_3	3
25_1	0	25_2	0	2_3	3
26_1	0	26_2	0	3_3	3
27_1	0	27_2	0	4_3	3
28_1	0	28_2	0	28_3	0

Table 2.2. Role played by the nodes in each distribution tree: solution of the tree intertwining problem in a network of 28 nodes.

uses only partially its uplink capacity in the considered tree by exploiting only i consecutive transmission "slots". Thus, nodes that are in class i, $0 \leq i < k$ with respect to a distribution tree, may transmit for other distribution trees in the unused transmission slots.

To formalize this concept, we introduce the notion of "class pattern". For a generic node n, let us define class pattern as the set

$$P(n) = \{x_{n,1}, x_{n,2}, \cdots, x_{n,k}\}$$

where $x_{n,i} \in \{0,1,\cdots,k\}$ is the class of the considered node n with respect to the tree i, $i \in \{1,2,\cdots,k\}$. Obviously, since each distribution tree repeats itself with period k and, as a consequence, the maximum number of transmission is bounded to k, for every node the following necessary condition must hold:

$$\sum_{i=1}^{k} x_{n,i} \leq k$$

However, this condition is not sufficient to guarantee that no conflict in the uplink capacity occurs during node transmission. In fact, a further condition is that the classes should be chosen so that no temporal overlap in the transmission slots occur. In what follows, we say that a node is "packed" to indicate that such a condition is verified for the class pattern of that node. For instance, if $k = 3$ (like in the example presented in Section 2.4.3), the class pattern $\{1,2,0\}$ satisfies the necessary condition, but it would not be valid, as the transmission for 1 slot in the distribution tree 1 would partially overlap with the transmission for 2 slots in the distribution tree 2.

We now evaluate the number of nodes in a given class. This is a function of the time interval T for which a chunk is transmitted across the network starting from the time instant at which it arrives at the source. First of all, we note that $N(c,k,1,c-1+T) = S_k(T)$ for every chunk c. Secondly, based on the theory developed in section 4.3.4, it is straightforward to observe that, among the total number $S_k(T)$ of nodes, the number of nodes $N_x(T)$ in class x is given by:

$$N_x(T) = \begin{cases} F_k(T-x) & 0 \leq x \leq k-1 \\ S_k(T-k) = \sum_{p=1}^{T-k} F_k(p) & x = k \end{cases}$$

The above condition has an important implication: for every pair of classes s and r with $r > s$, then $N_r(T) < N_s(T)$. This somehow suggests that a class s may always coexists with a class $r < s$. As a matter of fact, a node in class r in the tree i with $k-1 \geq r \geq \lfloor k/2 \rfloor + 1$ can be assigned to class $k-r$ in the tree

$(i+k-r)$ mod k. This is always possible under the assumption $k-1 \geq r \geq \lfloor k/2 \rfloor + 1$, since it is $N_r(T) < N_{k-r}(T)$. In other words, it is possible that all nodes in class $k-1, k-2, \cdots, \lfloor k/2 \rfloor + 1$ can be successfully packed (a class k node is self-packed by definition). However, this approach cannot be generalized when $1 \leq r < \lfloor k/2 \rfloor + 1$. For instance, if $k = 4$, a node in class 2 for tree 1 may be assigned to class 1 in trees 2 and 3; likewise, a node in class 2 for tree 2 may be assigned to class 1 in trees 3 and 4, and so on. In such a condition, $2 \times F_4(T-2) > F_4(T-1)$ class 1 nodes would be present in each tree, and this is not admissible.

2.4.3.2 Constructive demonstration

In this subsection we solve the tree intertwining problem by presenting a "constructive" procedure that can be applied for every value of k and it allows to intertwine the distribution trees without conflicts in the uplink capacity (or equivalently without overlapping between transmission slots).

We first recall a fundamental property of the k-step Fibonacci numbers that immediately follows from the definition (2.1). For any value $x \in \{0, 1, \cdots, k-2\}$ it is

$$F_k(T-x) > \sum_{i=x+1}^{k-1} F_k(T-i)$$

From such a property we draw the following important conclusion: *given $x \in \{0, 1, \cdots, k-2\}$, if each node in all classes y with y greater than x is assigned to class x in at most one tree, the constraint on the cardinality of the set of nodes in class x, namely $N_x(T)$, is satisfied.* In what follows, we refer to this condition as "feasibility condition".

Based on this result, we simply need to find a procedure which guarantees that the feasibility condition is satisfied. In particular, we propose a procedure that takes its inputs from the results of an apparently unrelated arithmetic procedure. For a class $1 < x < \lceil k/2 \rceil$ such an arithmetic procedure consists in the following steps:

1. set $r_0 = 0$ and $r_1 = x$, and compute $k = q_1 \times r_1 + r_2$;
2. start with the index $i = 2$;
3. let q_i be the unique solution of the equation $k - r_{i-1} = q_i r_i + r_{i+1}$;

4. compute $r_{i+2} = (k - r_i) \bmod r_{i+1}$;

5. if $r_{i+2} = 0$, then terminate; else increment i and return to step 3.

The output of such arithmetic procedure is the set of values r_i and q_i.

The rationale beyond this arithmetic procedure, and its relation with the underlying intertwining procedure and its feasibility proof, is best explained through a numerical example.

Let us consider the case $k = 11$, and start from $x = 4$. For the sake of simplicity, let us assume that there is only one class 4 node, one class 3 node and one class 1 node per tree (as proven before, we can decouple the demonstration from the class cardinality, and we only need to show that for every class 4 node we need at most one node in each inferior value class). At step 1 we set $r_1 = 4$ and we compute $r_2 = 11 \bmod r_1 = 3$ and $q_1 = 2$. In terms of our problem, this translates into saying that a class 4 node will be assigned to another class 4, of course in a different tree (this is the meaning of the term $q_1 = 2$), and to a class 3 entry (i.e. the rest of the integer division between 11 and 4). Now, in terms of trees, this implies that class 4 is present in 8 over 11 trees, while class 3 is present in 4 *different* trees. Moreover, 7 trees have not used their class 3 node. This is accounted for in the follow up of the procedure. Specifically, step 3 computes the division between $(11 - r_1) = 7$ and $r_2 = 3$, and it provides the values $q_2 = 2$ and $r_3 = 1$ as output (in fact $7 = 2 \times 3 + 1$). At step 4, the procedure computes $r_4 = 7 \bmod 3 = 1$. This translates in stating that each of the three remaining class 4 nodes will be combined with a class 3 node transmitting in $q_2 = 2$ trees (it is straightforward to observe - although formally cumbersome to prove in the general case and hence the proof is omitted - that these class 3 nodes are taken from the set of the seven remaining trees). Then step 3 is repeated and $q_3 = 2$ and $r_4 = 1$ are computed. This means that a further class 1 entry will complete the packing. To summarize, class 4 node in tree 9 is assigned a class 3 node in tree 2 and tree 5, and a class 1 node in tree 8. The same is repeated for the class 4 node of trees 10 and 11. At this stage i) class 4 is present in all trees; ii) class 3 is not present in tree 8, and iii) eight class 1 nodes are left. Since class 1 nodes do not overlap, in terms of transmission slot usage, with the remaining class 3 node in tree 8, the packing is completed.

The above approach can be repeated for every possible value of x and k, and it always yields a single solution which satisfies the above stated feasibility condition. The construction of one solution (among the many possible ones) for each pair x,k guarantees that the tree intertwining is always solvable.

2.5 Performance Evaluation

Figure 2.8 plots the stream diffusion metric $N(t)$ as a function of T^* in a $U = 2$ bandwidth scenario, for a single unbalanced tree ($k = 2$), two unbalanced trees ($k = 4$), infinite unbalanced tree ($k = \infty$) and a single *balanced* tree ($k = 2$ and parallel transmissions).

The first important observation about figure 2.8 regards the impact of the number of neighbor nodes k on the stream diffusion metric bound. The figure shows that there is a significant improvement when moving from the case $k = U = 2$ of single tree to that of multiple trees. Interestingly (but expected, as the Fibonacci constants ϕ_k increase only marginally when k becomes large), the advantage in using more than a few trees is limited: this is especially important if an algorithm is designed to mimic the unbalanced multiple tree operation, as complexity (i.e. signalling burden) increases with k.

The second important observation regards the improvement brought about by serializing the transmissions (and hence unbalanced trees) with respect to parallel chunk transmissions (and hence balanced trees). The figure shows that the performance improvement is significant: in the case $k = 2$ the stream diffusion metric $N(t)$ for serial chunk transmissions (i.e., the bound) is one order of magnitude greater than for parallel chunk transmissions at $t = 20$, and three orders of magnitude at $t = 50$.

2.6 Comparison with a literature proposal

We compare our solution (named "Streamline") to *SplitStream* [18], since it is maybe the most representative example of algorithms exploiting forest-based topologies, like our *Streamline*. In *SplitStream* the stream is stripped into *stripes* to be distributed

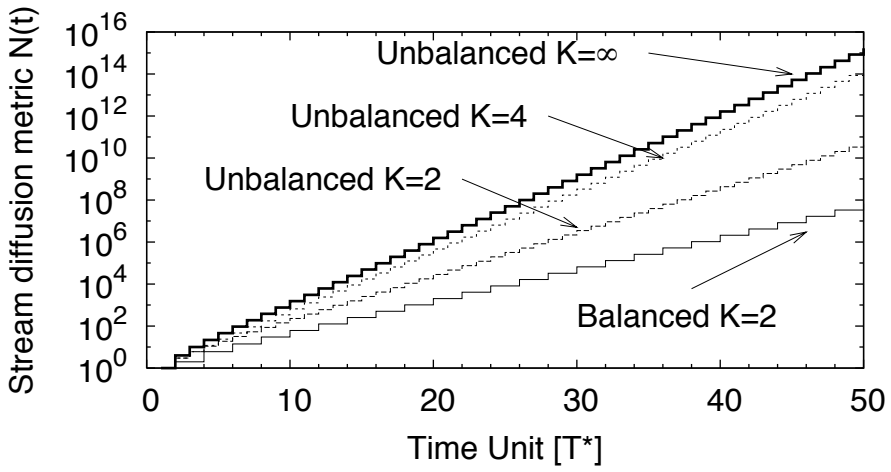

Figure 2.8. Stream diffusion metric $N(t)$ as a function of T^* in a $U = 2$ bandwidth scenario, for $k = 2$ (balanced and unbalanced tree), $k = 4$ (two unbalanced trees), and $k = \infty$ (infinite unbalanced trees).

across a forest of overlay multicast trees. The construction of the overlay multicast trees is based on Scribe [24]. Scribe is an application-level group communication system built on top of Pastry [25], which is a DHT-based self-organizing, structured, P2P overlay network. The key idea is the following: each multicast group is associated with a pseudo-random Pastry key, and the corresponding multicast overlay tree is formed by the union of the Pastry routes from each group member to the root node[6] for that group Pastry key. Messages are multicast from the root to the members using reverse path forwarding. With reference to figure 2.9 , source splits the content in its designated tree. Each stripe's stripeID starts with different digit. The interior node nodeIds share a prefix with the stripeId, so that they are interior node of one tree and leaves in the other trees. For instance node M has nodeId that starts with 1 so it is an interior node for the stripe whose id starts with 1 and leaf for the other ones.

In more detail, *SplitStream* uses a separate overlay multicast tree for each stripe. A fundamental property of the *SplitStream* overlay trees is that they are interior-node-disjoint trees. In other words, *SplitStream* nodes are organized into overlay trees in such a way that each node is interior node in at most one tree and leaf node

[6]We recall that the root node for a Pastry key is the node with the identifier that is numerically closest to the key.

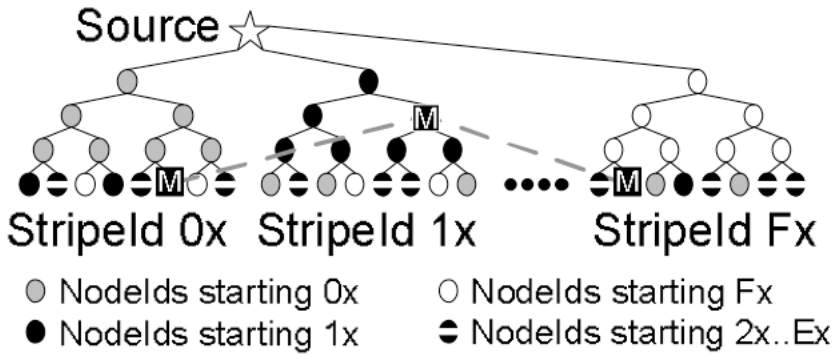

Figure 2.9. SplitStream forest contruction

in the remaining trees. To achieve this goal, *SplitStream* i) chooses stripe/group identifiers that all differ in the most significant digit, and ii) it takes advantage from the Pastry property of forwarding messages towards nodes whose identifiers share progressively longer prefixes with the messages key.

To compare *SplitStream* and *Streamline*, unfortunately we could not perform an analytical comparison, since *SplitStream* is a working practical solution but lacks an analytical model for *SplitStream*. Thus, we resorted to simulation techniques and implemented *SplitStream* by using a suitably developed simulation package, based on our OPSS [66].

In order to perform a fair comparison, we considered a number of stripes and, consequently, a number of overlay multicast trees at least equal to the number of *Streamline* distribution trees. Since the analytical model that we presented for Streamline assumes homogeneous networks in terms of uplink capacity, we also simulated *SplitStream* in a homogeneous uplink capacity scenario, and we always compared *SplitStream* and *Streamline* under the same value of the ratio U between the uplink capacity of peer nodes and the stream bitrate. As regards the downlink capacity, we used for *SplitStream* a value great enough so that downlinks are not a bottleneck of the system, as done for *Streamline*. In addition, we simulated *SplitStream* in absence of churn. Finally, we observe that *SplitStream* segments the stream into sub-streams organized as small IP packets and sequentially delivered across separate trees, whereas Streamline divides the stream into chunks of size larger than the typical IP packet size, delivered in a store-and-forward fashion

across separate trees. Nevertheless, in order to make the comparison between *SplitStream* and *Streamline* possible, we let *SplitStream* operate by distributing chunks. In other words, we simulated a P2P overlay network where i) peer nodes are organized according to *SplitStream* multicast trees, ii) the stream is segmented into chunks, iii) chunks are grouped into different sub-sets of chunks distributed across separate SplitStream multicast trees.

We evaluated the performance of *SplitStream* by considering the performance index $\overline{N}_{served}(c,d)$, that is the average number of nodes that complete the download of a chunk c with a chunk delivery delay less than or equal to d; the chunk delivery delay is the difference between the time instant at which the chunk arrives at the source and the time instant at which the chunk is completely received from a node.

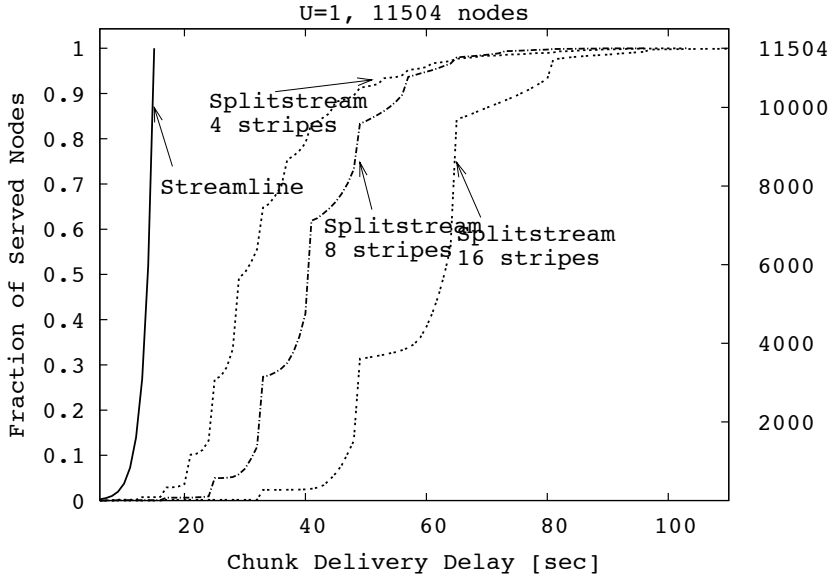

Figure 2.10. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *Streamline* and *SplitStream* with $N = 11504$ and $U = 1$

We first consider a scenario with 11504 nodes and a ratio between uplink capacity and stream bit rate equal to 1.

In figure 2.10 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$ by comparing *Streamline* to *SplitStream*; in the case of *SplitStream* we consider three different values of the number of stripes. We consider a network with 11504 nodes, ratio between uplink capacity and stream bit rate equal to 1, for three different

values of the number of stripes.

We observe that the outbound degree of each node, that is the number of children which each node may forwards stripes to, has been set to the number of stripes. The reason is that, since the ratio between uplink capacity and stream bit rate is equal to 1: i) the source may serve each chunk just to one child; ii) each node may serve a given chunk exactly to s children (s being the number of stripes), before the time instant it will receive the next chunk and it will have to serve it to the same children. The curve relative to *Streamline* has been analytically derived by setting $k = 4$. In fact, even if the performance of *Streamline* improve monotonically as a function of the number k of children, we chose $k = 4$ since, as we previously demonstrated, the improvement becomes negligible for greater values of k. We observe that *Streamline* outperforms *SplitStream* regardless of the number of stripes. In addition, as the number of stripes increases, the performance of *SplitStream* get worse. For instance, while *Streamline* serves all 11504 nodes within 15 seconds, with 4 groups, *SplitStream* serves on average 87 nodes in case of 4 stripes, 9 nodes in case of 8 stripes and 1 node in case of 16 stripes.

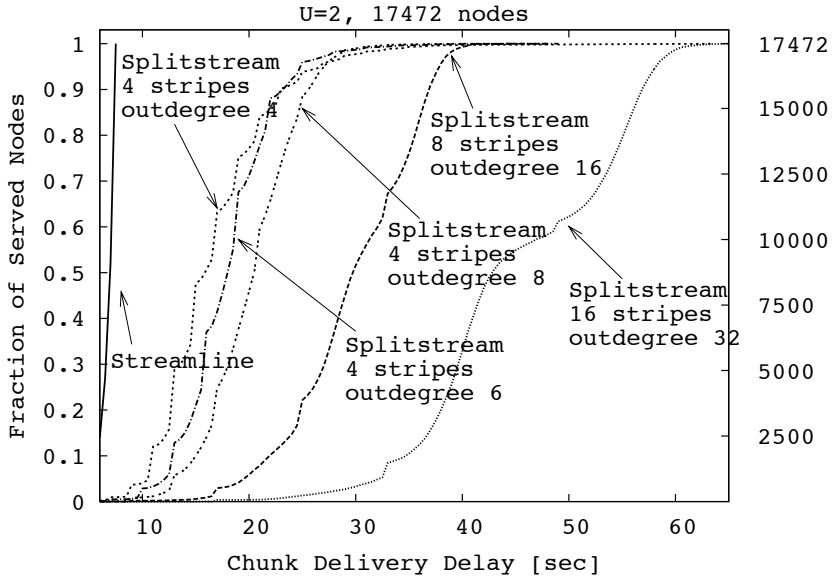

Figure 2.11. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *Streamline* and *SplitStream* with $N = 17472$ and $U = 2$.

Now we consider a second scenario with 17472 nodes and a ratio between uplink

capacity and stream bit rate equal to 2.

In figure 2.11 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$ by comparing *Streamline* to *SplitStream*; in the case of *SplitStream* we consider different values of the number of stripes and of the outbound degree. The curve relative to *Streamline* has been analytically derived by setting $k = 4$. As in the previous case, *Streamline* outperforms *SplitStream* regardless of the number of stripes and of the outbound degree. In addition, we can observe that, given a number of stripes, as the outbound degree increases, the performance of *SplitStream* worsen.

Chapter 3

A Theory-Driven Distribution Algorithm for Peer-to-Peer Real Time Streaming

3.1 Introduction

In this chapter a simple data-driven heuristic, called *O-Streamline* will be presented. This algorithm exploits the idea of using serial transmissions over multiple paths and relies on a pure data-oriented operation (i.e. chunk paths are not pre-established) to achieve performances close to the ones of the theoretical bound presented in chapter 2 but works in a distributed way. Therefore O-Streamline can be applied to real-world applications when the streaming flow is divided in chunks and the delay is an important performance of the system. The work presented in this chapter is realized in team work with professors Giuseppe Bianchi, Nicola Blefari Melazzi, Stefano Salsano and colleagues Francesca Lo Piccolo and Dario Luzzi.

3.2 O-Streamline: a distributed scheduling algorithm that approximate the delay bounds

In this section, we describe our proposal to turn the optimal scheduling algorithm previously found and named *Streamline* into *O-Streamline*, a practical and working distribution algorithm for P2P real time streaming.

O-Streamline tries to maintain the two basic principles of *Streamline*, that are i) the organization of peers and chunks in a finite number of overlay trees and groups and ii) the serialization of chunk transmissions.

While the second principle is easy to be put in practice, the first principle involves two distinct assumptions: i) the availability of a global and centralized vision of the whole network; ii) the absence of peer churn.

As regards the first assumption, *O-Streamline* does not count anymore on a global and centralized vision of the whole network, but it builds the overlay topology in a distributed manner and schedules chunk transmissions with a limited, local view of the overall topology.

O-Streamline constructs the overlay topology as follows. Peer nodes are uniformly divided into G different groups, in such a way that i) each node belongs only to one group, ii) each node establishes random overlay bidirectional connections with P peer nodes in the same group to which it belongs and O peer nodes in each of the remaining groups. This can be easily achieved in a real distributed environment. Provided that groups are progressively identified starting from 0, each node may establish its membership group if it extracts an (integer) random number with uniform distribution in the interval $[0,G-1]$.

The G groups are also used to organize the stream chunks. In more detail, if chunks are progressively indexed starting from 1, chunk i is associated with the group g such that $g = i \bmod G$.

To describe how *O-Streamline* schedules chunk transmission we let:

- L the chunk size (in bits)
- U the ratio between the stream bit rate R and the uplink capacity B of all nodes;

- $T = L/R$ the chunk duration (in s).

The scheduling algorithm operates as follows:

- the source node, which does not belong to any of the groups and it is connected to U neighbors per each group, sends the generic chunk in series to the U neighbors of the corresponding group. The source node serves the chunks to the neighbors of each group following the order implicitly established in the association between chunk identifiers and groups. The source node repeat this distribution pattern, modulo G, unless churn changes the overlay neighbors. Therefore each node which is child of the source receives a new chunk from the source every $G \times U \times \frac{L}{B}$ seconds;

- the generic peer relays only the chunks of the group it belongs to. A FIFO queue is used to manage the chunks of the group to which the generic peer belongs;

- the generic peer is interested only in chunks that arrive at the source, starting from the time instant at which the peer joins the system.

  This could be achieved by quering the source node at bootstrap time, asking for the identifier of the latest produced chunk. This implies that peer nodes must be somehow synchronized with the timing reference of the stream. However, this problem can be solved if the bootstrap node is the source, and sends to the new peer nodes joining the system the identifier of the next chunk that is going to be transmitted by the source; the case in which the bootstrap node is not the source is left for future work;

- the generic peer serves each chunk in the FIFO queue in series to $G \times U$ neighbors (if there are $G \times U$ neighbors missing that chunk), by giving priority to the neighbors of its own group. This implies that the number of neighbors P has to be at least equal to $G \times U$. Moreover, whenever possible, the generic peer serves the chunks belonging to the same group to which the peer belongs in the same order;

- if a peer ends serving a chunk to $G \times U$ neighbors and there are no new chunks

to be served in the FIFO queue, it tries to serve that chunk to other neighbors, until a new chunk to be served is enqueued in the FIFO queue.

Finally, it can be shown that the value of P must be greater than $G \times U$, for the algorith to work.

A last remark is that, being O-Streamline a data driven algorithm, it is not necessary to re-configure the whole overlay topology in case of peer churn.

In the next chapter, and in particular in section 4.4, performance evaluation of O-Streamline will be presented.

Chapter 4

Performance analysis of Peer-to-Peer Streaming Systems

4.1 Introduction

This chapter dealt with P2P streaming systems performance analysis. In particular it starts with the state of art of performance evaluation in those kind of systems (section 4.2), then it proposes a new simulator called "OPSS" for the analysis of large-scale p2p networks composed by hundred thousands nodes (section 4.3). This chapter ends with section 4.4 where there is a performance evaluation of O-Streamline, the algorithm presented in chapter 3. Algorithm performances are assessed using the "OPSS" simulator and compared with the theoretical bound found in chapter 2.

The work presented in this chapter is realized in team work with professors Giuseppe Bianchi, Nicola Blefari Melazzi, Stefano Salsano and colleagues Francesca Lo Piccolo and Dario Luzzi.

4.2 State of art of performance evaluation systems and metrics

Performance evaluation of scheduling algorithms for peer-to-peer mesh-based live streaming systems is still in its infancy. Despite plenty of literature has recently appeared in this area, most of the work focuses on the performance characterization of full-fledged systems involving possibly complex algorithmic solutions (such as joint push-pull chunk scheduling approaches), optimizations (such as large bandwidth assumed available at the stream origin), and loosely constrained network bandwidth conditions (typically assuming the stream rate to be a relatively small fraction of the available links bandwidth). The performance evaluation itself is frequently limited to small scale networks and often does not explicitly include chunk delivery delay as performance figure. Moreover often in literature there are not reference performance metrics so that a fair comparison between different choice is difficult to assess. Before reviewing the state of art of performance metrics and evaluation, its is possible to introduce the topic by identifying three main different approaches of performance evaluation: i) measurement-based studies or real systems ii) experimental testbed, such as PlanetLab, and iii) simulation tools. Measurement-based studies do not allow to consider different alternatives and to evaluate performance in advance of building and deploying a system. Experimental testbeds and current simulation tools suffer from scalability problems for different reasons. On the one hand, experimental testbeds would require a large network of emulator nodes, which is not easy to realize and to manage. On the other hand, the current simulation tools either are mostly oriented to the search phase and neglect the content distribution phase or perform the simulation at packet level, making unpractical to simulate a P2P live streaming system over a network of the order of 100K peers. Typical available results concern network size of the order of hundreds or few thousands peers. This is not representative of real-life P2P video streaming systems, which aim at streaming live multimedia content to a very large number, several hundred thousands if not millions, of users. It may be the case that network dynamics simulated in small-scale networks are not representative of large-scale P2P system deployments. Beside these difficulties, there is also an hard to reproduce component due to users behaviours

that can lead to phenomena like the flash crowds or churn, or that can heavy impact the whole system with their decision (e.g. when all the users acts as "free riders" that is when they acts in a selfish way, just taking without giving anything to other peers). To preserve the system against free riders, lot of solutions have been explored in literature. Basically the greatest part of them are based on game theory and/or try to bring in the peer to peer streaming systems, the solution that has been well tested and explored in the file-sharing applications such as the BitTorrent "tit-for-tat" strategy. Unfortunately the streaming scenario presents several differences from the file-sharing one, for instance not always reciprocity can be enforced as for the case of a simple single application layer distribution tree.

CoolStreaming/DONet is a Data-driven Overlay Network for live media streaming presented in section 1.3.1. The performance of DONet has been evaluated in [19] using PlanetLab [28][29]. PlanetLab is a global overlay network to support the design and the performance evaluation of applications widely distributed over the Internet. The control overhead and the continuity index are considered as performance metrics. The former represents the ratio between the control traffic volume and the video traffic volume; the latter is the number of segments that arrive before or on playback deadlines over the total number of segments. DONet performance is also compared with the performance of a tree-based overlay streaming system. Besides the continuity index, the average hop count is considered as raw approximation of end-to-end delay for delivering each segment. The number of used PlanetLab nodes ranges from 10 to 200 (passing through 50,100,150).

GridMedia is a unstructured P2P live media streaming presented in 1.3.2. Also GridMedia has been tested using the PlanetLab testbed. Pull and push-pull approaches are compared. The proposed experimental results relate to a number of PlanetLab nodes ranging from 300 to 340. Among the proposed performance indexes, we mention i) the absolute delay, that is the delay between the sampling time at the server and the playback time at the local node; ii) the delivery ratio, that is the ratio between the number of stream packets arriving before or right on absolute playback deadline and the total number of packets; iii) the α-playback-time, that is the minimum absolute delay at which the delivery ratio is larger than α ($0 \leq \alpha \leq 1$); iv) the control overhead of the gossip protocol, that is the average ratio between the control traffic and the total traffic at each node.

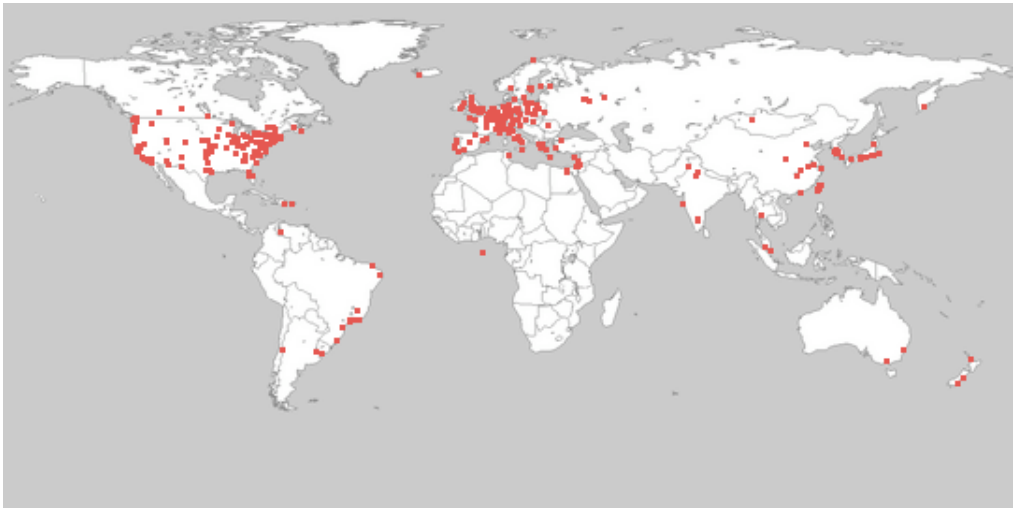

Figure 4.1. Maps of the 494 sites where are placed the 1068 PlanetLab nodes.

The same authors as [20] focus in [22] on the optimal streaming scheduling problem in data-driven overlay networks. The optimal streaming scheduling problem aims at addressing how each node optimally decides from which neighbor to request which block, and how it allocates its limited outbound bandwidth to every neighbor, in order to maximize the throughput.

To validate their algorithm, they use a discrete event-driven P2P simulator to simulate a data driven overlay network of 500 nodes. However, the authors do not give details about the kind of simulator they use. The low number of simulated nodes makes the hypothesis of packet-level simulator reasonable. The considered metric is the average delivery ratio, that is the ratio between the number of packets arriving before or right on the playback deadline averaged on all the nodes and the total number of packets. The proposed solution is compared with DONet, ChainSaw [30] and round-robin streaming scheduling.

NICE [15] is another overlay P2P live streaming system, but it is built on a hierarchically connected overlay topology differently from the previously described systems. The host hierarchy is used to define different overlay structures for control messages and data delivery paths. End-to-end latency is used as distance metric between host and it drives the association of nodes in clusters. NICE clusters and layers are created, maintained and eventually repaired by a fully distributed

protocol. Data overlay delivery path is instead the tree rooted at data source and implicitly defined by the control overlay topology hierarchy. A packet level simulator is used to evaluate NICE performance. Network topologies are generated using the Transit-Stub graph model and the GT-ITM topology generator [31]. The number of end hosts in the multicast group is ranges from 8 to 2048. Performance metrics such as the average link stress and the average path length are investigated. The first one is the number of identical packets sent over each underlying network link averaged across all the network links. The second one is the length (in number of hops) of the path from the source to the hosts averaged across all the hosts. The fraction of members that correctly receive the data packets in case of node failures and the byte-overhead for control traffic at the access links of the end-hosts are evaluated too. The achieved results are compared with the results obtained by simulating the application-layer multicast protocol Narada [32].

There are other P2P streaming applications like PPLive [26] or Sopcast [33] that are widely deployed but whose algorithms are not under public domain. The only solution for investigating their performance and behavior is to use a black-box measurement-based approach, as in [27] and in [34].

Regardless of the P2P live streaming systems described so far, a large number of P2P simulators have recently emerged. Most of them mainly focus on simulating the resource search phase and the related query message handling. This is the case of Aurora [35] and Serapis [36], which model the key announcement, insert or request process of Freenet-like systems. Similarly, P2Psim [37], FreePastry [38] and the Chord simulator [39], simulate only the DHT-based search phase. A similar approach is employed in other general-purpose P2P simulators, such as Neurogrid [40][41], 3LS [42], and Peersim [43][44]. Although the P2P query/search phases are undoubtedly representative of a P2P system, there is plenty of interest in quantitatively characterizing performance figures related to the resource distribution process among involved peers. All the previously mentioned simulation platforms are not suitable to this purpose, as they neglect the process of distributing data across peers. In fact, with regard to the low level network dynamics, either their effect is totally neglected, as in [35] and [40], where the overlay message transmission is immediate, or an exponentially distributed packet delay is used, as in [39], or the concept of distance between any two nodes is somehow defined and the overlay message

transmission time is identified with the latency between the relative nodes, as in [36][37][38][42][43].

To properly model the data distribution phase, GnutellaSim [45][46] interfaces with the ns-2 [47] discrete event packet-based network simulator, which provides a very detailed packet-level simulation model of the underlying transport network. However, such a simulation model compromises the scalability of the resulting simulation, as only a few hundreds nodes may be properly simulated in reasonable time with such a level of details.

4.3 OPSS: an Overlay Peer-to-peer Streaming Simulator for large-scale networks

On basis of the above observation, together with other colleagues, we propose OPSS [48] [49] , a new simulative approach that makes P2P video streaming performance evaluation scalable. Currently OPSS has been used by several universities all over the world. OPSS code is free for download from the reference website `http://opss.sourceforge.net`

4.3.1 How does OPSS achieve scalability?

In order to circumvent the tight scalability limits imposed by packet-based simulators and simultaneously to model networks dynamic with acceptable accuracy level, OPSS was conceived as discrete-event fluid-flow simulator. This allows to simulate the data distribution at the flow level, i.e. neglecting transmissions of single packets and focusing on events, such as start/end of a file or a file chunk transmission, which lead to a variation in the rate of the connections among peers. This approach dramatically reduces the number of simulation events and the related memory and computational load with respect to packet-level simulation, while retaining a satisfactory accuracy in the model of the data delivery process. We also assume that all active connections share the available transmission resources using TCP or a "TCP friendly" approach. Under this hypothesis it is possible to use a max-min fair [50]

rate allocation algorithm in order to evaluate the available capacity for each connection, given the link bandwidth constraints. The notion of max-min fair allocation is based on the following premises: i) no entity should receive an allocation larger than its demand, and ii) increasing the allocation of any entity should not result in the decrease of the allocation of another entity that received an equal or smaller allocation. It well approximates the TCP-like sharing uploading and downloading bandwidth between concurrent flows.

Evaluating the max-min fair rate allocation in a network of hundred of thousand peers, with millions of active connections is not an easy task. The classical centralized implementation of max-min fair rate allocation (as suggested for instance in [51]) does not scale well for the network dimensions of our interest. The overall computational load of max-min fair allocation in our scenario is the product of two different factors. On the one hand, a max-min fair re-computation is required every time a new traffic relation is established, or an old traffic relation is completed or interrupted (e.g. because of peer disconnection), the frequency of these events being linearly dependent on the number of simulated peers . On the other hand, the implementation suggested in [51] requires to re-compute the allocated rates per each network node, and thus it results in a complexity which grows linearly with the number of simulated peers. Whereas the first factor depends only on the simulated P2P application logic, it is possible to act on the second factor to reduce the computational load of max-min fair rate allocation. In fact, as it was observed in [50], when a new connection is established or an old connection is interrupted or completed, such events may affect only a subset of the existing connections. The above observation have been exploited to develop an exact and more efficient max-min fair rate allocation implementation under the assumption of bottleneck links only in the access side of the network. More details about such implementation may be found in [50]. The reported results show that the algorithm proposed in [50] outperforms traditional max-min computation approaches by as much as a factor 100 for a million nodes network.

We have developed OPSS starting from the implementation of the max-min fair rate allocation algorithm proposed in [50]. As in [50] we made the assumption that rate bottlenecks occur only in the access part of the network. This assumption is employed in both analytical models appeared in the literature [52] as well as in

simulation programs such as [53] and [40]. It is justified by the current bandwidth gap between access links and core network trunk, and by the empirical observation that practical P2P clients typically further throttle the upload bandwidth, which in most cases results fully utilized by the uploading connections.

Obviously, the most serious limit in our approach is that the max-min fair bandwidth allocation well approximates a TCP-like steady state bandwidth sharing. Due to this, our approach is well suited to the case of persistent connections between peers. In addition, it is currently impossible to simulate Transit-Stub topologies such as the ones generated by GT-ITM topology generator. To overcome this last limit extending the efficient and exact implementation of max-min fair rate allocation proposed in [50] to the case of generic topologies, even if not trivial, could be a reasonable solution.

4.3.2 Implementation details

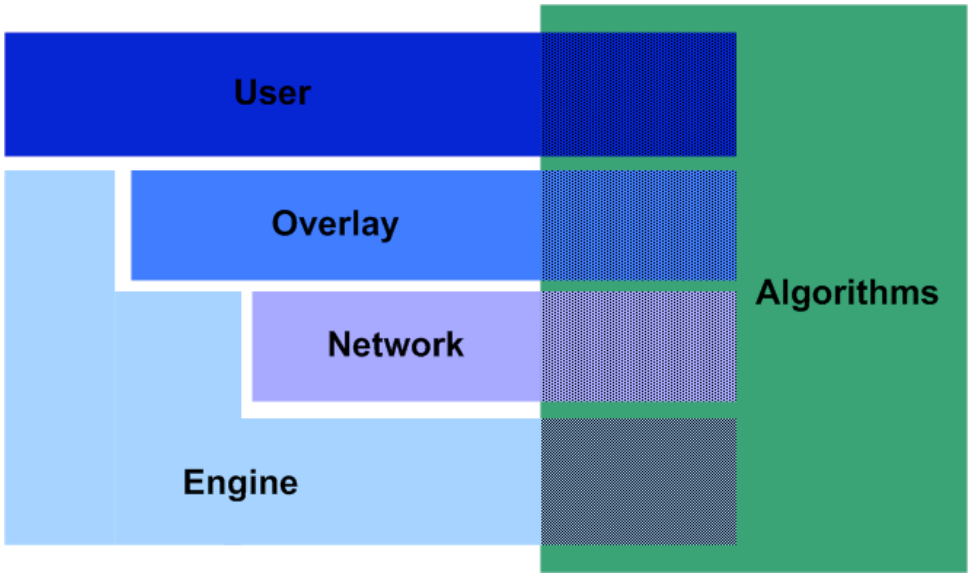

Figure 4.2. OPSS Architecture

OPSS is written in C++ and is publicly available [54] under GPL license. It was designed according to a modular implementation logic. Figure 4.2 illustrates the main blocks of the simulator architecture. As the figure shows, it is possible

to identify three layers: User, Overlay and Network. User layer represents the peer behavior, taking into account for example connection and disconnection policies (i.e. the 'churn behavior'). Overlay layer is responsible to simulate the overlay network and the overlay interactions between peers. Network layer represents the network behavior, and it currently implements the optimized max-min fair rate allocation approach as described in [50]. All the above layers interact with the Engine block, which contains the discrete-event-related classes and mages the event executions. Engine block is also responsible for output log file where events are dumped with the corresponding time. The set of events that will be included in the log file is customizable to prevent log files to become too big in size. While User, Overlay, Network and Engine include the basic structures common to any P2P streaming system, the Algorithms block is responsible of the P2P streaming algorithm and application to be simulated, including the control communication between nodes and the scheduling algorithm of stream segments. It inherits the basic structures of User, Overlay, Network and Engine, and allows to customize them. In such a way, it allows the implementation of any P2P live streaming mechanism.

The previously described approach makes OPSS a very flexible simulator, as it offers the possibility of implementing the logic of P2P streaming application as separate module. Moreover, simulator users may exploit different basic classes provided by User, Overlay, Network and Engine blocks and potentially implement any kind of P2P streaming algorithms. For further information about how to write algorithms, please refer to the guide available on the reference site.

4.3.3 Performance metrics

In this section we discuss the performance metrics we may evaluate using OPSS. Due to the characteristics of the application (streaming of real-time multimedia flows), the considered metrics are basically related to the delay of the received chunks. In the definition and evaluation of these delay related metrics, we need also to carefully consider that some chunks may not be received by some receiver.

Consider a real time multimedia streaming system where the multimedia stream is divided in chunks of duration T [s]. The originator of chunks starts at $t = t_{start} = 0$, and ends up at $t = t_{end}$. The total number of originated chunks will

be t_{end}/T. In order to produce consistent measurements, we need to observe the system in an interval of duration $w_{end} - w_{start}$, with $0 \leq w_{start} < w_{end} \leq t_{end}$. For simplicity, we assume that w_{start} corresponds to the generation time of one given chunk, and we observe C chunks starting from the one created at w_{start}. Therefore the generation times of the observed chunks will be $t_c = w_{start}+(c-1)T$, $c = 1,2,\cdots,C$. The first observed chunk originates at $t_1 = w_{start}$ and the last observed chunk at $t_C = w_{start}+(C-1)T$. In order to allow the last chunk to be received by all receivers, we need to choose our observation interval end time w_{end} such that $w_{end} > t_C$. In particular, let be $w_{end} = t_C + D_{max}$, where D_{max} is the maximum delay we are considering in our evaluation of the system. Note that the origination of chunks will continue also after the origination of the last observed chunk, in the time interval in which we are still observing the system and waiting for the last observed chunk to be received.

Assume that there are $(N-1)$ receiving nodes [1] and that node n receives the c-th observed chunk at time $t_r(c,n) = w_{start} + (c-1)T + d(c,n)$. Then $d(c,n)$ is the delay of chunk c at node n. Let us consider the last chunk C. If $d(C,n) > D_{max}$, the event is out of our observation window and it will be lost. For the generic chunk c, the relative reception event at node n goes out of our observation window if $d(c,n) > D_{max}+(C-c+1)T$. From a methodological point of view, it is not good that the maximum observable delay for a chunk depends on the chunk number c. Therefore we think it should be better to set D_{max} as maximum chunk delay for all chunks and to consider a chunk c lost if $d(c,n) > D_{max}$.

We consider that a receiving node n can be active or not by defining its activity function $a(t,n)$ as follows: $a(t,n) = 1$ if node n is active at time t, $a(t,n) = 0$ if node n is not active at time t. The activity of node n during the observation window is:

$$A(n) = \frac{1}{w_{end} - w_{start}} \int_{w_{start}}^{w_{end}} a(t,n)dt \tag{4.1}$$

The number of active nodes at time t is given by:

$$N_A(t) = \sum_n a(t,n) \tag{4.2}$$

[1] The total node number is N if we add the stream source to the receiving nodes.

Therefore the average number of active nodes N_A over the observation window will be:

$$N_A = \frac{1}{w_{end} - w_{start}} \int_{w_{start}}^{w_{end}} N_A(t)dt \tag{4.3}$$

We define the chunk delivery ratio (CDR) for a chunk c (that depends on D_{max}) as the ratio between the nodes that have received the chunk c and the average number of active nodes when the chunk is originated. Note that we should consider the average number of active nodes in a time interval following the chunk origination event as this is the number of potential receivers for the chunk. As the chunk delivery delay is variable, it is not clear over which time interval we should average $N_A(t)$. A simpler solution is to define a "conventional" chunk delivery ratio for a chunk c using the overall average number N_A of active nodes over the observation window. This is reasonable if the average number of active nodes does not change over time. According to such assumption, the chunk delivery ratio relative to chunk c is:

$$CDR_{D_{max}}(c) = \frac{1}{N_A} \sum_{n} r_{D_{max}}(c,n) \tag{4.4}$$

where $r_{D_{max}}(c,n) = 1$ if chunk c is received by node n with $d(c,n) \leq D_{max}$, while $r_{D_{max}}(c,n) = 0$ if chunk c either is not received or is received with $d(c,n) > D_{max}$.

We also define the overall chunk delivery ratio (that depends on D_{max} as well) as:

$$CDR_{D_{max}} = \frac{1}{C} \sum_{c} CDR_{D_{max}}(c) = \frac{1}{C \cdot N_A} \sum_{c,n} r_{D_{max}}(c,n) \tag{4.5}$$

Metrics related to the chunk delivery ratio are very common in P2P streaming performance studies [19][20]. These metrics can measure the continuity of the playback; however, to better characterize live streaming applications, it is advisable to take into account also the absolute delay of chunks at the peer nodes. To this purpose, OPSS allows also to evaluate metrics related to $d(c,n)$, that represents the time interval between the time instant at which chunk c is available at stream source and the time instant at which node n completes the download of chunk c. Specifically, we consider the average chunk delay for a chunk c, which can be evaluated by averaging the delay over all nodes that received that chunk (with a delay lower

than or equal to D_{max}):

$$\overline{d}^{chunk}_{D_{max}}(c) = \frac{1}{N_A \cdot CDR_{D_{max}}(c)} \sum_{n|r_{D_{max}}(c,n)=1} d(c,n) \tag{4.6}$$

It could be interesting to consider the perspective of a given node n and to evaluate the perceived performances. First of all, we can evaluate the chunk delivery ratio seen by a given node n:

$$CDR^{node}_{D_{max}}(n) = \frac{1}{C \cdot A(n)} \sum_{c} r_{D_{max}}(c,n) \tag{4.7}$$

We can also evaluate the average chunk delay perceived by a generic node n:

$$\overline{d}^{node}_{D_{max}}(n) = \frac{1}{C \cdot A(n) \cdot CDR^{node}_{D_{max}}(n)} \sum_{c|r_{D_{max}}(c,n)=1} d(c,n) \tag{4.8}$$

It is also interesting to consider the α-chunk delay percentile (CDP) of the distribution of chunk delay perceived by a generic node n. Typical values that we can consider are $\alpha = 95$, $\alpha = 99$.

$$CDP^{node}_{D_{max}}(\alpha,n) = x | Prob\{d(c,n) < x\} = \frac{\alpha}{100} \tag{4.9}$$

The overall average chunk delay can be evaluated by averaging the delay over all received chunks (with a delay lower than or equal to D_{max}):

$$\overline{d}_{D_{max}} = \frac{1}{C \cdot N_A \cdot CDR_{D_{max}}} \sum_{c,n|r_{D_{max}}(c,n)=1} d(c,n) \tag{4.10}$$

Note that this is different from averaging $\overline{d}^{chunk}_{D_{max}}(c)$ over c.

4.3.4 The evaluated P2P streaming algorithms

Goal of this section is to show OPSS performance in terms of both scalability and capability of producing correct results. To this purpose, we simulated two "trivial"

P2P streaming distribution schemes ("Balanced M-ary tree" and "Trivial Mesh"), for which we could easily derive analytical models. We compared the experimental results achieved by OPSS with the results of the analytical model, and verified OPSS correctness. On the other hand, OPSS scalability was simply evaluated by simulating an ever increasing number of nodes. In the following subsections, we describe the simulated distribution schemes and we report the corresponding analytical models and experimental results.

4.3.4.1 Balanced M-ary tree

This stream distribution scheme corresponds to a balanced M-ary distribution tree. The stream source is the root of the tree. The stream video is divided into segments or chunks, and $R = 1/T$ denotes the source chunk rate [chunk/s]. Each node downloads chunks from one single node, and it uploads chunks to M nodes. According to the tree graphs jargon, each node has M children. We assume that all nodes (including the stream source) join the system simultaneously and form the distribution tree. We also assume a static situation, in which all nodes persist through the whole lifetime of the simulation.

Due to the last assumption, the issues related to the number of active nodes at a time instant and the average number of active nodes may be neglected. Another assumption we make is that $w_{start} = t_1 = 0$, where according to subsection 4.3.3 w_{start} and t_1 denote respectively the observation start time and the first chunk creation time. The c-th chunk will be referred to as c and the relative creation time is $t_c = (c-1)/R$.

We now introduce the level concept. With reference to a node, the level l represents its distance from stream source as number of hops in the overlay tree. The level of the stream source is $l = 0$, while the last level is denoted as $l = L$. If all levels are complete, the number of nodes at level l is M^l and the total number of nodes is $N = \sum_{l=1}^{L} M^l$. In the following, we will always consider trees with complete levels.

All nodes are assigned an access link with uplink and downlink capacities W_{up} and W_{down} [chunk/s]. To simplify, we assume symmetrical access links, that means also $W = W_{up} = W_{down}$. As consequence, each node downloads chunks at W/M

[chunk/s] from its father in the tree. If $W/M < R$, the distribution system cannot work as each node does not have enough capacity to download the stream of chunks. We thus restrict our attention to the case in which the available portion of the father's uplink capacity is greater than or equal to the rate of the stream to be received: $W/M \geq R$.

It is convenient to express the delay of chunk c at node n in terms of the corresponding node level l. The reason is that all nodes at the same level perceive the same delay. Specifically, given the level l, $l = 1,2,\cdots,L$, and the chunk c, the corresponding chunk delay is

$$d(c,l) = \frac{M}{W} \cdot l \tag{4.11}$$

The average chunk delay for chunk c is

$$\begin{aligned} \overline{d}^{chunk}(c) &= \frac{\sum_{l=1}^{L} M^l \cdot d(c,l)}{\sum_{l=1}^{L} M^l} \\ &= \frac{M\{M^l\,[L\,(M-1)-1]+1\}}{W\,(M-1)\,(M^l-1)} \end{aligned} \tag{4.12}$$

Note that it does not depend on c. The reference to D_{max} is omitted, as we are assuming an ideal system where chunks are not lost. We only need to set $D_{max} > L \cdot W/M$ so that we are able to observe all chunk reception events.

The average chunk delay perceived by a generic node at level l is

$$\overline{d}^{node}(l) = \frac{1}{C}\sum_{c=1}^{C} d(c,l) = \frac{M}{W} \cdot l \tag{4.13}$$

The assumption in (4.13) is that C chunks are observed during the observation window.

Let us now consider the experimental results we achieved by setting $M = 2$. In more detail, we simulated a set of binary distribution trees by varying the maximum depth L of the tree in the range [1,17] (e.g. the node number ranges from 3 to 131171). Figure 4.3 illustrates the binary tree corresponding to $L = 4$. Observation time end was set to 3600 sec. Moreover, $W = 2$ [chunk/s] and $R = 1$ [chunk/s] were used as upload capacity and chunk creation rate. We highlight the "deterministic" nature of the simulations, since there is no variability in the input data and all

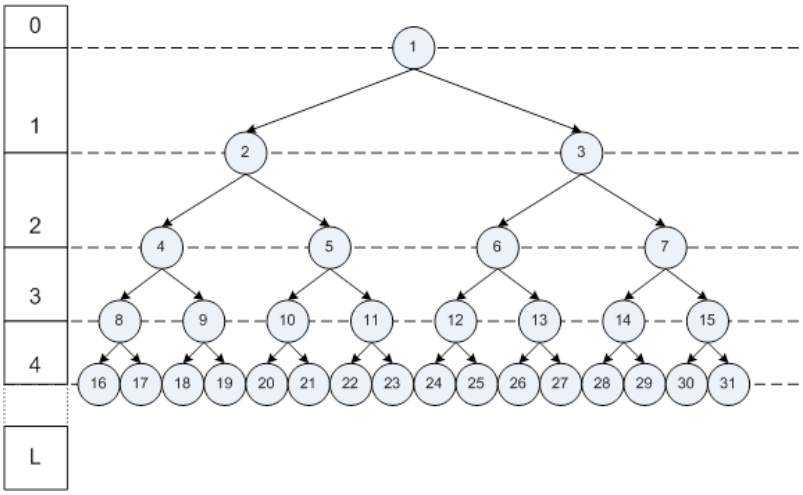

Figure 4.3. Binary distribution tree with N=31 nodes

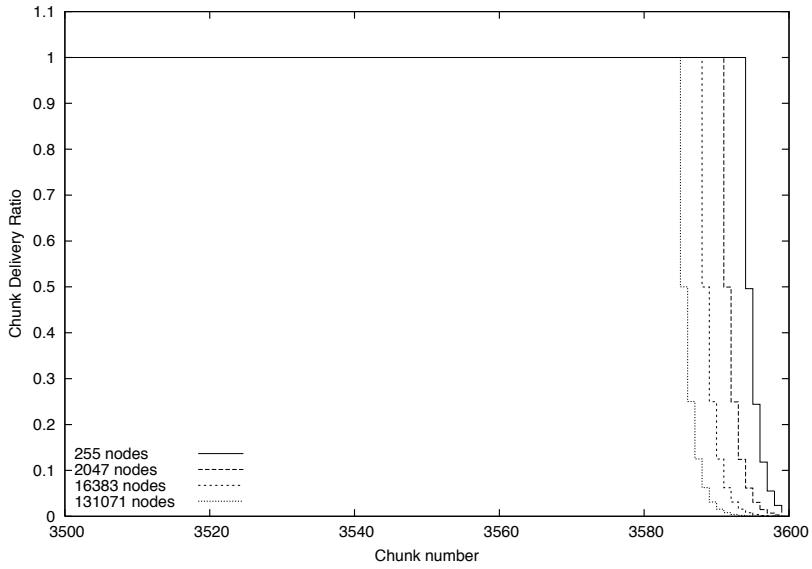

Figure 4.4. Chunk delivery ratio

nodes are always active. Therefore, the results of each simulation are relative to a single simulation run and we do not indicate any confidence interval (the same considerations apply to the next subsection).

Figure 4.4 shows the chunk delivery ratio $CDR(c)$ defined in (4.4) versus c. As the observation end time w_{end} is 3600 sec and the last observed chunk is generated at 3600, we are not able to observe the complete diffusion of all chunks created during the observation window. The correct approach in order to observe all reception events is set C to the greatest c such that the condition $(L + c - 1)M/W < w_{end}$ is verified. For instance, when the simulated nodes are 255, solving the above condition leads to $c = 3593$, corresponding to the highest vertical step in the curve for 255 nodes.

The cumulative distribution function of the average chunk delay for a given chunk $\overline{d}^{chunk}(c)$, defined in (4.12), is illustrated in Figure 4.5. According to equation (4.12), the cumulative distribution function confirms that the average chunk delay is constant, and there is a perfect matching between the values achieved by equation (4.12) and the values achieved by simulation.

Figure 4.6 shows the cumulative distribution function of the average chunk delay perceived by the generic node. Given a number of simulated nodes, the steps

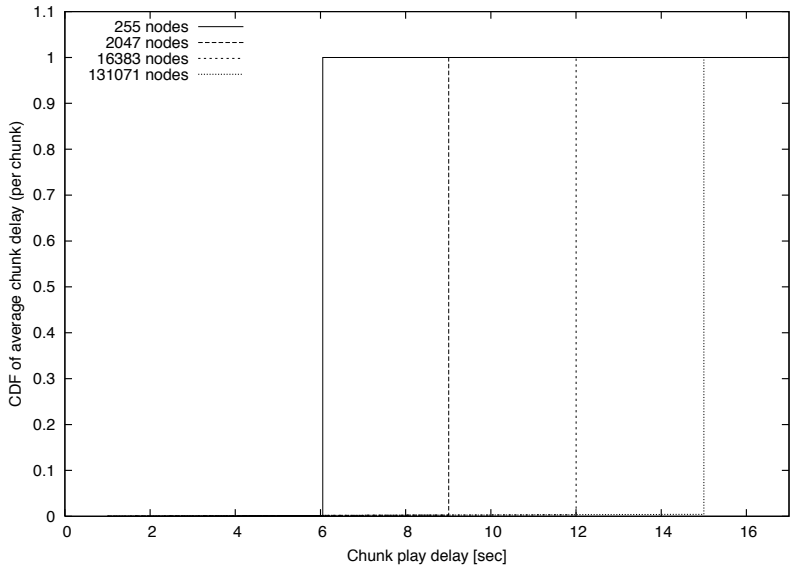

Figure 4.5. Cumulative distribution function of average chunk delay for a given chunk

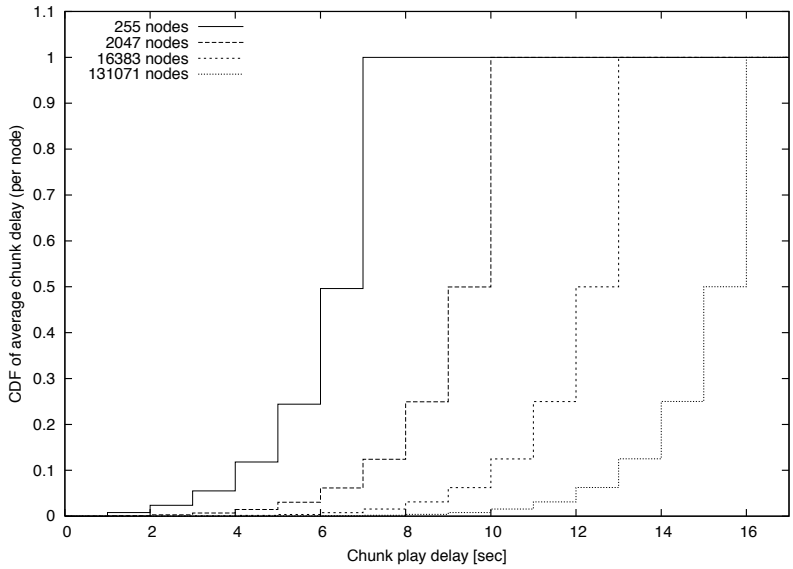

Figure 4.6. Cumulative distribution function of average chunk delay at a given level

correspond to the different tree levels and their probability values may be deduced from the ratio between the number of nodes in that level and the total number of nodes.

4.3.4.2 Trivial mesh

The algorithm represents a really simple streaming distribution system based on mesh topologies. Specifically, the stream source generates chunks at rate R [chunk/sec] and it uploads S chunks simultaneously to S nodes. All other nodes (i.e. excluding the source node) maintain S connections for downloading chunks and S connections for uploading chunks, and they use them simultaneously to upload/download different chunks. Thus nodes form groups of S nodes. The first S nodes are connected directly with stream source and download the stream chunks from it. The second group of S nodes opens a connection with each node of the first group to download available chunks, and so on. In such a way, if we further assume that the total number of nodes is $N = L \cdot S + 1$, it is possible to identify L different levels, each consisting of a group of S nodes. The previous assumption implies also that all levels are complete. Figure 4.7 shows the trivial mesh topology corresponding to $N = 17$ and $S = 4$.

Building the overlay topology according to the above trivial mesh scheme is not smart, as the underlying algorithm does not allow to increase the number of peers at each level: the tree depth increases linearly with the node number and the chunk transfer delay increases as well. Anyway, the purpose of our work now is just to check OPSS capability of producing the expected results, and we are going to provide an analytical model of this trivial mesh. Like in the previous experiment, we

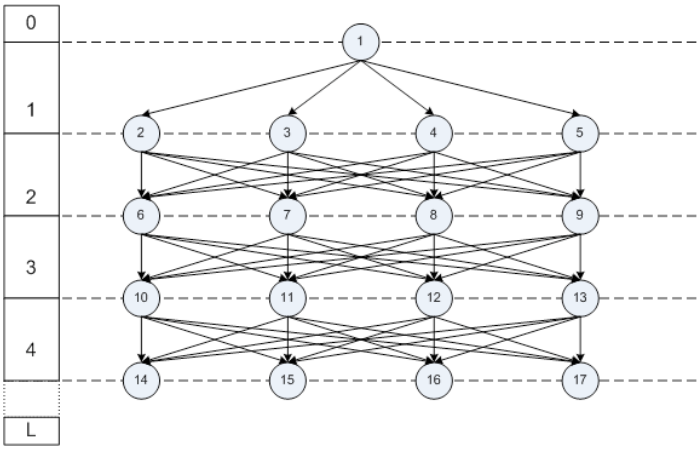

Figure 4.7. Trivial mesh with N=17 nodes and S=4

assume that nodes persist through the whole simulation time. Moreover, with regard to access link upload and download capacities, the condition $W = W_{up} = W_{down}$ [chunk/s] holds. There is however one exception: the stream source has a capacity that is S times the capacity of other nodes. This means that the S nodes connected to the stream source download chunks at rate W. The other nodes may download S chunks in parallel from S different suppliers, and each chunk is downloaded at a rate of W/S [chunk/s]. We will focus on the case $W = R$.

Since all nodes in a level perceive the same chunk delay, we can refer to levels instead of individual nodes. With regard to the observation window, we assume that i) $w_{start} = t_1 = 0$, ii) the creation time of the generic c-th chunk is $t_c = (c-1)/R$. In addition, we assume that source node and nodes at level 1 join the system at time w_{start}, while all other nodes join the system simultaneously at time instant $t_S = S/R$, i.e. when the first S chunks have already been created and transferred to nodes at level 1. Therefore, at instant t_S each node of level 2 will open S connections to download the first S chunks from the S nodes at level 1. The download of these S chunks will last S/W [s]. As in our hypothesis, the downloads will last exactly S/R or equivalently ST [s]. When such downloads end, there will be S new chunks available at nodes at level 2 and nodes at level 3 will start S new downloads from nodes at level 2. This procedure is straightforwardly replicated in all levels below.

Given the level l, $l = 1,2,\cdots,L$, and the chunk c, the corresponding chunk delay is

$$d(c,l) = \frac{S}{R} + \left[ceil\left(\frac{c}{S}\right) + l - 1 \right] \frac{S}{W} - \frac{c-1}{R} \tag{4.14}$$

This delay will be periodic of period S, as a burst of S chunks will be received at the same time by nodes at same level. In particular, the most recent chunk of the burst will experience the lowest delay, the oldest chunk of the burst will experience the highest delay and the delay difference among two next chunks in the burst is T.

The average chunk delay for the chunk c is

$$\begin{aligned} \overline{d}^{chunk}(c) &= \frac{\sum_{l=1}^{L} S \cdot d(c,l)}{\sum_{l=1}^{L} S} \\ &= \frac{S}{2W}(L-1) + \frac{S}{W} ceil\left(\frac{c}{S}\right) - \frac{c-1-S}{R} \end{aligned} \tag{4.15}$$

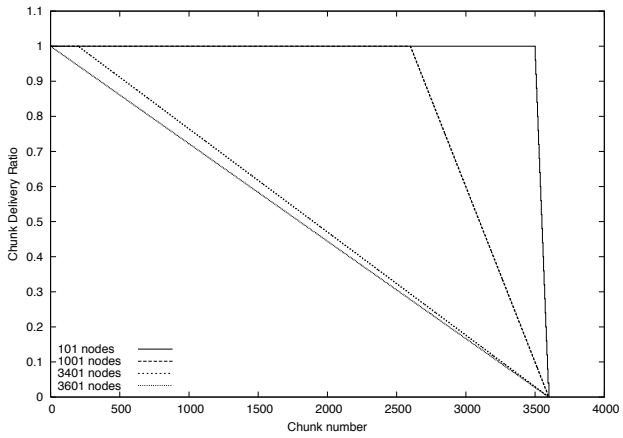

Figure 4.8. Chunk delivery ratio

The average chunk delay perceived by a generic node at level l is evaluated under the assumption that the number C of observed chunks is an integer multiple of S, e.g. $C = J \cdot S$ with $J \in Z^+$. Specifically, it results:

$$\begin{aligned} \overline{d}^{node}(l) &= \frac{1}{C}\sum_{c=1}^{C} d(c,l) \\ &= \frac{S\left[R\left(J-1+2l\right)-W\left(J-1\right)\right]}{2RW} \end{aligned} \tag{4.16}$$

At this point we report the experimental results we achieved by setting $S = 4$. The node number ranges from 101 to 3601 passing through 1001 and 3401. Observation end time was set to $w_{end} = 3600$. Moreover, $W = R = 1$ [chunk/s].

The chunk delivery ratio is shown in Figure 4.8. According to formula (4.16), the average chunk delay at level l is linearly dependent of l. This means that, due to the fixed observation window, when the node number and consequently the level number grow, an ever increasing number of chunk reception events will be lost. When the node number is 3601, the first created chunk is received only by nodes in the first $(L-1)$ levels. In reason of this, simulating a higher node number does not make sense if the observation window is not accordingly increased.

The cumulative distribution of the average chunk delay for a given chunk is illustrated in Figure 4.9. As it can be seen, each curve is characterized by two different parts: the first one grows almost linearly and accounts for the chunks that

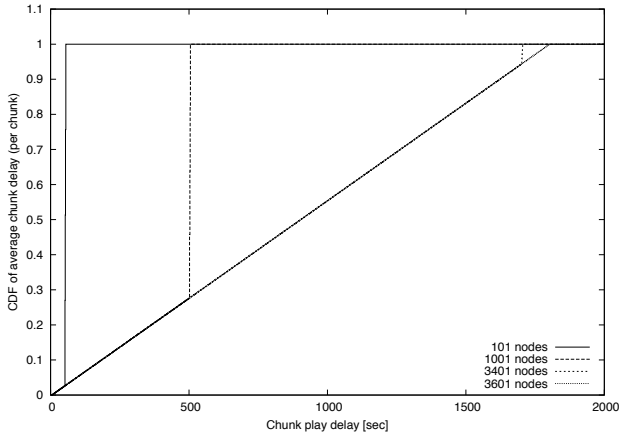

Figure 4.9. Cumulative distribution function of average chunk delay for a given chunk

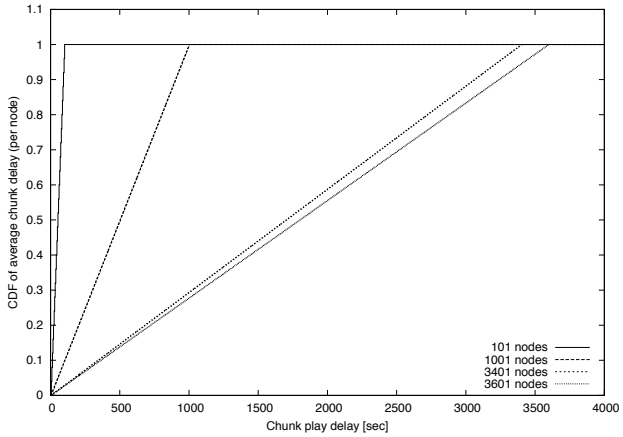

Figure 4.10. Cumulative distribution function of average chunk delay at a given level

are not received by all nodes within the observation window; the second one relates to the chunks that all nodes receive within the observation window and it is almost vertical (actually there are S different values separated by 1 second, as the average chunk delay is periodic with period S and the values it may take increase by step $T = 1/R$).

Figure 4.10 shows the cumulative distribution function of average chunk delay at a given level. The reported curves exhibit a step trend with each step corresponding to a specific level, even if it is difficult to appreciate the step trend, as the node number and consequently the level number grow. This complies with equation

(4.16), even if the last one has been derived under the approximation of observed chunk number multiple of S.

4.3.5 Simulator performance

Just to give an idea about OPSS performance, we provide some details about the simulation computational load on the pc-desktop we used for our experiments. Specifically, the pc was equipped with a 3.2 GHz bi-processor CPU and 4 Gigabyte RAM. Table 4.1 refers to the binary tree algorithm and shows, for a given node number, the time necessary to complete the simulation, the resulting log file size and the time necessary to analyze log file with an our own C++ application. The analysis results have been then used to achieve the previously reported graphs.

Binary tree algorithm			
Nodes	Simulation time	Log file size	Analysis time
131071	$\sim 6h$	15.6GB	$\sim 5h$

Table 4.1. Main computational load parameters

4.4 O-Streamline performance evaluation

In this section we evaluate the performance of *O-Streamline* by considering the following performance indexes:

- $\overline{N}_{served}(c,d)$: is the average number of nodes that complete the download of a chunk c with a chunk delivery delay less than or equal to d; the chunk delivery delay is the difference between the time instant at which the chunk arrives at the source and the time instant at which the chunk is completely received from a node. Note that this performance index is very similar to the performance index $N(c,k,t)$ used to evaluate the performance of *Streamline*. In fact, if in $N(c,k,t)$ we consider the chunk delivery delay $t-c-1$ instead of the absolute time t of chunk delivery and we average on all chunks, we obtain $\overline{N}_{served}(c,d)$;
- chunk delivery ratio $R(n,S_n)$: with reference to a node n, the chunk delivery ratio is the ratio between the number of (completely) received chunks and the

total number of chunks generated during the duration of the session of that node, S_n.

We evaluated these performance indexes via simulations by using the OPSS simulator [66]. The duration of the simulations is 1800 s. For each simulation result we assessed the 95% confidence intervals. These intervals are plotted in the figures only when they are significant; when they are small (e.g. less than 5%) they are not shown to improve the readability of the figures themselves.

We simulated a homogeneous scenario in terms of downlink and uplink capacity: all nodes, including the source, have the same uplink and downlink capacity. As regards the downlink capacity, like in *Streamline*, we set a value great enough so that downlinks are not a bottleneck of the system. As regards the uplink capacity, we remove another assumption of *Streamline*, i.e., that the ratio between the uplink capacity of all nodes and the the stream bit rate is equal to 1. In *O-Streamline* we consider also values of this ratio greater than or equal to 1. As a consequence, when this ratio is greater than one, we had to use the OPSS simulator also to evaluate the performance of *Streamline*.

As regards the churn model we make some assumptions, which must not be seen as too oversimplified, since the main goal of this paper is to present a practical and working theory-driven algorithm and to test its robustness against churn rather than to give a real model of churn. These assumptions are:

- the session times are exponentially distributed;
- when a peer node leaves the system, a new peer node joins the system. This makes possible to keep the number of simulated peer nodes constant throughout the whole simulation and to simplify the evaluation of simulation results;
- each node may accept up to X connections per group besides the number P or O of connections per group[2]. This guarantees a topology perturbation after node disconnections. In fact, accepting more connections than the minimum number P or O, implies that new peer nodes may be placed in topological positions different from the ones of the corresponding peer nodes which are

[2]In all the simulations X is a random variable with uniform distribution in the interval (0,2).

going to disconnect and to be replaced. In addition, also peer nodes loosing a neighbor due to a disconnection may look for another neighbor and maintain P or O (depending on the group to which the disconnecting node belongs) as minimum number of connection per group.

The presentation of simulation results is organized as follows. First, we refer to a churn-free simulation scenario where peer nodes are always on (subsection 4.4.1). This allows us to compare *O-Streamline* with *Streamline*, thus understanding *O-Streamline*'s position with respect to ideal performance bounds. Then, we introduce the churn and evaluate its implications on the performance of *O-Streamline* (subsection 4.4.2).

4.4.1 Comparison with Streamline, no churn

We assume that the number P of neighbors in the same group to which the generic peer belongs is equal to the number O of neighbors in the remaining groups.

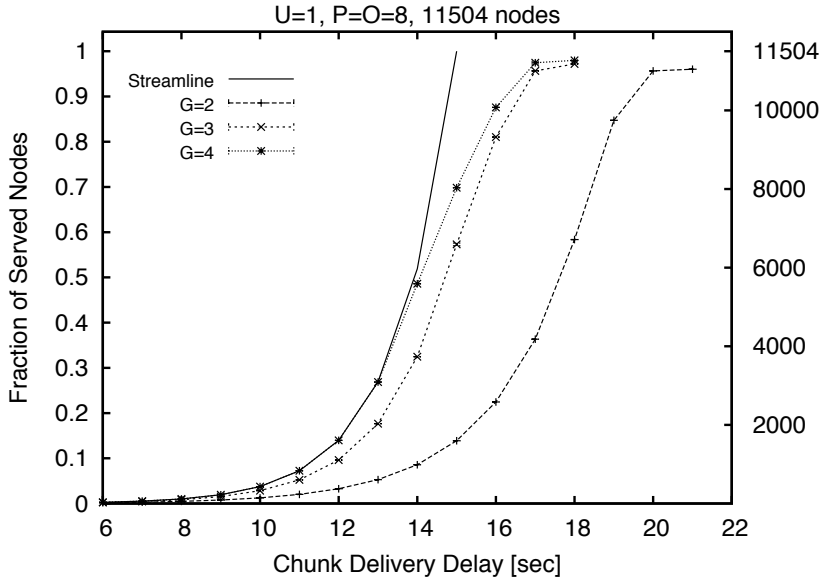

Figure 4.11. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *Streamline* and *O-Streamline* varying the number of groups.

We first consider a scenario with 11504 nodes, 8 neighbors per group ($P = O = 8$) and a ratio between uplink capacity and stream bit rate equal to 1.

In figure 4.11 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$ for three different values of the number G of groups. The curve relative to *Streamline* has been analytically derived by setting $k = 4$. In fact, even if the performance of *Streamline* improve monotonically as a function of the number k of children, we chose $k = 4$ since the improvement becomes negligible for greater values of k (see [67] for more details). We observe that as the number of groups increases, the performance of *O-Streamline* get closer to the ones of *Streamline*. For instance, with 4 groups, the difference between *O-Streamline* and *Streamline* is negligible for chunk delivery delay up to $12 - 13$ seconds; the maximum chunk delivery delay necessary to serve the whole network of *O-Streamline* is 3 seconds higher than that of *Streamline*. With 3 groups the maximum chunk delivery delay of *O-Streamline* does not worsen, while with 2 groups the maximum chunk deliver is 6 seconds higher than the maximum chunk delivery of *Streamline*.

As a consequence, we can state that there exists a threshold for the number of groups G over which improvements become negligible. In addition the number of groups G should be chosen so as to limit the signaling burden required to maintain the relationships with neighbors belonging to all the G groups.

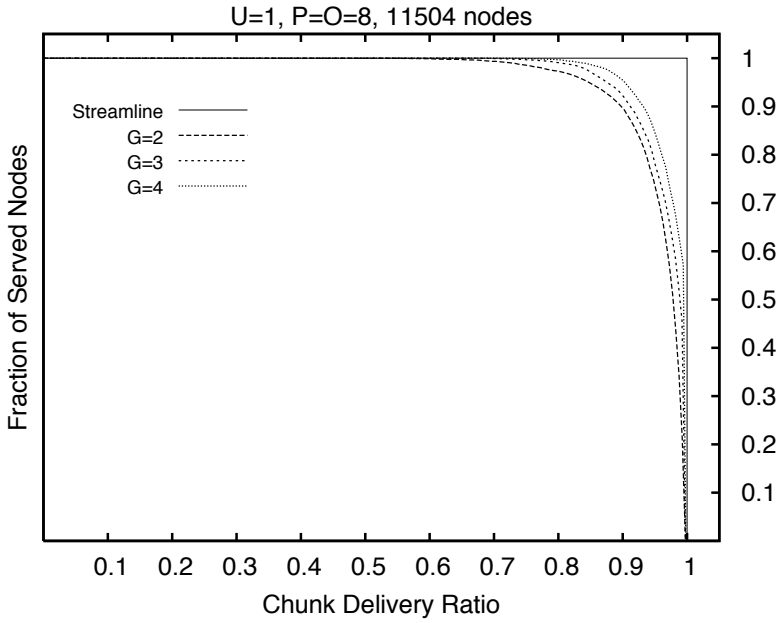

Figure 4.12. Inverse cumulative distribution of chunk delivery ratio in *Streamline* and *O-Streamline*.

In figure 4.12 we plot the inverse cumulative distribution function[3] of $R(n,S_n)$ (the chunk delivery ratio), for three different values of the number G of groups. As in the previous figure, the curve relative to *Streamline* has been analytically derived by setting $k = 4$. We also note that the chunk delivery ratio in *Streamline* is equal to 1 for every chunk and every node. As expected, we observe that *Streamline* succeeds in completely diffusing all chunks. Instead, *O-Streamline* has a chunk delivery ratio that increases with the number of groups. For instance 95% of nodes download at least 80%, 85% and 90% of all chunks with 2, 3 and 4 groups, respectively.

Now we consider a second scenario with 17472 nodes, 2 groups and a ratio between uplink capacity and stream bit rate equal to 2.

In figure 4.13 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$ for three different values of the number of neighbors per group (we recall that we assumed $P = O$). The curve relative to *Streamline* has been derived via simulations under the assumption of 4 parents/children per node.

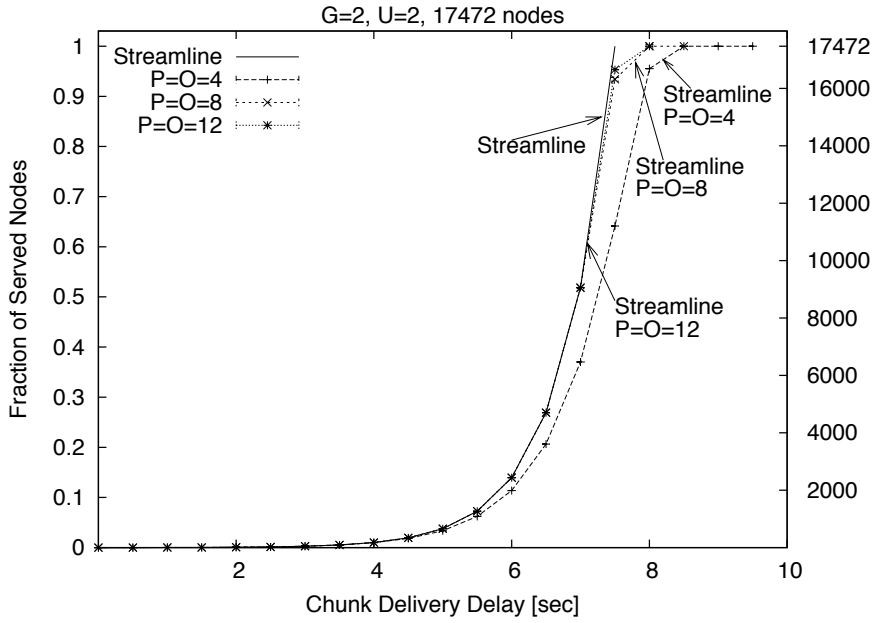

Figure 4.13. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *Streamline* and *O-Streamline* varying the number of neighbors per group.

We observe that as the number of neighbors per group increases, the performance

[3] i.e. the complement to one of the cumulative distribution function

of *O-Streamline* get closer to the ones of *Streamline*. For instance, with 12 neighbors per group, the difference between *O-Streamline* and *Streamline* is negligible for chunk delivery delay up to 6 seconds; the maximum chunk delivery delay necessary to serve the whole network of *O-Streamline* is 1 second higher than that of *Streamline*.

Another observation is that the performance of *O-Streamline* get closer to the ones of *Streamline* as the ratio between uplink capacity and stream bit rate increases. As a matter of fact if we compare figure 4.11 and figure 4.13 for the cases of $G = 2$ and $P = O = 8$ the maximum chunk delivery delay is 21 s for $U = 1$ and 8.5 s for $U = 2$.

Finally, we note that we do not present results on the chunk delivery ratio in this scenario, since the choice of $U = 2$ guarantees the complete diffusion of all chunks, regardless of the number of neighbors per group.

4.4.2 Impact of churn

We consider a scenario with 17472 nodes and a ratio between uplink capacity and stream bit rate equal to 2. The average session time of each node is equal to 10 minutes.

In figure 4.14 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$ for three different values of the number G of groups. The number of neighbors per group is 8 ($P = O = 8$). We observe that: i) the performance worsen with respect to what happens in absence of churn, ii) the lower the number of groups, the more significant the worsening, iii) a single group implies a maximum chunk delivery delay of about 25 seconds; using 2 or 3 groups makes possible to reduce this delay to about $16 - 17$ seconds.

In figure 4.15 we plot the inverse cumulative distribution function[4] of $R(n,S_n)$ (the chunk delivery ratio), for three different values of the number of groups. The number of neighbors per group is 8 ($P = O = 8$). Also this performance index improves as a function of the number of groups.

In figure 4.16 we plot the cumulative distribution function of $\overline{N}_{served}(c,d)$, for three different values of the number of neighbors per group (we recall that $P = O$). We observe that: i) performance worsen with respect to performance in absence of

[4] i.e. the complement to one of the cumulative distribution function

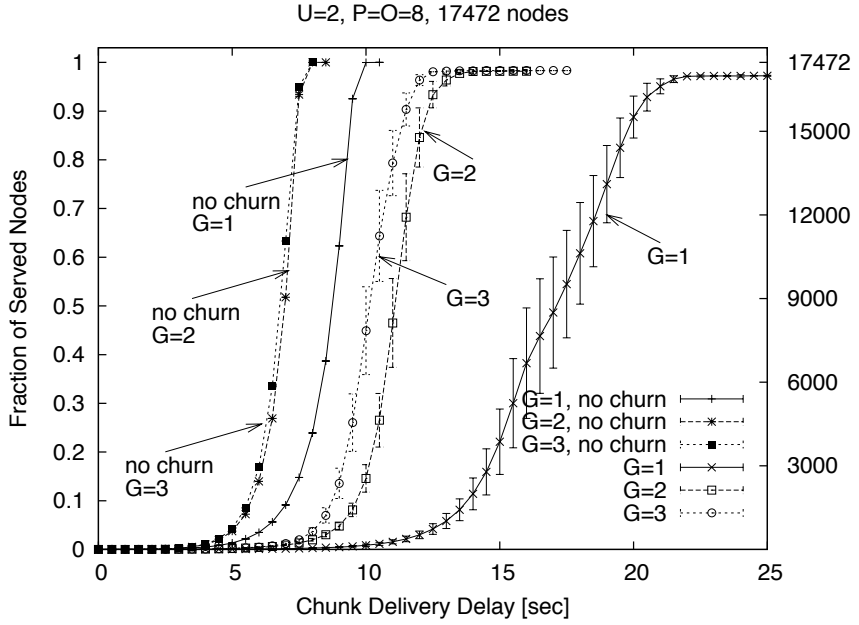

Figure 4.14. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *O-Streamline* in case of peer churn.

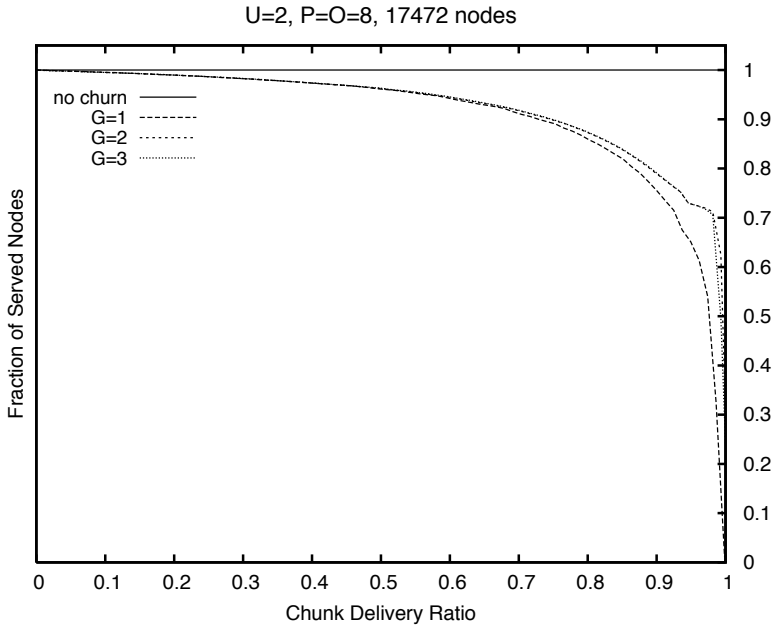

Figure 4.15. Inverse cumulative distribution of chunk delivery ratio in *O-Streamline* in case of churn.

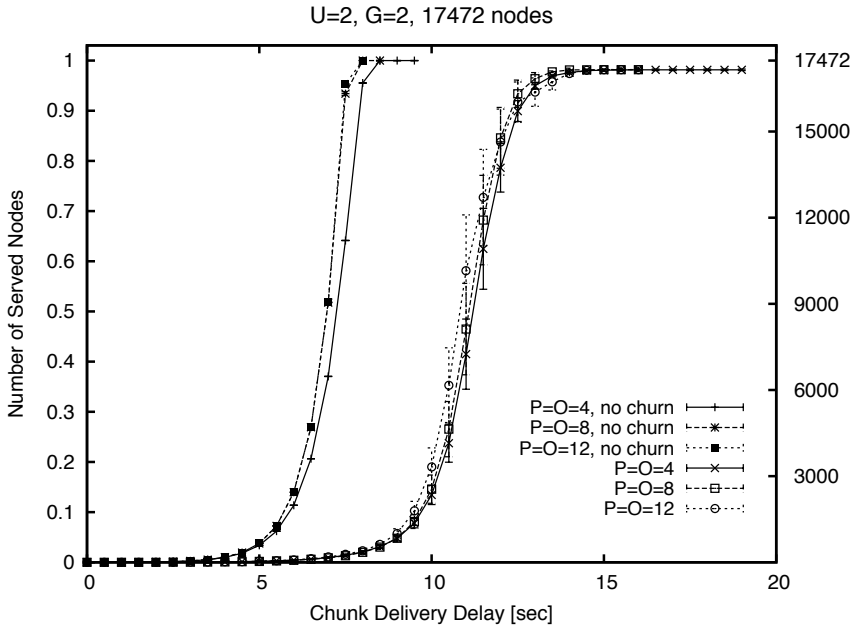

Figure 4.16. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *O-Streamline* in case of a churn

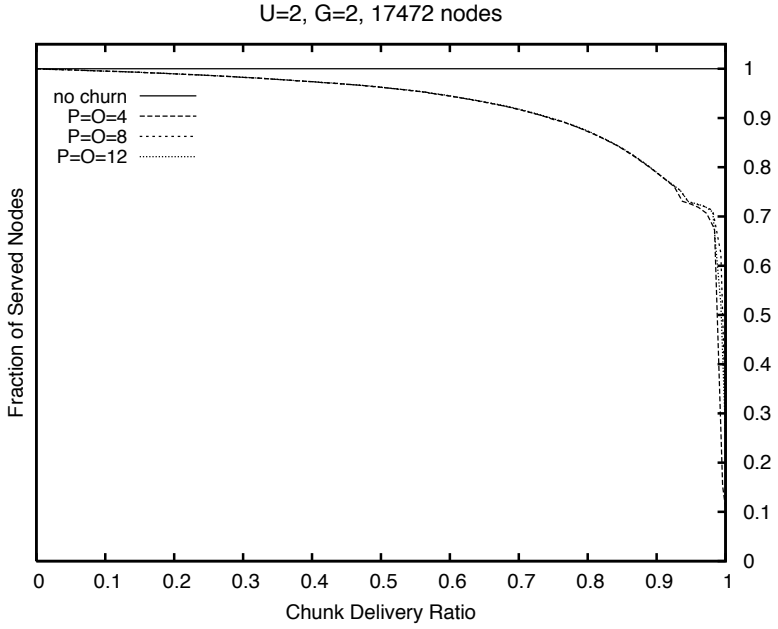

Figure 4.17. Inverse cumulative distribution of chunk delivery ratio in *O-Streamline* in case of a churn

churn, ii) increasing the number of neighbors per group improves mostly the maximum chunk delivery delay and only marginally the lower values of chunk delivery delay.

In figure 4.17 we plot the inverse cumulative distribution function[5] of $R(n,S_n)$ (the chunk delivery ratio), for three different values of the number of neighbors per group (we recall that $P = O$). The variation of the number of neighbors per group does not affect significantly the performance.

[5]i.e. the complement to one of the cumulative distribution function

Conclusions

Goal of this dissertation thesis was to analyse multimedia P2P systems and to give contributions in the related research.

It started with a review of the state of the art of p2p streaming systems (chapter 1), analysing both structured and unstructured architectures. The result of this study is that it is hard to decide which one performs better, basically because those system are very complex and the main way to deal with them is to perform simulations or to use testbeds. Using these methodologies (simulations and testbeds) it is possible in general to derive comparative measurements between different algorithms, but it seems to be a lack of absolute reference values.

For this reason a theoretical performance bound for chunk-based P2P streaming systems has been found and presented in chapter 2. Such a bound has been derived in terms of the stream diffusion metric, a performance metric which is directly related to the end-to-end minimum delay achievable in a P2P streaming system. The presented bound depends on i) the upload bandwidth available at each node, assumed homogeneous for all nodes, and ii) the number of neighbors to transmit chunks to. k-step Fibonacci sequences play a fundamental role in such a bound. The importance of the presented theoretical bound is twofold: on the one hand, it provides an analytical reference for performance evaluation of chunk-based P2P streaming systems; on the other hand, it suggests some basic principles, which can be exploited to design real-world applications.

From the lesson learned in the theoretical study, a "practical" algorithm called O-Streamline is designed and presented in chapter 3. Basically the theoretical work suggests i) the serialization of chunk transmissions, and ii) the organization of chunks

in different groups so that chunks in different groups are spread according to different paths. O-Streamline exploits these principles realizing them by means of pure distributed and data-oriented operation (i.e. chunk paths are not pre-established). This assures an intrinsic stability of the algorithm respect to peer churn.

The O-Streamline performance analysis is assessed in chapter 4. To perform this analysis, there is the need of a simulator that can reproduce chunk transfers in a large scale network. Unfortunately to the best of our knowledge, there does not exist an "open" simulator which is able to reproduce performance of systems composed by hundred thousands nodes: usually authors write ad-hoc simulator for each work. For this reason an open-source simulator called OPSS is developed and released as open-source software. This tool allows to perform performance analysis of the O-Streamline algorithm both in a still network than in a network with peer churn. Simulation results confirm that algorithm performances in term of delay are very close to the theoretical bound.

Rank	Application	2007	2009	Change
1	Web	41.68	52.00	+10.31
2	Video	1.58	2.64	+1.05
3	VPN	1.04	1.41	+0.38
4	Email	1.41	1.38	-0.03
5	News	1.75	0.97	-0.78
6	P2P	2.96	0.85*	-2.11
7	Games	0.38	0.49	+0.12
8	SSH	0.19	0.28	-0.08
9	DNS	0.20	0.17	-0.04
10	FTP	0.21	0.14	-0.07
	Other	2.56	2.67	+0.11
	Unclassified	46.03	37.00	-9.03

Figure 4.18. Weighted average percentage Internet traffic. Change is in terms of percentage of all Internet traffic.

In conclusion, from when I started to dealt with multimedia P2P streaming lot of things changed. In a recent work, authors of [69] present the largest study of global Internet traffic since the start of the commercial Internet. Authors analyze two years of detailed traffic statistics covering 256 Exabytes of Internet traffic across 110 large and geographically diverse cable operators, international transit backbones, regional networks and content providers.

From this analysis emerges a global decline of P2P filesharing and a dramatic rise in video traffic as we can see by Fig. 4.18: web (and video over HTTP) has the largest and faster growing while Filesharing P2P (whose classification is operated on well port basis) is the fastest shrinking. This happens because i) Most P2P uses random ports and 40% or more are encrypted (so it is difficult to classify) ii) P2P Filesharing has been partially replaced by Direct Download iii) browser is becoming the main application front end (e.g. for mail, video). In other word, the world of "naive" Napster-like applications is doomed to fall down as well as new technologies, probably web-based, will appears.

For what concern video, for instance, now its traffic share is estimated to represent about the 25%+ of all the Internet traffic, but more and more video are carried with HTTP and Flash. In turn Flash player, from version 10, enrich their virtual machine with peer assisted networking functionalities using the Real Time Media Flow Protocol (RTMFP). This can mitigate the heavy bandwidth requirements that multimedia providers should offer to a still growing number of users that demands more quality (for instance HD videos). Even if applications front-end change, it seems like the peer to peer paradigm applied to multimedia contents and its relative solutions and fundamentals will play an important role for a long time.

Bibliography

[1] Napster web site, http://www.napster.com/

[2] SETI@home project web site, http://setiathome.berkeley.edu/

[3] Compute Against Cancer web site, http://www.computeagainstcancer.org/

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, *Wide-area cooperative storage with CFS*, in Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), Chateau Lake Lou, Banff, Canada, 2001

[5] Skype, http://en.wikipedia.org/wiki/Skype

[6] BitTorrent web site, http://www.bittorrent.com/

[7] B. Cohen, *Incentives build robustness in BitTorrent*, on line available at http://www.bittorrent.com/bittorrentecon.pdf

[8] Gnutella Protocol specification 04, on line available at http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf

[9] Gnutella Protocol specification 06, on line available at http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[10] Kazaa web site, http://www.kazaa.com

[11] N. Leibowitz, M. Ripeanu, A. Wierzbicki, *Deconstructing the Kazaa network*, in Proceedings of 3rd IEEE Workshop on Internet Applications (WIAPP'03), Santa Clara (CA), USA, 2003

[12] J. Liang, R. Kumar, K. W. Ross, *The FastTrack overlay: a measurement study*, in Computer Networks Journal (Elsevier), Volume 50, Issue 6, Pages 842-858, 2006

[13] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulie, and S. Patarin, *Peer sharing behaviour in the eDonkey network, and implications for the design of server-less file sharing systems*, in Proceedings of the 1st ACM

SIGOPS EuroSys (EuroSys2006), Leuven, Belgium, 2006

[14] C. Diot, B. Levine, B. Lyles, H. Kassem, D. Balensiefen, *Deployment issues for the IP multicast service and architecture*, in IEEE Network, vol. 14 (1), 10-20, 2000

[15] S. Banerjee, B. Bhattacharjee, C. Kommareddy, *Scalable application layer multicast*, in Proceedings of ACM SIGCOMM, Pittsburgh, PA, USA, 2002

[16] D. A. Tran, K. A. Hua, T. Do, *ZIGZAG: an efficient peer-to-peer scheme for media streaming*, in Proceedings of IEEE INFOCOM, San Francisco, CA, USA, 2003

[17] V. N. Padmanabhan, H. J. Wang, P. A. Chou, *Distributing streaming media content using cooperative networking*, in Proceedings of NOSSDAV, Miami Beach, FL, USA, 2002

[18] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh, *SplitStream: high-bandwidth multicast in cooperative environments*, in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), The Sagamore, Bolton Landing, NY, USA, 2003

[19] X. Zhang, J.C. Liu, B. Li, P. Yum, *CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming*, in Proceedings of IEEE INFOCOM, Miami, FL, USA, 2005

[20] M. Zhang, L. Zhao, Y. Tang, J. Luo, S. Yang, *Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet*, in Proceedings of ACM Multimedia 2005, Singapore, Singapore, 2005

[21] N.Magharei, R.Rejaie, *PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming*, in Proc. of IEEE INFOCOM, 2007.

[22] M. Zhang, Y. Xiong,, Q. Zhang, S. Yang, *On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks*, in Proceedings of IEEE Globecom 2006, San Francisco, CA, USA, 2006

[23] L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, G. Bianchi and N. Blefari-Melazzi, *A push-based scheduling algorithm for large scale P2P live streaming*, in Proceedings of QoS-IP 2008, on line available at `netgroup.uniroma2.it/p2p/QoSIP08.pdf`.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, *SCRIBE: A large-scale and decentralized application-level multicast infrastructure*, in IEEE Journal on

Selected Areas in Communications, 2002.

[25] A. Rowstron, P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, in Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001

[26] PPLive web site, `www.pplive.com/en/index.html`

[27] X. Hei, C. Liang, J. Liang, Y. Liu, K. Ross, *Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System*, in Proceedings of IPTV Workshop, International World Wide Web Conference, Edinburgh, Scotland, 2006

[28] *PlanetLab testbed*, `http://www.planet-lab.org/`

[29] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, *PlanetLab: An Overlay Testbed for Broad-Coverage Services*, in ACM Computer Communications Review, vol. 33, no. 3, 2003

[30] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A.E. Mohr, *Chainsaw: Eliminating Trees from Overlay Multicast*, in Proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS'05), Ithaca, NY, USA, 2005

[31] K. Calvert, E. Zegura, S. Bhattacharjee, *How to Model an Internetwork*, in Proceedings of IEEE INFOCOM, San Francisco, CA, USA, 1996

[32] Y. Chu, S. G. Rao, H. Zhang, *A case for end system multicast*, in Proceedings of ACM SIGMETRICS 2000, Santa Clara, CA,USA, 2000

[33] SopCast web site, `http://www.sopcast.org/`

[34] S. Ali, A. Mathur, H. Zhang, *Measurement of Commercial Peer-To-Peer Live Video Streaming*, in Proceedings of Workshop on Recent Advances in P2P Streaming, Waterloo, ON, Canada, 2006

[35] *Aurora simulator*, `http://freenet.cvs.sourceforge.net/freenet/aurora/`

[36] *Serapis simulator*, `http://freenet.cvs.sourceforge.net/freenet/Serapis/`

[37] T. M. Gil, F. Kaashoek, J. Li, R. Morris, J. Stribling, *P2Psim simulator*, `http://pdos.csail.mit.edu/p2psim/`

[38] Rice University, *FreePastry simulator*, `http://freepastry.org/FreePastry/download.html`

[39] I. Stoica, M. Walfish, *Chord simulator*, `http://cvs.pdos.csail.mit.edu/cvs/~checkout~/sfsnet/simulator/`

[40] Neurogrid simulator , `http://www.neurogrid.net/php/simulation.php`

[41] S. Joseph, *An extendible open source P2P simulator*, P2P Journal, 1-15, 2003

[42] N. S. Ting, R. Deters, *3LS - A Peer-to-Peer network simulator*, in Proceedings of 3rd International Conference on Peer-to-Peer Computing, Linkping, Sweden, 2003

[43] M. Jelasity, G. P. Jesi, A. Montresor, S. Voulgaris, *PeerSim simulator*, `http://peersim.sourceforge.net/`

[44] M. Jelasity, A. Montresor, O. Babaoglu, *A modular paradigm for building self-organizing peer-to-peer applications*, in Proceedings of the International Workshop on Engineering Self-Organising Applications (ESOA'03), Melbourne, Australia, 2003

[45] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, *GnutellaSim simulator*, `http://www-static.cc.gatech.edu/computing/compass/gnutella/`

[46] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, *Mapping peer behavior to packet-level details: a framework for packet-level simulation of Peer-to-Peer systems*, in Proceedings of MASCOTS 2003, Orlando (FL), USA, 2003

[47] *The Network Simulator ns-2*, `http://www.isi.edu/nsnam/ns/`

[48] L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, *Simulation of Peer-to-Peer Streaming over Large-Scale Network using Opss*, in Proceedings of the First International Workshop on Network Simulation Tools (NSTools) Workshop, October 22, 2007 Nantes

[49] L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, *OPSS an overlay peer to peer streaming simulator for large scale networks*, special issue of ACM SIGMETRICS Performance Evaluation Review, Volume 35, Issue 3 December 2007

[50] F. Lo Piccolo, G. Bianchi, S. Cassella, *Efficient simulation of bandwidth allocation dynamics in P2P Networks*, in Proceedings of Globecom 2006, San Francisco, CA, USA, 2006

[51] D. Bertsekas, R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1987

[52] D.Qiu, R.Srikant, *Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks*, in Proceedings of ACM SIGCOMM'04, Portland, OR

(USA), September 2004

[53] M.Baker, T. Giuli, *A scalable flow-based network simulator*,Technical report, Stanford University, 2002

[54] OPSS web site, `http:/opss.sourceforge.net`

[55] C.J. Bovy, H.T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal and P. Van Mieghem, *Analysis of End-to-end Delay Measurements in Internet*, in Proceedings of the Passive and Active Measurements Workshop (PAM2002), Fort Collins, CO, USA, 2002 (on line available at `http://www.ripe.net/projects/ttm/Documents/PAM2002_TUD.pdf`).

[56] F. Pianese, D. Perino, J. Keller, E. Biersack, *PULSE: an adaptive, incentive-based, unstructured P2P live streaming system*, IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks, Volume 9, N. 6, 2007.

[57] X. Yang and G. De Veciana, *Service Capacity of Peer to Peer Networks*, in Proc. of IEEE INFOCOM, 2004.

[58] Y. Liu, *On the minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?*, in Proc. of ACM Multimedia, 2007.

[59] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, M. Chiang, *Performance Bounds for Peer-Assisted Live Streaming*, in the Proc. of ACM Sigmetrics, 2008.

[60] R. Kumar, Y. Liu, K. Ross *Stochastic Fluid Theory for P2P Streaming Systems*, in Proc. of IEEE INFOCOM, 2007.

[61] Z. Li, B. Li, D. Jiang, L. Chi Lau, *On achieving optimal throughput with network coding*, in Proc. of IEEE INFOCOM, 2005.

[62] M. Zhang, Q. Zhang, L. Sun, S. Yang, *Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?*, in IEEE Journal of Selected Areas in Communications, 2007.

[63] E. P. Miles, *Generalized Fibonacci numbers and associated matrices*, The American Mathematical Monthly, Vol. 67, No. 8 (Oct. 1960), pp. 745-752.

[64] E. W. Weisstein, *Fibonacci n-step number*, published electronically at http://mathworld.wolfram.com/Fibonaccin-StepNumber.html

[65] E. W. Biersack, P. Rodriguez, and P. Felber, *Performance Analysis of Peer-to-Peer Networks for File Distribution*, in Proc. of QofIS'04, 2004.

[66] *OPSS simulator*, `http://netgroup.uniroma2.it/twiki/bin/view.cgi/Netgroup/OpssPublicPage`

[67] G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, S. Salsano and D. Luzzi, *Streamline: an Optimal Distribution Algorithm for Peer-to-Peer Real-time Streaming over Forest-Based Topologies*, submitted for pubblication and on line available at `netgroup.uniroma2.it/p2p/streamline.pdf`.

[68] L. Bracciale, D. Luzzi, F. Lo Piccolo, G. Bianchi, N. Blefari Melazzi, S. Salsano, *A Theory-Driven Distribution Algorithm for Peer-to-Peer Real Time Streaming* , in Proc. of IEEE GLOBECOMM, 2008.

[69] C. Labovitz, D. McPherson, and S. Lekel-Johnson, *2009 Internet Observatory Report* Arbor Networks

Appendix A: Publications

- L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano; "OPSS an overlay peer to peer streaming simulator for large scale networks", special issue of ACM SIGMETRICS Performance Evaluation Review, Volume 35, Issue 3 December 2007

- L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano; "Simulation of Peer-to-Peer Streaming over Large-Scale Network using Opss", First International Workshop on Network Simulation Tools (NSTools) Workshop, October 22, 2007 Nantes

- L. Bracciale, F. Lo Piccolo, D. Luzzi, G. Bianchi, N. Blefari Melazzi, S. Salsano; "A push-based scheduling algorithm for large scale P2P live streaming.", 4th International Telecommunication NEtworking WorkShop on QoS in Multimedia IP Networks (QoS-IP), February 13 - 15 2008, Venice, Italy

- L. Bracciale, F. Lo Piccolo, D. Luzzi, N. Blefari Melazzi, G. Bianchi, S. Salsano; "A Theory-Driven Distribution Algorithm for Peer-to-Peer Real Time Streaming", Gruppo nazionale Telecomunicazioni e Teoria dell'Informazione (GTTI 2008), Firenze, Italy, June 2008

- L. Bracciale, F. Lo Piccolo, D. Luzzi, N. Blefari Melazzi, G. Bianchi, S. Salsano; "A Theory-Driven Distribution Algorithm for Peer-to-Peer Real Time Streaming", Proc. of IEEE Globecom 2008, New Orleans, LA, November 2008

- G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, S. Salsano; "Fundamental Delay Bounds in Peer-to-Peer Chunk-based Real-time Streaming Systems", Proc. of 21st International Teletraffic Congress, Paris 2009-06-10

- F. Lo Piccolo , D. Battaglino, L. Bracciale, M. Di Filippo, A. Bragagnini, M. S. Turolla N. Blefari Melazzi "Towards fully IP-enabled IEEE 802.15.4 LR-WPANs", Demos and Posters for the Sixth Annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks

- G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, S. Salsano, "Streamline: an Optimal Distribution Algorithm for Peer-to-Peer Real-time Streaming" , IEEE Transactions on Parallel and Distributed Systems, TPDS-2008-05-0208.R1

- A.Detti, L.Bracciale, F.Fedi: "Robust data Replication Algorithm for MANETs with Obstacles and Node Failures", IEEE International Conference on Communications (ICC) 23-27 May 2010, Cape Town, South Africa

- D. Battaglino, L. Bracciale, F. Lo Piccolo, A. Bragagnini, M. S. Turolla, N. Blefari Melazzi: "Self-configuring and IP-enabled Low Rate Wireless Personal Area Networks based on 6LoWPAN" and IEEE 802.15.4, IEEE INFOCOM Demo Session, San Diego (CA) 2010

- D. Battaglino, L. Bracciale, F. Lo Piccolo, A. Bragagnini, M. S. Turolla, N. Blefari Melazzi: "On the IP support in IEEE 802.15.4 LR WPANs: self configuring solutions for real application scenarios" Annual Mediterranean ad hoc networking workshop (Med-Hoc-Net) 23-25 June 2010. Juan les pins (France)

- D. Battaglino, L. Bracciale, F. Lo Piccolo, A. Bragagnini, M. S. Turolla, N. Blefari Melazzi: "IPv6 solutions enabling mobile services for the Internet of Things" World Telecommunications Congress WTC 2010- Vienna, Austria-13.-14. September 2010