Streamline: an Optimal Distribution Algorithm for Peer-to-Peer Real-time Streaming

G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, S. Salsano

Abstract—In this paper we propose and evaluate an overlay distribution algorithm for P2P, chunk-based, streaming systems over forest-based topologies. In such systems, the stream is divided in chunks; chunks are delivered by each node in a store-and-forward way. A relaying node starts distributing a chunk only when it has completed its reception from another node. Peers are logically organized in a forest of trees, where each tree includes all peers. The source periodically distributes different chunks to each tree for their delivery.

Our key idea consists in employing serial transmission: for each tree and thus for each chunk, the source node sends the chunk to its children in series; the same holds for each peer node of the tree, excluding the leaves.

Besides this basic idea, the contributions of this paper are: 1) we demonstrate the feasibility of serial transmission over a forest of trees, which is not a trivial problem, unlike the case of parallel transmission; 2) we derive an analytical model to evaluate the system performance; 3) we derive a theoretical bound for the number of nodes reachable in a given time interval or equivalently for the time required to reach a given number of nodes; 4) we prove the optimality of our approach in terms of its capability to reach such bound; 5) we develop a general simulation package for P2P streaming systems and we use it to compare our solution to literature results.

Finally, we stress that this paper is focused on the theoretical properties and performance understanding of the proposed distribution algorithm, rather than on its practical implementation in a real system. However, we briefly describe also a practical workable implementation of our algorithm.

Index Terms—Distributed Systems, Distributed applications, Performance of Systems, Performance attributes.

1 Introduction

Peer-to-peer (P2P) overlay systems are being proposed to stream multimedia audio and video content from a source to a large number of end-users.

In this scenario, the most natural and earlier solution for deploying a P2P streaming system is to organize peer nodes in one or more overlay multicast trees, and hence continuously deliver the streamed information across the formed paths. This is the case in [1], [2], [3]. Since the information, organized in the form of small IP packets, is sequentially delivered across these trees, the processing time of packets at each network node is marginal, and performance bounds depend only on the "optimality" of the formed distribution paths (with respect to some meaningful performance metric). However, in practice, this approach may not be viable in large-scale systems, and with nodes characterized by intermittent connectivity (churn). In fact, whenever a node in the middle of a path abruptly disconnects, complex procedures would be necessary to i) allow the reconstruction of the distribution path, and ii) allow the nodes affected by such event to recover the amount of information lost during the path reconfiguration phases. To overcome such limitations, a completely different approach, called *data-driven*, delivers content on the basis of content availability information, locally exchanged among connected peers, without any a priori pre-established path. This approach essentially creates a mesh topology among overlay nodes. Several proposed solutions, such as [4], [5], [6], [7], adopt the data-driven approach.

In this paper, 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. The data-driven solutions proposed in [4], [6], [7] 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. Thus, a chunk contains a number of packets which is greater than one; typical chunk sizes are in the order of 60 kbytes [4]. Instead, systems such as [1], [2], [3] still handle the information in store-and-forward fashion but with the important difference that the basic unit of this operation is the IP packet, which is considerably smaller than a typical chunk.

As for the overlay distribution topology, we too assume a so-called forest-based topology: different chunks are cyclically distributed across a finite set of distribution trees, where each tree includes all peers and each tree is used to distribute a sub-set of chunks.

The main novelty of our approach consists in using a serial transmission: for each tree and thus for each chunk, the source node sends the chunk to its children in series; the same holds for each peer node of the tree, excluding the leaves.

• *The Authors are with the Dipartimento di Ingegneria Elettronica (DIE), Universitá di Roma "Tor Vergata", via del Politecnico 1, 00133, Rome, Italy. E-mail: {giuseppe.bianchi, blefari, lorenzo.bracciale, francesca.lopiccolo, stefano.salsano}@uniroma2.it*

This may seem a small difference with respect to literature proposals, but we will show that this choice has important consequences. The idea behind serial transmission is that giving all the uplink capacity to a single child, instead of sharing it among multiple children, allows that child to start serving other nodes before than in the case of parallel transmission, and before than in any other ways of dividing the available capacity. This means that all nodes, except the last served ones, can start serving their children in the smallest possible amount of time; in turn, served children can become fathers of other children in the smallest possible amount of time, and so on.

From the point of view of a given node, serial transmission, parallel transmission, or any other ways of dividing the available uplink capacity are the same, i.e. there is not a particular advantage, for a given node (provided that all the capacity is fully used). However, if we look at the system on the whole, we will show that serializing the chunk distribution brings about significance performance advantages in terms of number of nodes reached in a given time interval or equivalently in terms of time required to reach a given number of nodes. We show that serial transmission is the best possible policy, for whatever value of the number of nodes to be reached, and of the time instant in which the system is observed.

The main contributions of this paper are:

- we demonstrate the feasibility of serial transmission over a forest of trees, i.e., that serial transmission does not prevent the realization of multiple distribution trees *without conflicts in the uplink capacity*; as it will become clear later on, this is not a trivial result;
- we derive an analytical model to evaluate the system performance, based on k-step Fibonacci numbers; since the sum of the first t values of a k-step Fibonacci series plays a fundamental role in our analytical model, a side result of our work is the derivation of a novel explicit expression for such sum and for its asymptotic value for large values of t;
- we derive a theoretical performance bound for the number of nodes reachable in a given time interval, or equivalently for the time required to reach a given number of nodes; this bound applies to any chunk-based distribution algorithm operating over a forest topology;
- we prove the optimality of our algorithm in terms of its capability to reach such bound;
- we find an asymptotic expression of the above performance bound in terms of the number of nodes receiving a chunk in a large time interval.
- we develop a simulation package and we use it to compare our solution to literature results.

Since we formally prove that our solution is optimal, in the sense explained above, we would not need to compare our algorithm to other solutions. However, we want to give a feeling of the quantitative improvements brought about by the proposed serial transmission. Ideally, we would like to compare analytically our solution to literature work. However, quite simply, analytical models of other solutions do not exist. For this reason, we follow a twofold approach: i) we perform an analytical comparison between *Streamline* and an ideal parallel transmission (conceived by ourselves for this purpose), which differs from *Streamline* only in the fact of employing parallel transmission instead of serial transmission; ii) we perform a comparison between *Streamline* and a reference literature solution, *SplitStream* [3], by using a suitably developed simulation package, based on our OPSS [8].

Finally, we emphasize that our paper is theory-oriented, and as such addressed to understand the performance limits achievable in forest-based topologies. Thus, we assume that the forest-based distribution topology is derived thanks to a sort of Maxwell's demon that holds a global and centralized vision of the whole network. This conceptual entity is able to instruct peer nodes on how to optimally organize the overlay forest-based topology and schedule chunk transmissions. Also, we do not take into account propagation delays induced by the underlying physical network topology. Practically speaking, we assume that propagation delays are negligible when compared with the chunk transmission time - indeed an assumption valid for most practical P2P streaming systems. Finally, we assume that there is no churn.

These are rather strong assumptions, but we need them to make the problem tractable. In this way we can set the framework of what is doable and achievable. The value of this paper lies in such theoretical analysis, which sheds new light on these phenomena.

However, we are aware that it is also important to show to the reader the possibility of really implementing in the real world a scheme based on our idea. For this reason, this paper includes a brief description of a practical workable implementation of our algorithm. As a matter of fact, we have shown in a conference paper [9] that it is possible to remove most of the assumptions listed above and design a practical and working algorithm, named Operational Streamline or simply *O-Streamline*. *O-Streamline* does not rely on a centralized vision of the whole network and takes the churn into account. In [9] we also evaluate, by means of simulations, the performance of *O-Streamline*, and we show that such performance is close to the optimal bounds derived for *Streamline* in this paper. *O-Streamline* and its analysis make possible to state that serial transmission is to be preferred also in a realistic environment, and give a definitive evidence of the superiority of our approach. The benefits of *Streamline* are realizable in practice and not limited to an ideal environment.

This paper is structured as follows. Section 2 introduces the framework of our proposal. Section 3 describes our proposal. In Section 4 we derive the analytical model of our algorithm and prove that our algorithm is optimal for the reference topology, in the sense explained above. Section 5 assesses the performance of our algorithm. Section 7 discusses the related work. Section 8 gives some concluding remarks. Finally, Appendix A derives some results on k-step Fibonacci sums, necessary to evaluate the performance of our algorithm.

2 Reference framework

We assume the following reference framework. We consider a stream generated by a single node (source) at a constant rate R (Kbps) and segmented into chunks with fixed size C

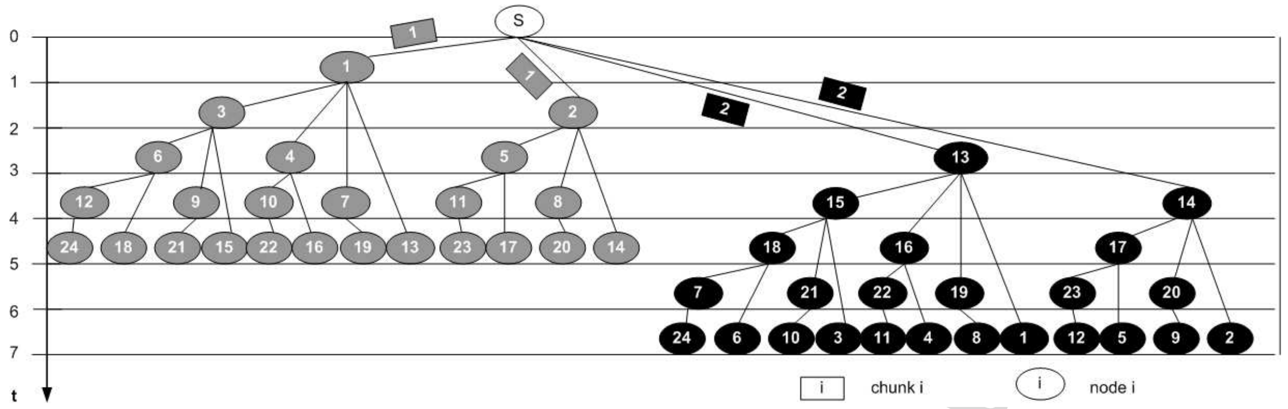

Fig. 1. *Streamline* distribution architecture for the case $U = 2, k = 4$ and a network of 24 nodes.

(Kb). The source starts transmitting at time instant $t = 0$. We define $T = C/R$ (s) the chunk duration, which is also the time elapsing between two consecutive chunks. Besides to the source, the network is composed of an unlimited number of nodes, which join the system simultaneously, are always on and are assumed to be homogeneous in terms of uplink capacity (equal to B Kbps) and downlink capacity (this simplification is commonly used also in other literature proposals). The downlink capacity is assumed to be great enough so that downlinks are not a bottleneck of the system. Each node has k overlay parents, from which it receives chunks, and k overlay children, to which it serves chunks, being the parent set potentially different from the children set. We assume that the ratio between the uplink capacity of peer nodes and the stream bitrate is equal to U, being U an integer number. Finally, we normalize all performance figures with respect to the time needed to download a complete chunk at rate B. Thus, the latter time interval will be the unit of time.

3 STREAMLINE

Our distribution architecture organizes peers in multiple overlay trees, in such a way that each tree includes all peers. The root nodes of the trees are children of the source. The source periodically distributes different chunks to each tree for their delivery in such a way that each tree is used to distribute a sub-set of chunks. For each tree, and thus for each chunk, the source node sends the chunk to its children in series; the same holds for each peer node of the tree, excluding the leaves.

From the point of view of a given node, serial transmission, or parallel transmission or any other way of dividing the available uplink capacity among children are the same, i.e. there is not a particular advantage, for a given node; however, if we look at the system on the whole, we will show that serializing the chunk distribution brings about significance performance advantages. In fact, all nodes, except the last served ones, start serving other nodes in the smallest possible amount of time.

We also assume that the number k of children is greater than U. This assumption is necessary in a forest-based topology. Since each node may transmit a given chunk to at most U children, the condition $k > U$ is necessary to organize peers in multiple overlay trees and transmit different sub-sets of chunks across different overlay trees. We also note that this conclusion is independent from the way of dividing the available uplink capacity among children. For the sake of simplicity, we assume that k is an integer multiple of U, even if it is possible to generalize our analysis to arbitrary integer values of k. Thus, the number of trees is k/U; the number of sub-sets of chunks is equal to k/U as well, and each sub-set of chunks is distributed along a tree.

The operation of our distribution architecture is illustrated in figure 1 for the case $U = 2, k = 4$ and a network of 24 nodes. The trees are $k/U = 2$ and there are two sub-set of chunks.

In this figure the source is denoted with an “S”. Nodes and 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, and 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. 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 $t = 0$; after finishing this transmission, i.e at $t = 1$, it sends the first chunk to node 2, in series. When the second chunk is available at the source (i.e. $t = 2$), the source serves children other than the ones to which the same source has served the first chunk. In more detail, the source starts transmitting the second chunk to node 13 at $t = 2$ and after finishing this transmission, i.e at $t = 3$, it sends the second chunk to node 14, in series.

In such a way, after transmitting the first chunk to nodes 3 and 4 in series (i.e. at $t = 3$), node 1 starts serving the first chunk also to the remaining children 7 and 13 in series. The same holds for node 2, which transmits the first chunk only three times instead of four, since all the nodes in the network have already received the first chunk at $t = 5$ and there are no other nodes to be served. In their turn, all the served nodes transmit the first chunk to their children (if any) in series. In the same manner, nodes 13 and 14 distribute in series the second chunk to their children, which in turn transmit the received chunk in series to their children (if any), and so on, until all nodes in the network receive the second chunk.

And what about the third and the fourth chunk? As soon

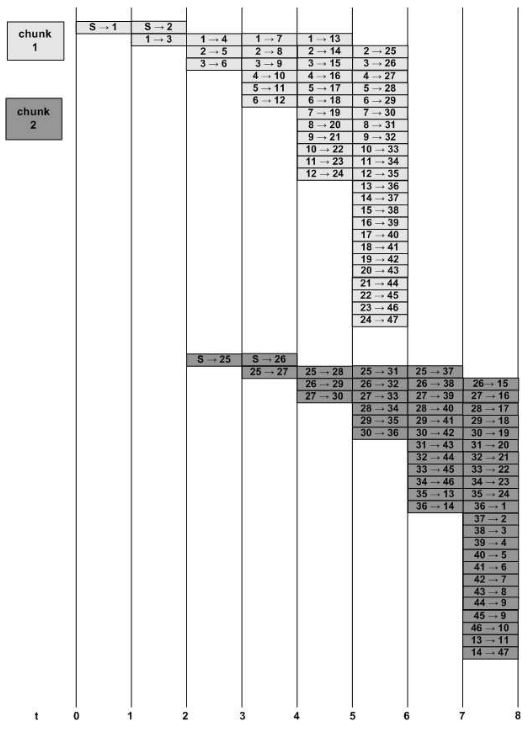

Fig. 2. *Streamline* distribution process of the first two chunks in the time interval $(0, 8)$ for the case $U = 2, k = 4$.

as the third chunk is available for the transmission at the source (i.e. at $t = 4$), the source transmits it to nodes 1 and 2 in series, and the third chunk is distributed along the *grey* distribution tree used for the first chunk. The fourth chunk is instead distributed along the *black* distribution tree, used for the second chunk. In other words, the system works as if there were two classes of chunks, grey-colored chunks and black-colored chunks, distributed through two different trees, respectively the grey-one and the black-one. In this example, grey-colored chunks are the even-index ones, while black-colored chunks are the odd-index ones. It is to be noted that each node in the grey (black) distribution tree receives a new grey (black) chunk every 4 time units and can serve all its 4 children, if such children have no still received the grey (black) chunk. Thus, the case $U = 2, k = 4$ is characterized by two distribution trees, which repeat themselves with period 4. In general, the k/U distribution trees repeat themselves with period k.

Figure 2 provides an alternative representation of the operation of the proposed distribution algorithm for the case $U = 2, k = 4$. From the notational point of view: i) the source is denoted with an “S”; ii) nodes are progressively indexed starting from 1; iii) in the notation $m \rightarrow n$, m denotes the sender node and n denotes the receiver node. The time is shown in the bottom part of the figure, from 0 to 8 (time units). The figure depicts which nodes are transmitting to which other nodes, for each time unit. Going from the left part of the figure to the right part, we see how chunks are progressively distributed starting from the source till they reach all nodes. Chunk 1 is available at the source at time 0, while chunk 2 is available at the source at time 2. The upper part of the figure makes possible to follow the transmission of the first chunk from the source to all nodes. The lower part refers to the second chunk. For instance, after the first time unit, node 1 transmits the first chunk to node 3, while the source transmits the same chunk to node 2; then, after the second time unit, nodes 1, 2 and 3 transmit the first chunk respectively to nodes 4, 5 and 6. The figure shows only the transmission of the first two chunks; however, as explained above, the other chunks will be transmitted by repeating the same pattern with period $k = 4$. The figure allows stating that the number of nodes that receive the first two chunks of the stream within 8 time units is 47.

To complete the definition of *Streamline* we must first solve a problem, which we call “tree intertwining” problem. For our distribution algorithm to work it is necessary that the k/U distribution trees are organized in such a way that no conflicts in uplink capacity arise.

In other words we must verify that there exists an organization of the k/U distribution trees such that a given node is not required to serve more than one receiver *at the same time*. Otherwise our approach would not work.

In the next subsection we describe the “tree intertwining” problem, by means of a low-dimension example, with $U = 1$; we could make the same description in more general terms, and using parameters instead of specific numbers but we find the numerical example more effective. The solution of the general “tree intertwining” problem can be found in [10], since we can not include it here for space limitations.

3.1 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 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 $1, 4, 7, \ldots$ and in general all chunks c with $c \bmod 3 = 1$;
- $T_{3,2}$ is used to transmit chunks $2, 5, 8, \ldots$ and in general all chunks c with $c \bmod 3 = 2$;
- $T_{3,3}$ is used to transmit chunks $3, 6, 9, \ldots$ and in general all chunks c with $c \bmod 3 = 0$.

Figure 3 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. 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 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

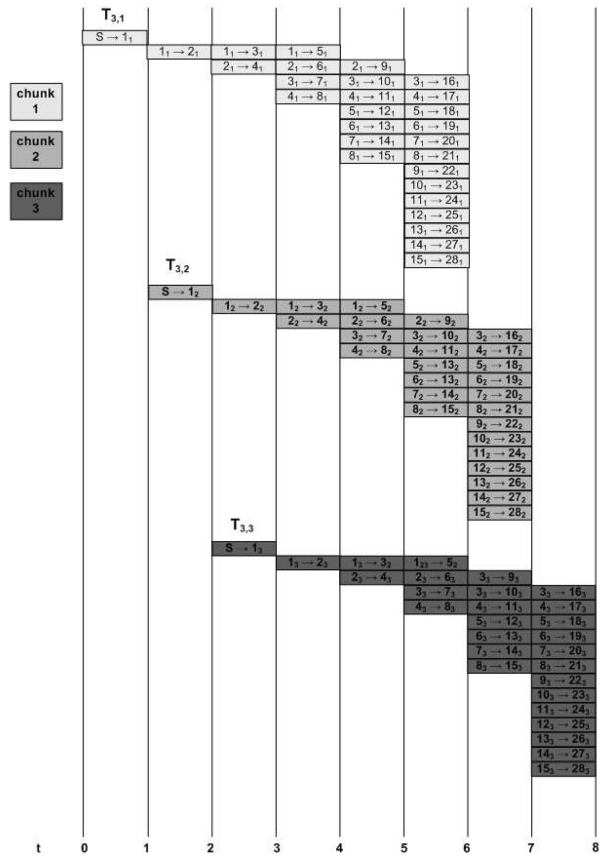

Fig. 3. Overlay trees $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$ of *Streamline* in a network of 28 peer nodes: distribution process of the first 3 chunks in the time interval $(0, 8)$

indexing of figure 2 would imply that we have already solved the problem.

From figure 3, 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 $9_i, 10_i, \ldots, 15_i$, $i = 1, 2, 3$, transmit for only 1 time unit. We refer to these nodes as "class 1" nodes.
- 13 nodes, namely $16_i, 17_i, \ldots, 28_i$, $i = 1, 2, 3$, 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 3. The general procedure to solve the problem can be found in [10].

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 1. 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 1 presents only one of them.

$T_{3,1}$		$T_{3,2}$		$T_{3,3}$	
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 1
Role played by the nodes in each distribution tree: solution of the tree intertwining problem in a network of 28 nodes.

4 Performance analysis

In this section we derive an analytical model to evaluate the performance of *Streamline*. The main performance indexes that we evaluate are:

- $N(c,k,U,t)$: the number of peer nodes that complete the download of the c-th chunk within t time units, being k the number of parents/children and U the ratio B/R between the uplink capacity U and the stream bit rate R;
- $T(c,k,U,n)$: the maximum amount of time needed by n peer nodes to complete the download of the c-th chunk, being k the number of parents/children and U the ratio B/R. Since chunk c arrives at the source node at time $(c-1)U$, the time needed to distribute the chunk since its generation (absolute delay) is readily computed as $T(c,k,U,n)-(c-1)U$.

Since the expressions that we are going to derive for these indexes are not in closed form, we provide also an asymptotic, closed form expression of such indexes (in subsection 4.2). This expression is not only very simple and convenient to use, but is also very accurate already for very small values of the relevant system parameters (see the numerical comparison in subsection 5.2).

As regards the performance index $N(c,k,U,t)$, we observe that: i) the source always transmits only one chunk to only one of its children at any one time; ii) the source starts transmitting the first chunk at time instant 0, serves it to U of its children in series and completes the transmission at time instant U, when the second chunk is available, and so on; iii) by generalizing the property in ii), the source starts transmitting the generic c-th chunk at time instant $(c-1)U$, it serves it to U of its children, and it ends serving the last child at time instant cU, when the $(c+1)$-th chunk is available; iv) the number of nodes that complete the download of the c-th chunk within t time units is the sum of the numbers of nodes that complete the download of that chunk exactly after $t, t-1, t-2$ time units, and so on, up to $t-(c-1)U-1$ time units. In fact, no node can complete the download of the c-th chunk before time instant $t=(c-1)U+1$, that is when the source ends transmitting that chunk to the first child.

To evaluate the performance metric $N(c,k,U,t)$, we let $n(c,k,U,i)$ be the number of nodes that complete the download of the c-th chunk exactly i time units after the generation of that chunk at the source. It results $N(c,k,U,t) = \sum_{i=1}^{t-(c-1)U} n(c,k,U,i)$. In order to evaluate $n(c,k,U,i)$, we distinguish among three cases:

1) if $1 \le i \le U$, the source is still serving the c-th chunk and $n(c,k,U,i) = n(c,k,U,i-1) + n(c,k,U,i-2) + \cdots + n(c,k,U,0)$, where $n(c,k,U,0)=1$, to take the children served by the source into account;
2) if $U < i \le k$, no node has finished serving its k children and $n(c,k,U,i) = n(c,k,U,i-1) + n(c,k,U,i-2) + \cdots + n(c,k,U,1)$;
3) if $i > k$, only the nodes that have completed the download of that chunk exactly $i-1, i-2, i-3, \cdots, i-k-1$ time units after the generation of that chunk have still children to be served; nodes that have completed the download of that chunk with a delay less than $i-k-1$ time units have already served all their k children; consequently $n(c,k,U,i) = n(c,k,U,i-1) + n(c,k,U,i-2) + \cdots + n(c,k,U,i-k-1)$.

It is not difficult at this point to derive the condition $n(c,k,U,i) = F_k(i) + F_k(i-1) + \cdots + F_k(i-U+1)$, where $F_k(\cdot)$ is the k-step Fibonacci sequence, defined as follows

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

By assuming that t is always an integer value, generalization to continuous time being straightforward, we can write:

$$N(c,k,U,t) = \sum_{i=1}^{t-(c-1)U} \sum_{j=1}^{U} F_k(i-j+1) \tag{2}$$

The performance index $N(c,k,U,t)$ may be re-written as a function of the sum of the first values of a k-step Fibonacci sequence

$$S_k(t) = \begin{cases} 0 & \text{if } t \le 0 \\ \sum_{i=1}^{t} F_k(i) & \text{if } t \ge 1 \end{cases} \tag{3}$$

By exploiting (3), formula (2) can be rewritten as

$$N(c,k,U,t) = \sum_{j=1}^{U} S_k(t-(c-1)U-j+1) \tag{4}$$

We turn now our attention to the evaluation of $T(c,k,U,n)$. Unfortunately it is not possible to derive analytically this performance index, because *Streamline* generates, by its nature, unbalanced distribution trees that impede to evaluate the number of nodes for each level in closed form. The only thing that we can do is to provide an implicit formulation for $T(c,k,U,n)$, which can be solved numerically by trial and error:

$$\begin{aligned} T(c,k,U,n) &= \min\{t \ge (c-1)U : \\ &\quad \sum_{j=1}^{U} S_k(t-(c-1)U-j+1) \ge n\} \end{aligned} \tag{5}$$

We conclude the Section by presenting an important result: we show that our algorithm is optimal for the considered topology. The algorithm is optimal in the sense that it allows reaching the greatest possible number of nodes in a given time interval or equivalently it allows reaching a given number of nodes in the smallest possible time interval.

Before presenting the theorem we introduce some notations. Let us denote a generic distribution algorithm with $D(\mathcal{A})$, where $\mathcal{A}$ is the set of uplink capacity allocation strategies of all nodes in the considered forest-based topology. We also let $N_D(c,k,U,t)$ be the number of peer nodes that complete the download of the c-th chunk within t time units, being k the number of parents/children, U the ratio B/R and $D = D(\mathcal{A})$ the distribution algorithm. We can now state the theorem.

Theorem 1: For every distribution algorithm $D = D(\mathcal{A})$, $N_D(c,k,U,t)$ is upper-bounded as follows:

$$N_D(c,k,U,t) \leq \sum_{j=1}^{U} S_k(t-(c-1)U-j+1) \quad \forall t \tag{6}$$

where $S_k(\cdot)$ is the sum of a k-step Fibonacci sequence introduced in (3).

A direct consequence of this Theorem is that *Streamline* is optimal in the sense given above among all distribution algorithms for the considered topology, since $N(c,k,U,t) = \sum_{j=1}^{U} S_k(t-(c-1)U-j+1)$. The proof of this Theorem is given in the following subsection 4.1.

4.1 Proof of Theorem 1

Proof: Let X be a generic non-source peer. Let us define the metric $\overline{N}_{D,X}(c,k,U,d)$ as the number of peer nodes which receive the c-th chunk either directly or indirectly (through a neighbor, or a neighbor of a neighbor, etc) from X within d time units since the time instant at which peer node X has received that chunk. Let us include in the count also peer X itself. By construction, i) $\overline{N}_{D,X}(c,k,U,d) = 0 \ \forall d < 0$, as peer X has not yet received the c-th chunk and hence it has not yet started to distribute it further, ii) $\overline{N}_{D,X}(c,k,U,0) = 1$, as the only peer which can receive the c-th chunk with a null delay is X itself, iii) $\overline{N}_{D,X}(c,k,U,d)$ is a monotone non decreasing function of d.

We first prove the following result

$$\overline{N}_{D,X}(c,k,U,d) \leq S_k(d+1) \quad \forall D, \forall X, \forall d \tag{7}$$

The proof is given by contradiction.

Let us assume that there exists a distribution algorithm D_0 and a peer node X_0 such that $\overline{N}_{D_0,X_0}(c,k,U,d) > S_k(d+1)$, starting from a time instant that we denote as d_0. Let $ch_{X_0,1}, ch_{X_0,2}, \ldots, ch_{X_0,k}$ be the children of peer node X_0. Let also $T_{1,D_0}, T_{2,D_0}, \ldots, T_{k,D_0}$ be the time intervals between the time instant at which peer node X_0 receives the c-th chunk and the time instants at which the children $ch_{X_0,1}, ch_{X_0,2}, \ldots, ch_{X_0,k}$ receive chunk c from node X_0. Subscript D_0 points out the dependence of such time instants on the considered distribution algorithm $D_0 = D_0(\mathcal{A}_0)$, which includes also the uplink capacity allocation strategy of peer node X_0. In addition, we observe that peer node X_0 does not necessarily transmit the c-th chunk to all its children. Due to this, it is possible that $T_{i,D_0} = \infty$ for some i. Without loss of generality, we assume $T_{1,D_0} \leq T_{2,D_0} \leq \ldots \leq T_{k,D_0}$.

By exploiting the above notation, we observe that

$$\overline{N}_{D_0,X_0}(c,k,U,d) = 1 + \sum_{i=1}^{k} \overline{N}_{D_0,X_0}(c,k,U,d-T_{i,D_0}) \tag{8}$$

where $\overline{N}_{D_0,X_0}(c,k,U,d-T_{i,D_0}) = 0$ if $T_{i,D_0} = \infty$.

We now observe that the minimum duration of each transmission is 1. This happens when the sending node assigns all the uplink capacity to only one receiver node. As a consequence, i) the minimum value that T_{1,D_0} may assume is 1, ii) the minimum value that T_{2,D_0} may assume is 2 and so on up to $T_{k,D_0} = k$. Thus, $T_{i,D_0} \geq i \ \forall i = 1,2,\ldots,k$.

As $\overline{N}_{D_0,X_0}(c,k,U,d)$ is by definition a monotonically non-decreasing function, the following condition holds

$$\overline{N}_{D_0,X_0}(c,k,U,d) \leq 1 + \sum_{i=1}^{k} \overline{N}_{D_0,X_0}(c,k,U,d-i) \tag{9}$$

Condition (9) has to hold also for $d = d_0$. As a consequence,

$$\begin{aligned} 1 + \sum_{i=1}^{k} \overline{N}_{D_0,X_0}(c,k,U,d_0-i) \geq \\ \geq \overline{N}_{D_0,X_0}(c,k,U,d_0) > S_k(d_0+1) \end{aligned} \tag{10}$$

where the latter inequality holds because of the contradiction assumption.

On the other side, $d_0 = \min\{d : \overline{N}_{D_0,X_0}(c,k,U,d) > S_k(d+1)\}$ (we recall that d_0 is the first value of time such that $\overline{N}_{D_0,X_0}(c,k,U,d) > S_k(d+1)$). Thus,

$$\begin{aligned} \overline{N}_{D_0,X_0}(c,k,U,d_0-1) &\leq S_k(d_0) \\ \overline{N}_{D_0,X_0}(c,k,U,d_0-2) &\leq S_k(d_0-1) \\ &\ldots \\ &\ldots \\ \overline{N}_{D_0,X_0}(c,k,U,d_0-k) &\leq S_k(d_0-k+1) \end{aligned} \tag{11}$$

Conditions (11) also imply that

$$1 + \sum_{i=1}^{k} \overline{N}_{D_0,X_0}(c,k,U,d_0-i) \leq 1 + \sum_{i=1}^{k} S_k(d_0-i+1) \tag{12}$$

Since $1 + \sum_{i=1}^{k} S_k(d_0-i+1) = S_k(d_0+1)$ for Lemma 1 (see Appendix A), condition (12) contradicts condition (10), and this is absurd. So we can conclude that $\overline{N}_{D,X}(c,k,U,d) \leq S_k(d+1) \ \forall D, \forall X, \forall d$.

We now observe that the performance metric $N(c,k,U,t)$ differs from the performance metric $\overline{N}_{D,X}(c,k,U,d)$ because it represents the number of nodes which have received the c-th chunk starting from the source S, rather than a generic node X, and not including the source in the count. We also observe that in the considered forest-based topology the source serves each chunk to U out of its k children. We denote the U children receiving chunk c from the source as $ch_{s,1}, ch_{s,2}, \cdots, ch_{s,U}$. Let $T_{s1,D}, T_{s2,D}, \ldots, T_{sU,D}$ be the time intervals between the time instants at which the children $ch_{s,1}, ch_{s,2}, \cdots, ch_{s,U}$ receive chunk c from the source and the time instant $(c-1)U$. Without loss of generality, we assume $T_{s1,D} \leq T_{s2,D} \leq \ldots \leq T_{sU,D}$. Thus, it results

$$N(c,k,U,t) = \sum_{j=1}^{U} \overline{N}_{D,ch_{s,j}}(c,k,U,t-(c-1)U-T_{sj,D}) \tag{13}$$

As previously said, the minimum value that each transmission duration may assume is 1. As a consequence, i) the minimum value that $T_{s1,D}$ may assume is 1, ii) the minimum value that $T_{s2,D}$ may assume is 2 and so on up to $T_{sU,D} = U$. Thus, $T_{si,D} \geq i \ \forall i = 1,2,\ldots,U$ and the following inequality chain holds:

$$N(c,k,U,t) \leq \sum_{j=1}^{U} \overline{N}_{D,ch_{s,j}}(c,k,U,t-(c-1)U-j) \leq \sum_{j=1}^{U} S_k(t-(c-1)U-j+1) \quad \forall t, \forall D \tag{14}$$

which proves Theorem 1. □

4.2 Asymptotic approximation of $N(c,k,U,t)$ and $T(c,k,U,n)$

In this subsection we derive an asymptotic, closed form approximation of $N(c,k,U,t)$ and $T(c,k,U,n)$ with respect to t and n, respectively. This is done by exploiting the asymptotic expression (30) of k-step Fibonacci Sums, derived in Appendix A, Lemma 4. It results:

$$\begin{aligned} N(c,k,U,t) &= \sum_{j=1}^{U} S_k(t-(c-1)U-j+1) \approx \\ &\approx \sum_{j=1}^{U} \frac{\phi_k \cdot \phi_k^{t-(c-1)U-j+1}}{(\phi_k-1)Q_k(\phi_k)} - \sum_{j=1}^{U} \frac{1}{k-1} = \\ &= \frac{\phi_k^2(1-\phi_k^{-U})\phi_k^{-(c-1)U}}{Q_k(\phi_k)(\phi_k-1)^2} \cdot \phi_k^t - \frac{U}{k-1} \end{aligned} \tag{15}$$

$$\begin{aligned} T(c,k,U,n) \approx\ & (c-1)U+ \\ &+ \left\lceil \log_{\phi_k} \left[\frac{(\phi_k-1)^2 Q_k(\phi_k)}{\phi_k^2 \left(1-\phi_k^{-U}\right)} \cdot \left(n + \frac{U}{k-1}\right) \right] \right\rceil \end{aligned} \tag{16}$$

The numerical values for the Fibonacci constants ϕ_k and the parameters $Q_k(\phi_k)$ are provided in Appendix A.

An interesting observation is that the derived performance metrics explicitly account for the fact that each node can feed at most k neighbors. If this restriction is removed (i.e., if each node can reach whatever other node), then, by using expression (31) provided in Appendix A, Lemma 5, we obtain a more simple and immediate expression for $N(c,\infty,U,t)$

$$\begin{aligned} N(c,\infty,U,t) &= \sum_{j=1}^{U} S_\infty(t-(c-1)U-j+1) = \\ &= \sum_{j=1}^{U} 2^{t-(c-1)U-j} = 2^{t-(c-1)U}(1-2^{-U}) \end{aligned} \tag{17}$$

A similar result can be derived also for $T(c,\infty,U,n)$. We finally observe that relation (17) has already been derived in [13], where the author limits itself to investigate full-mesh overlay topologies and the case $U = 1$.

5 PERFORMANCE RESULTS

In this section we provide some performance results of *Streamline*. We use the exact formulas (4) and (6) in subsection 5.1. We use the approximated, asymptotic formulas (15) and (16) in subsection 5.2, also to compare the exact solution of the performance indexes of *Streamline* to their respective asymptotic approximations.

In principle, we would not need to compare our algorithm to other solutions, having formally proven its optimality; however, we want to give a feeling of the quantitative improvements brought about by the proposed serial transmission. For this reason, *Streamline* is analytically compared in subsection 5.1 to an ideal parallel transmission (conceived by ourselves for this purpose), which differs from *Streamline* only in the fact of employing parallel transmission instead of serial transmission. In addition, we perform in subsection 5.3 a comparison between *Streamline* and a reference literature solution, *SplitStream*, by using a suitably developed simulation package, based on our OPSS [8].

5.1 Exact evaluation of $N(c,k,U,t)$ and $T(c,k,U,n)$

Before presenting the results, we note that the values of the performance indexes $N(c,k,U,t)$ and $T(c,k,U,n)$ evaluated for chunk c are equal to those evaluated for chunk $c+D$, if we shift the time by $D \cdot U$ time units[1]. Thus, in principle we would not need to present results for different values of c. However, we will do that, for the convenience of the reader. This fact has also the important consequence that the performance of our approach is equally good both at the start of the transmission, when there are a few chunks, and as time passes.

Figure 4 shows the time $T(c,k,U,n)$ needed by n peer nodes to complete the download of the 100-th chunk for four different combinations of k and U values. In order to make a fair comparison, we assume that the bit rate of the stream (R) is kept constant, while the available uplink capacity of peer nodes (B) changes so as to make U take the two values considered in the figure. In other words, we consider the case in which the same stream is distributed in two different networks, where peer nodes have two different values of uplink capacity B. Since we normalize all performance figures with respect to the time needed to download a complete chunk at rate B, different values of the available uplink capacity B imply different time scales. This would make the comparison of the performance indexes in figure 4 very difficult. To solve the problem, we normalize all performance indexes in figure 4 with respect to the time needed to download a complete chunk in the case $U = 1$. The figure shows that: i) given a ratio U between the available uplink capacity B and the stream bit rate R, the greater the number k of parents/children, the lower the time needed to stream the same content to the same number of nodes, ii) given a number k of parents/children, the greater the available uplink capacity, the better the performance. For instance, to stream the 100-th chunk across 10000 nodes in the case $U = 1$, *Streamline* with $k = 2$ requires 118 time units, while *Streamline* with $k = 6$ needs 114 time units. Likewise, to stream the 100-th chunk across 10000 nodes in the case $k = 6$, *Streamline* with $U = 1$ requires 118 time units, while *Streamline* with $U = 3$ needs about 103.67 time units. Out of curiosity, and without any claims of performing a comparison with other approaches, we present in this figure also the performance of an ideal reference scheme, which differs from *Streamline* only in the fact of employing parallel transmission instead of serial transmission. As said in the Introduction, we would not need to compare our algorithm

1. Actually this holds for the performance index $N(c,k,U,t)$. However, it is possible to evaluate $T(c+D,k,U,n)$ starting from $T(c,k,U,n)$, by adding $D \cdot U$.

to other solutions, as we formally prove the optimality of our approach. However, we present this curve only to show the consequence of adopting a parallel transmission in place of a serial transmission, all other system parameters being equal.

Figure 5 shows the number of peer nodes $N(c, k, U, t)$ that can complete the download of the 1-st chunk as a function of the time t and for four different combinations of k and U values. This figure is affected by the problem of different time scales as well. We adopted the same solution as in figure 4. The figure shows that i) given a ratio U between the available uplink capacity B and the stream bit rate R, the greater the number k of parents/children, the higher the number of served peer nodes in the same time, ii) given a number k of parents/children, the greater the available uplink capacity, the better the performance. For instance, the number of peer nodes that can complete the download of the 1-st chunk at time unit 50 in the case $U = 2$ is 1.52×10^{28} for $k = 4$ and 2.24×10^{29} for $k = 6$. Likewise, the number of peer nodes that can complete the download of the 1-st chunk at time unit 50 in the case $k = 4$ is 1.09×10^{14} for $U = 1$ and 1.52×10^{28} for $U = 2$.

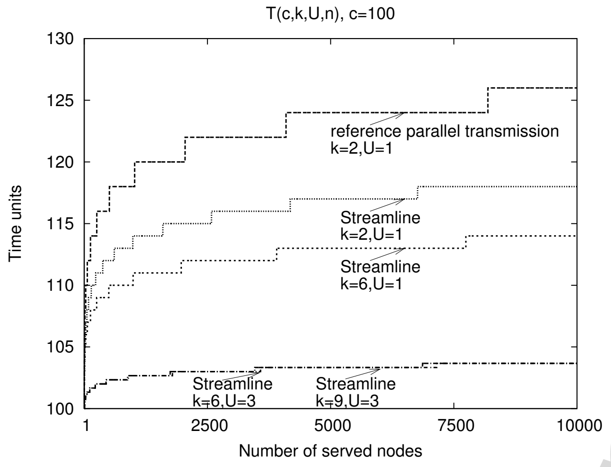

Fig. 4. Time units, $T(c, k, U, n)$, needed by n peer nodes to complete the download of the 100-th chunk for four different pairs of the values of the number k of parents/children and the ratio $U = B/R$ in *Streamline*.

It is also important to note that both performance indexes of *Streamline* improve monotonically as a function of the number k of parents/children. However, we verified that the performance improvement becomes negligible as k gets greater than 4 or 5. This is readily justified as the performance depends on the number of deployed trees through the Fibonacci constant values ϕ_k (see Section 4); as a matter of fact, Fibonacci constants ϕ_k very rapidly tend to the limit value 2 [15] (e.g. ϕ_4 is already 1.96595). From a practical point of view, this is a useful property, as the number of trees and their dynamic management may introduce signalling overhead which may turn to be a practical limiting factor on k.

5.2 Comparison between exact and asymptotic formulas

In this subsection we compare the exact solution of the performance indexes of *Streamline* to their asymptotic, closed form approximations.

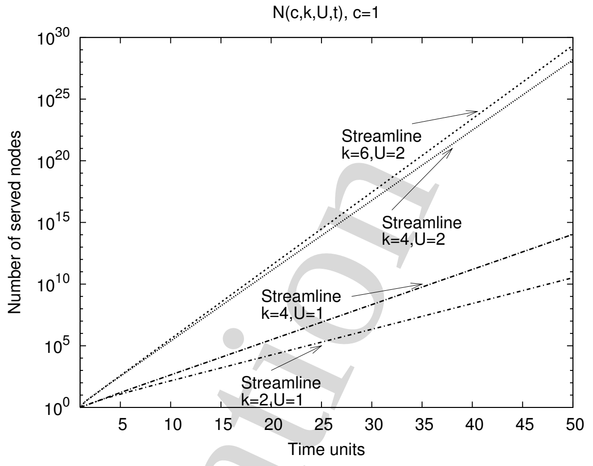

Fig. 5. Number of peer nodes, $N(c, k, U, t)$, that can complete the download of the 1-st chunk as a function of the time t for four different pairs of values of the number k of parents/children and the ratio $U = B/R$ in *Streamline*.

Figure 6 shows the time $T(c, k, U, n)$ needed by n peer nodes to complete the download of the 100-th chunk in the case $U = 1$, 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 accurate already for very small values of t.

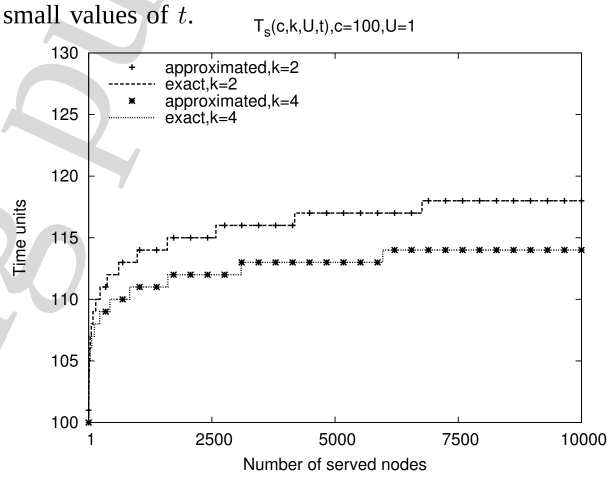

Fig. 6. Asymptotic and exact evaluation of the time units, $T(c, k, U, n)$, needed by n peer nodes to complete the download of the 100-th chunk in the case $U = 1$ as a function of the number n of peer nodes, and for two values of k.

The same behavior can be observed in figure 7, which shows the number of peer nodes $N(c, k, U, t)$ that can complete the download of the 1-st chunk as a function of the time t and for two pairs of values of the number k of parents/children and ratio U.

As a matter of fact the exact and approximated curves can not be distinguished in both the above figures.

5.3 Comparison between Streamline and a literature proposal

We compare our solution to *SplitStream* [3], since it is maybe the most representative example of algorithms exploiting

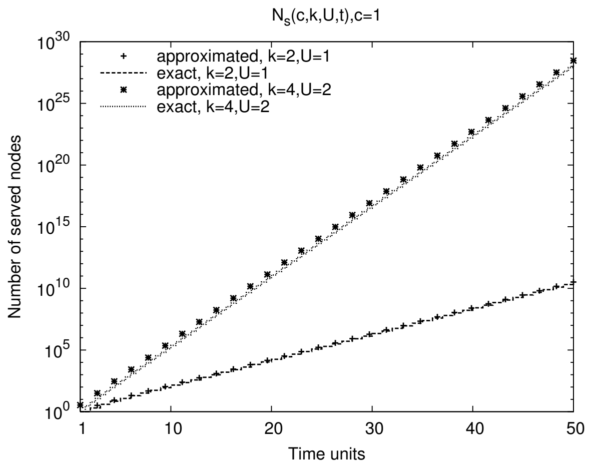

Fig. 7. Asymptotic and exact evaluation of the number of peer nodes, $N(c,k,U,t)$, that can complete the download of the 1-st chunk as a function of the time t, and for two combinations of k and U values.

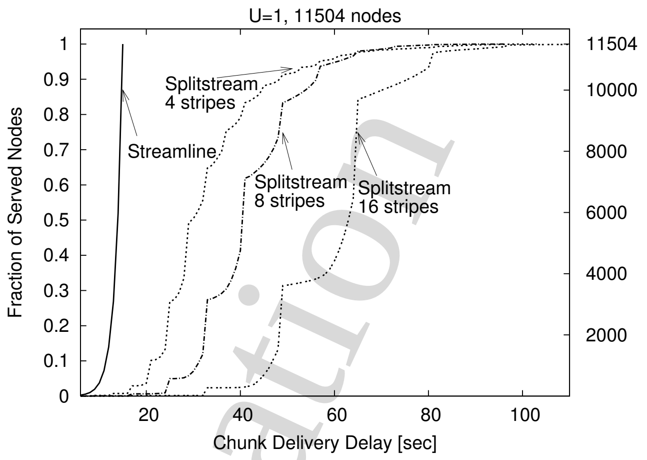

Fig. 8. Cumulative distribution function of $\overline{N}_{served}(c,d)$ in *Streamline* and *SplitStream* in case of a network with 11504 nodes, ratio between uplink capacity and stream bit rate equal to 1, for three different values of the number of stripes.

forest-based topologies, like our *Streamline*. In *SplitStream* the stream is stripped into *stripes* to be distributed across a forest of overlay multicast trees. The construction of the overlay multicast trees is based on Scribe [11]. Scribe is an application-level group communication system built on top of Pastry [12], 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[2] for that group Pastry key.

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 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 [8].

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 in this paper 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. 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 $\overline{N}_{served}(c,d)$.

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

In figure 8 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 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.

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

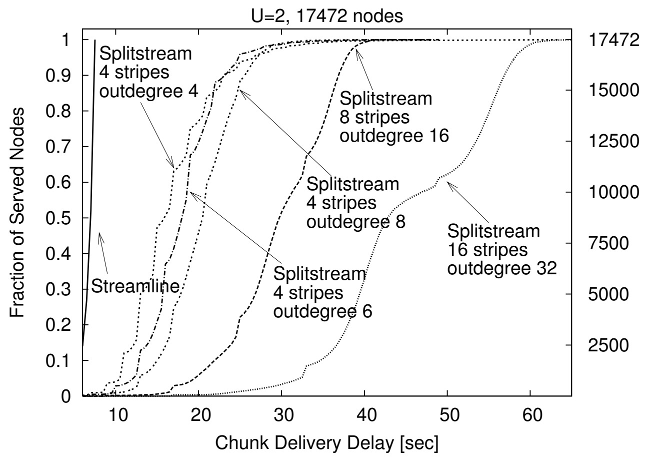

Fig. 9. Cumulative distribution function of $\overline{N}_{served}(c, d)$ in *Streamline* and *SplitStream* in case of a network with 17472 nodes, ratio between uplink capacity and stream bit rate equal to 2, for different values of the number of stripes and the outbound degree.

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 observed in subsection 5.1 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.

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

In figure 9 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.

6 O-Streamline: from theory to practice

O-Streamline puts into practice the two basic principles of *Streamline*: i) the organization of peers and chunks in a finite number of overlay trees and groups and ii) the serialization of chunk transmissions. Here we present very briefly *O-Streamline*; we refer the reader to [9] for more details.

The second principle is easy to be put into practice. As regards the first principle, *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 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 R, C, B, T, U be the same parameters as in *Streamline*.

The scheduling algorithm operates as follows:

- the source, 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 serves the chunks to the neighbors of each group following the order implicitly established in the association between chunk identifiers and groups. The source repeats 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 $G \times U \times \frac{C}{B}$ 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 querying the source node at bootstrap time, asking for the identifier of the latest produced chunk;
- the generic peer serves each chunk in the FIFO queue in series to $G \times U$ neighbors, by giving priority to the neighbors of its own group (if there are $G \times U$ neighbors missing that chunk). This implies that the number of neighbors P has to be at least equal to $G \times U$;
- 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.

We observe 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 addition, we observe that the value of P must be greater than $G \times U$, for the algorithm to handle the churn. In fact, peer nodes may loose a neighbor, due to a disconnection, and have less neighbors than $G \times U$ in their own group; thus, such nodes may look for another neighbor and maintain $G \times U$ as the minimum number of connections in that group.

To give an idea of the performance of *O-Streamline*, we now present some simulation results, obtained with our OPSS simulator [8], which refer to the case of a network with 17472 nodes and a ratio between uplink capacity and stream bit rate equal to $U = 2$. For a more comprehensive performance evaluation, we refer the reader to [9].

We consider the same performance index $\overline{N}_{served}(c, d)$ as the one considered in subsection 5.3 to evaluate the performance of *SplitStream*. As already said, the advantahe of such performance index lies in its high closeness to the performance index $N(c, k, U, t)$ used to evaluate the performance of *Streamline*.

In figure 10 we plot the cumulative distribution function of $\overline{N}_{served}(c, d)$ in the case of $G = 2$ groups, for three different values of the number of neighbors per group, under the assumption $P = O$. The curve relative to *Streamline* has been analytically derived by setting $k = 4$. We observe that as the number of neighbors per group increases, the performance of *O-Streamline* get closer to the ones of *Streamline*.

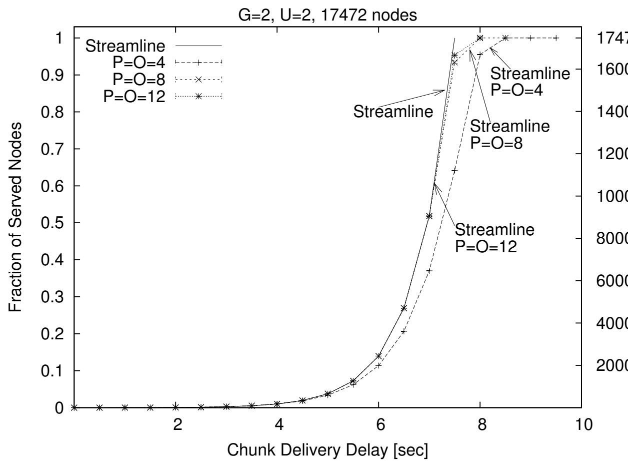

Fig. 10. Cumulative distribution function of $\overline{N}_{served}(c, d)$ in *Streamline* and *O-Streamline* in case of a network with 17472 nodes, $U = 2$ and $G = 2$ groups, for three different values of the number of neighbors per group.

Figure 11 refers to the same scenario as in figure 10, with the only difference that nodes disconnect due to churn. We assume that the session times are exponentially distributed with an average value of 10 minutes. We plot the cumulative distribution function of $\overline{N}_{served}(c, d)$ in the case of $P = O = 8$ for three different values of the number G of groups. 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.

7 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.

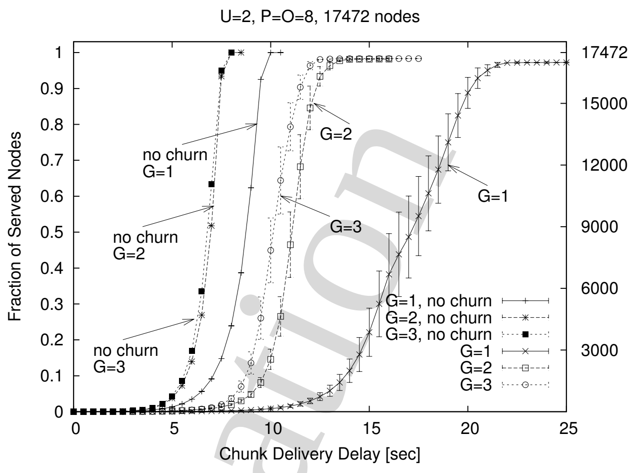

Fig. 11. Cumulative distribution function of $\overline{N}_{served}(c, d)$ in *O-Streamline* in case of a network with 17472 nodes, $U = 2$ and $P = O = 8$ neighbors per group, for three different values of the number of groups.

In addition, the few available theoretical works mostly focus on systems where the streaming information, optionally organized in sub-streams, is continuously delivered across overlay paths. In such case, a fluidic approach is typically used to evaluate performance and, as long as some feasibility conditions are met (see e.g. [14]), 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. As a consequence, the delay performance optimization becomes a minimum path cost problem. 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 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 [17], where it is proven that: i) if we assume no restriction on the number of children a node may upload to, a tree depth equal to two is always sufficient, ii) if we assume that a node may upload to at most M children, the minimum tree depth is $\lceil \log_M((M - 1)n + 1) \rceil$, being n the total number of nodes to be served. Other studies address the issue of how to maximize throughput by using various techniques, such as network coding [18] or pull-based streaming protocol [19].

This work differs from the previously cited ones mainly because it focus on chunk-based systems. Chunk-based systems have a key difference with respect to sub-stream-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 the delay performance are mostly affected by the chunk transmission time and by the uplink capacity available at each link. Besides, we can use discrete-time approaches instead of fluidic approaches. Surprisingly enough, according to the best of our knowledge and our literature survey, there is only one work [13] where chunk-based systems are theoretically analyzed.

In more detail, the author of [13] derives a minimum delay bound for P2P video streaming systems, and proposes the so called *snow-ball* streaming algorithm to achieve such bound. However, such bound has been derived under the full-mesh overlay topology assumption, and it is the same that we found as a particular case in our analysis when $k \to \infty$. Differently from [13], we consider a forest-based topology, and we show that we can achieve performance very close to the ones of the full-mesh case, even with a limited overlay connectivity among nodes.

8 Conclusions

In this paper we proposed a new distribution algorithm whose performance coincides with the theoretical upper-bound on the number of nodes reached versus time achievable for a forest topology (and hence, dually, on the time required to deliver a stream); our algorithm is optimal in that sense. The proposed analytical model allows evaluating the system performance quite easily. The asymptotic approximation provides a closed formula that is useful to better understand the phenomena at hand and yet is sufficiently accurate to be used to assess the performance in real cases of interest. A side result of this work is the novel derivation of explicit, asymptotic, and recursive expressions for the sum of the first t values of a k-step Fibonacci sequence.

References

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

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

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, *Splitstream: High-bandwidth multicast in cooperative environments*, in Proc. of the 19th ACM Symposium on Operating Systems Principles, The Sagamore, NY, USA, 2003.

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

[5] M. Zhang, Y. Xiong, Q. Zhang and S. Yang, *On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks*, in Proc. of IEEE GLOBECOM, 2006.

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

[7] 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.

[8] L. Bracciale, F. Lo Piccolo, D. Luzzi and S. Salsano *OPSS: an Overlay Peer-to-peer Streaming Simulator for large-scale networks*, in ACM SIGMETRICS Performance Evaluation Review, Volume 35, Issue 3, 2007.

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

[10] G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, S. Salsano, *Streamline: an Optimal Distribution Algorithm for Peer-to-Peer Real-time Streaming*, Technical Report, 2009 (on line available at netgroup.uniroma2.it/p2p/streamline-tech-rep.pdf).

[11] 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.

[12] A. Rowstron, P. Druschel, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, in Proc. of IFIP/ACM Middleware, 2001.

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

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

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

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

[17] 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.

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

[19] 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.

Appendix A
Some results on k-step Fibonacci Sums

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, earlier defined by (3). 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 (1). Then, $S_k(n)$ may be recursively computed as:

$$S_k(n) = 1 + \sum_{i=1}^{k} S_k(n-i) \quad \forall n \geq 1 \tag{18}$$

The proof is based on the mathematical induction. Condition (18) is immediately verified for $n = 1$. Hence, let us assume that condition (18) holds for all indices up to n. By applying such condition to $S_k(n)$ and by using the k-step Fibonacci series definition (1), it is straightforward to prove that (18) holds also for $n + 1$:

$$\begin{gathered} S_k(n+1) = S_k(n) + F_k(n+1) = \\ = 1 + \sum_{i=1}^{k} S_k(n-i) + \sum_{i=1}^{k} F_k(n+1-i) = \\ = 1 + \sum_{i=1}^{k} [S_k(n-i) + F_k(n-i+1)] = 1 + \sum_{i=1}^{k} S_k(n+1-i) \end{gathered} \tag{19}$$

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

$$S_k(n) = \frac{\sum_{i=1}^{k}(i+1-k)F_k(n+i)}{k-1} - \frac{1}{k-1} \tag{20}$$

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 (20) (achievable for $k = 2$).

The proof requires some algebraic elaboration. We start by reformulating the linear recurrence (1) as the following difference equation:

$$F_k(i) + \sum_{j=1}^{k-1} F_k(i+j) - F_k(i+k) = 0 \quad \forall i \geq 1 \tag{21}$$

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

$$\sum_{i=1}^{n}\left\{F_k(i)+\sum_{j=1}^{k-1}F_k(i+j)-F_k(i+k)\right\}=0 \quad \forall n \geq 1 \tag{22}$$

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:

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

Using the last elaboration and then solving equation (22), we achieve

$$S_k(n)=\frac{\sum_{i=1}^{k}(i+1-k)F_k(n+i)}{k-1}-\frac{\sum_{i=1}^{k}(i+1-k)F_k(i)}{k-1} \tag{23}$$

Equation (20) 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 [16] 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 [15] as

$$F_k(n)=\sum_{j=1}^{k}\frac{\phi_{k,j}^{n}}{Q_k(\phi_{k,j})} \tag{24}$$

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

$$P_k(x)=x^k-x^{k-1}-x^{k-2}-\cdots-x-1=\frac{x^{k+1}-2x^k+1}{x-1} \tag{25}$$

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

$$\begin{array}{rcl} Q_2(x) & = & -1+2x \\ Q_3(x) & = & -1+4x-1x^2 \\ Q_4(x) & = & -1+6x+0x^2-1x^3 \\ \vdots & \vdots & \vdots \qquad \vdots \qquad \vdots \\ Q_k(x) & = & -1+2(k-1)x+\sum_{i=2}^{k-1}(k-i-2)x^i \end{array} \tag{26}$$

Thanks to the key relation provided in Lemma 2, we can now substitute the exact expression of $F_k(\cdot)$ (24) in (20), thus obtaining:

$$S_k(n)=\frac{\sum_{i=1}^{k}(i+1-k)\sum_{j=1}^{k}\frac{\phi_{k,j}^{n+i}}{Q_k(\phi_{k,j})}}{k-1}-\frac{1}{k-1}=$$
$$=\sum_{j=1}^{k}\frac{\phi_{k,j}^{n}}{(k-1)Q_k(\phi_{k,j})}\sum_{i=1}^{k}(i+1-k)\phi_{k,j}^{i}-\frac{1}{k-1} \tag{27}$$

Now,

$$\sum_{i=1}^{k}(i+1-k)\phi_{k,j}^{i}=\frac{\phi_{k,j}}{\phi_{k,j}-1}\left[k-1+\frac{1-2\phi_{k,j}^{k}+\phi_{k,j}^{k+1}}{\phi_{k,j}-1}\right] \tag{28}$$

The last fraction in (28) vanishes, as this is the characteristic polynomial (25) computed for one of its roots. Hence, expression (27) simplifies to the final expression:

$$S_k(n)=\sum_{j=1}^{k}\frac{\phi_{k,j}}{(\phi_{k,j}-1)Q_k(\phi_{k,j})}\phi_{k,j}^{n}-\frac{1}{k-1} \tag{29}$$

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 (25). However, such roots are known to satisfy an important property [16]: 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 (24) and (29) rapidly becomes negligible as the index n grows. As a consequence, the following approximate expression holds:

$$S_k(n)\approx\frac{\phi_k}{(\phi_k-1)Q_k(\phi_k)}\phi_k^{n}-\frac{1}{k-1} \tag{30}$$

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, the first few values of the terms $Q_k(\phi_k)$ are $Q_2(\phi_2)=2.23607, Q_3(\phi_3)=2.97417, Q_4(\phi_4)=3.40352, Q_5(\phi_5)=3.65468, Q_6(\phi_6)=3.80162$.

Lemma 5: *Derivation of* $S_\infty(n)$. A possibility would be to obtain this as the limit of expression (30) 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

$$S_\infty(n)=\sum_{i=1}^{n}F_\infty(i)=1+\sum_{i=2}^{n}2^{n-2}=2^{n-1} \tag{31}$$

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 (30) 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.