Incentive, Resilience and Load Balancing in Multicasting through Clustered de Bruijn Overlay Network (PrefixStream)

Anh-Tuan Gai, Laurent Viennot

Incentive, Resilience and Load Balancing in Multicasting through Clustered de Bruijn Overlay Network

Anh-Tuan Gai*
INRIA Rocquencourt
anh-tuan.gai@inria.fr

Laurent Viennot†
INRIA Rocquencourt
laurent.viennot@inria.fr

Abstract: In this paper, we consider the problem of multicasting a stream of packets in a large scale peer-to-peer environment. In that context peers should have incentive to cooperate. We present PrefixStream, an algorithm that addresses this problem by using reciprocity in packet forwarding. Each node thus has incentive to forward since recipients send back other packets of the stream. To achieve this efficiently, PrefixStream strips the content across two sets of clustered trees built upon the symmetric de Bruijn graph. This both allows to banish nodes that do not respect reciprocity of exchanges and gives resilience to node failures. Furthermore, it reduces the forwarding load of every node to the stream bandwidth (every node uploads as much as it downloads) even when the size of its cluster varies. Conversely to previously proposed hierarchical schemes, PrefixStream promotes disjoint clustering. This enables loose maintenance and network latencies optimization. We sketch the design of PrefixStream and analyze its performances.

Key words: peer-to-peer, multicast, incentive, streaming, de Bruijn.

1 Introduction

This paper proposes an algorithm called PrefixStream for multicast streaming in a very large scale peer-to-peer network. We call multicast streaming an application where a source is sending a flow of packets to a large number of receivers. IP multicasting is certainly the most efficient way for multicast, however the burden of duplicating packets is carried by intermediate routers which are often independent from the source and the receivers. This may explain why transit networks hardly ever support IP multicasting. On the other hand, attention is now centered on end-system or application-level multicast where the participants duplicate themselves the packets [4, 19, 15].

We concentrate on a system where the scarcest resource is the forwarding capacity of nodes. We typically target streaming applications for ADSL users. To function properly, such a system requires at least an average upload bandwidth per node greater than the stream bandwidth. PrefixStream allows all users to receive the stream under the condition that each user devotes to the system an upload capacity equal to the stream bandwidth.

Most existing solutions assume varying upload capacities of nodes. However, we argue that in a selfish context, all nodes should bear the same forwarding load. The reason is twofold. Selfish nodes tend to spend the minimum capacity required to obtain the service. Such behavior has been observed in file sharing applications where free riders [2, 18] tend to upload only if they cannot download otherwise. In such realistic selfish context, all nodes end up devoting same minimal bandwidth to the system, i.e. the stream bandwidth. Secondly, it may be the case that most of the nodes have an upload capacity close to the stream bandwidth since applications will tend to give the best quality available to the majority of users. (In an heterogeneous system, the classical approach is to form classes of users with equivalent capacity. Each class then certainly needs balanced forwarding loads.) Let us first discuss the three major design goals achieved by PrefixStream.

Low delay with balanced forwarding loads. First of all, minimizing the maximal delay is a classical concern in multicasting especially for live streaming. For that purpose, it is necessary to forward each packet of the stream along a tree rooted at the source. This allows to reach all nodes with a logarithmic number of retransmissions. Notice that interior nodes of the tree forward more data than they receive and that leaves do not forward at all. In order to balance the forwarding load of nodes, multiple disjoint trees must be used where each participant works more in some trees and less in others. Such solutions were first introduced to efficiently multicast in the hypercube [9] through edge disjoint trees. However, application-level multicast rather requires interior-node disjoint trees [4] where each node is interior in only one tree (being a leaf in other trees).

Efficient maintenance of the topology. As nodes may frequently enter and leave the system, maintaining this set of trees may become a challenging task. PrefixStream partitions the nodes in clusters and build trees among these clusters. Multicasting trees use one node per cluster and are easily maintained as long as each cluster contains a living node. A simple distributed algorithm is used to form clusters: nodes gather together according to the prefix of their identifier. The de Bruijn graph over prefixes allows

*Domaine de Voluceau, B.P.105, 78153 Le Chesnay cedex, France.
Research supported by the PairAPair project, ACI Masse de données.

to construct interior-node disjoint trees of regular degree. The drawback of this resilient scheme is longer propagation delays in large clusters.

Incentive to cooperate. Requiring balanced forwarding loads provides fairness. Nevertheless, it does not necessarily ensure that nodes behave as instructed. Nodes that do not fulfill their forwarding load should be detected and replaced by some more reliable nodes. To allow this, PrefixStream makes symmetric exchanges between nodes. This is possible through the use of two sets of trees where an edge of a tree in one set also appears with reversed direction in some other tree of the second set. If a node does not forward a packet as expected, the recipient may refuse to serve it in return. We use similar reciprocity inside clusters since each node regularly sends packets to all other members of its cluster. This exchange reciprocity thus enables tit-for-tat incentive mechanisms as introduced by BitTorrent [5]. Notice that the delay constraints and the packet level exchange of the streaming context prevent from using directly BitTorrent.

Our contributions. The de Bruijn topology was already known as a good candidate for distributed hashtables [14, 7, 10, 1, 8]. We show that it is also well suited for multicasting. First of all, the de Bruijn graph naturally contains interior-node disjoint trees of regular degree. This structure ensures both load balancing and low propagation delays. Moreover, the use of the de Bruijn graph and its symmetric allows to introduce reciprocity in exchanges. This brings the first peer-to-peer streaming algorithm with incentive to cooperate. On the other hand, we introduce a new way of clustering nodes which is naturally distributed. This enables low topology maintenance and resilience to churn.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 outlines the PrefixStream design. We analyze PrefixStream performances in Section 4.

2 Related work

Many application-level multicast have been proposed recently [4, 19, 15] (see [16] for an overview). Almost all of them rely on the hypothesis that nodes behave as instructed.

CoopNet [15] uses a centralized server to build multiple trees respecting degree constraints (the degree of a node is constrained by its upload capacity). On the other hand, Zigzag [19] uses a single tree with hierarchical clusters. Pulse [16] is an ongoing work to adapt BitTorrent algorithm to streaming. It is inspired both from the unstructured topology of BitTorrent and from its tit-for-tat exchange mechanism for giving incentive to cooperate. It exchanges small chunks of the stream within a time sliding window. However, it is still unclear whether BitTorrent may scale to make exchanges at very small chunk level. Indeed, the control overhead for announcing chunks bitmaps to neighbors may become comparable to the stream bandwidth. Bigger chunks incur less overhead but more delay. Similarly, PrefixStream uses reciprocity of exchanges both inside clusters and between clusters. However, it relies on a structured topology ensuring low propagation delays and low control overhead.

PrefixStream mostly resemble SplitStream [4]. SplitStream strives to construct interior-node disjoint trees based on the Pastry [17] overlay network. Each tree is basically constructed as the union of all routes from nodes to a given source. SplitStream additionally relies on an incremental insertion procedure to respect degree constraints of nodes and to cope with node departures. This mechanism is efficient when many nodes have spare capacity. However, when node capacities are limited to the stream bandwidth, this procedure generally rebuild a full branch of the tree where the node is inserted. Besides a complex reparation process, the main drawback of using this balancing mechanism is to make some nodes interior in several trees, moving away the topology from its primary design goal. A single node departure can then affect several trees. The need for this balancing mechanism is inherently due to the hypercube like topology used. We argue that the basic tree construction scheme of SplitStream produces unbalanced trees where interior nodes close to the root have higher degree than interior nodes closer to the bottom of the tree. For example, without constraint, the degree of the root can be as much as $(d-1)\log_d n$ with n nodes and d trees when it should be d to have a balanced system. See Annex A for a sketch of proof. The hypercube is indeed better suited for constructing edge disjoint trees [9]. On the other hand, the de Bruijn topology generalizes the heap binary-tree structure and naturally builds regular degree trees. PrefixStream brings the following improvements: it always succeeds in building interior-node-disjoint trees, clustering provides low topology maintenance under churn, and reciprocity of exchanges enable tit-for-tat incentive mechanisms.

3 PrefixStream Design

As in [15, 4], the content is split into stripes multicasted along separate trees. Any efficient encoding such as Multiple Description Encoding (MDC) or a more classical erasure coding can be used along these stripes. The main novelty of PrefixStream resides in its topology design which is basically a de Bruijn graph over clusters of nodes.

3.1 De Bruijn Topology

Definition 1 *A base d left shifting de Bruijn graph of p digits is a directed graph with d^p vertices. Each vertex is a sequence of p digits in base d. An arc from u to v is present when v may be obtained by left shifting u by one position and adding a new digit on the right. The right shifting de Bruijn graph is obtained with the backward edges of the left shifting de Bruijn graph: an arc from u to v is present when v may be obtained by right shifting u and adding a new digit on the left. We call de Bruijn graph the symmetric graph obtained with both left and right right shifting edges.*

PrefixStream is built upon the de Bruijn graph because it contains two symmetrical sets of interior-node-disjoint trees. One set is obtained by left shifting, the other by right shifting. A left tree is obtained by following left shifting edges from a root $r = x \cdots x$ with all digits equal to x. The sons of the root have the form $x \cdots xy$, its grand sons have the form $x \cdots xyz$ and so on. Notice that all interior vertices have x as first digit. Using d different digits x thus yields d interior-node-disjoint left trees. The root of each tree has degree $d-1$ and all other interior nodes have degree d. Interior-node-disjoint right trees can be obtained similarly. Moreover, the two sets are symmetrical: the father of a node in a tree of one set is a son in a tree of the other set.

Notice that for $d = 2$, the left shifting tree with root $0 \cdots 0$ is simply the classical heap structure where node i has sons $2i$ and $2i+1$ (multiplying by 2 is equivalent to left shifting the binary representation). (The root node of degree 1 is usually excluded from the structure.) The de Bruijn topology thus naturally generalizes the heap structure for constructing d-ary trees.

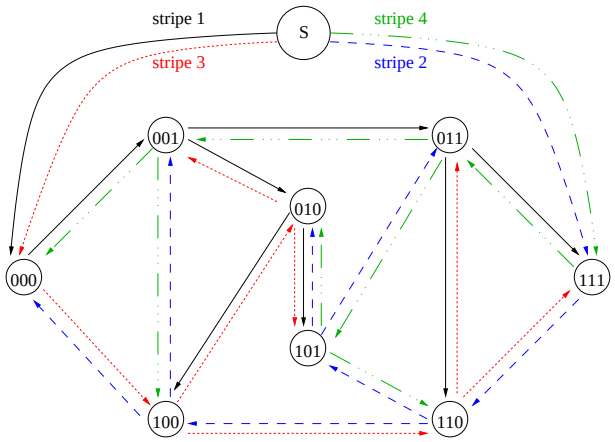

Figure 1: A simple example illustrating PrefixStream inter-cluster communications with $d = 2$ and $p = 3$.

Figure 1 illustrates how PrefixStream enables bi-directional exchanges between nodes of the de Bruijn graph. We will see in the sequel how each node of the de Bruijn graph is operated by a cluster of peers. The trees are rooted at nodes "000" and "111" which have degree 1 (other interior nodes have degree 2). Stripe 1 (resp. stripe 3) is multicasted in the left (resp. right) tree of "000", stripe 2 (resp. stripe 4) in the left (resp. right) tree of "111". (Notice that roots have degree 1 and other interior nodes have degree 2.) More generally, with a base d de Bruijn graph, the stream is split into $2d$ stripes. Each stripe is multicasted in turn in one of the $2d$ trees of degree d rooted at one of the d nodes with all the same digits. Each node is interior in one left shifting multicast tree, and in one right shifting multicast tree. It is a leaf in other trees. When a node serves d nodes in the left (resp. right) shifting tree where it is interior, it is served by one of them in each d right (resp. left) shifting tree. This is the basic property allowing exchange reciprocity.

Note that a node failure affects at most two stripes over $2d$, which is almost equivalent to loosing one stripe over d. Indeed, if the failing node is not a root, very few nodes may actually loose two stripes as detailed in Annex B.

3.2 Clusters

A set of d interior-node-disjoint trees of degree d can only be obtained when all the trees are complete d-ary trees. This is only possible with $n = (d^{p+1} - 1)/(d - 1)$ for some p. Similarly, the disjoint trees of the de Bruijn graph with roots of degree $d - 1$ requires exactly d^p nodes for some p. To accommodate to a variable number of peers and to increase resilience, PrefixStream uses a clustering scheme. Peers are partitioned into d^p disjoint clusters and clusters exchange data along the de Bruijn graph as described in the previous section. p may vary when the number of peers changes by a factor of d.

Definition 2 *A cluster tree is a tree between clusters. Clusters are disjoint sets of participating nodes. In a cluster tree each node knows the nodes belonging to its cluster, the nodes of its parent cluster and the nodes of its son clusters.*

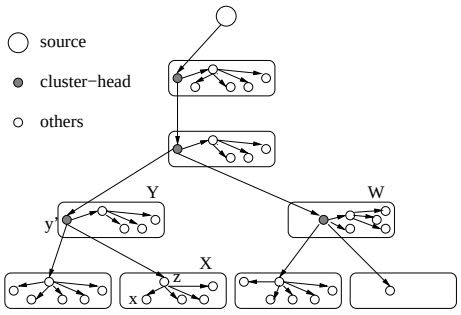

Figure 2: Multicasting inside a cluster tree of PrefixStream.

Figure 2 illustrates how PrefixStream distributes the forwarding load in a cluster tree. A cluster head of an interior node is elected by receiving a packet from the parent cluster. It forwards the packet to each cluster son, and then to another member of its cluster (if there is one). This delegate node then forwards the packet to all other members of the cluster as in cluster Y of Figure 2. The cluster head may additionally forward the packet to more members of its cluster for load balancing purposes as explained in Subsection 3.4 and illustrated in cluster W of Figure 2. In a leaf cluster, the receiving node forwards to all other nodes of its cluster. Cluster heads may change for each packet of the stream. However, good reciprocity and latency optimizations should stabilize this choice.

3.3 Incentive to cooperate

The basic idea for allowing reciprocity between clusters is to select as cluster head the nodes that most often serve the cluster in reversed trees. Assume the source uses a $2d$ cyclic sequence to select the roots of the cluster trees. A

first round uses the d left trees and a second round uses the d right trees. A possible choice consists in using the left trees with roots $0\cdots0$, $1\cdots1$,... and the right trees with roots $0\cdots0$, $1\cdots1$,... In the most simple scheme, each cluster head forwards its packet to the node that most recently sent a packet of a previous round to its cluster. However, a cluster head failure could then result in the loss of d packets in a row inside its cluster. To avoid this we propose to use the following logical reciprocity.

When cluster head x forwards its packet to a node y in a cluster son Y, it piggy-backs the identifier and address of the delegate node z of its cluster that should be used for reciprocity. In the following round, when cluster head y' of Y has to forward a packet back, it sends it to z as illustrated between clusters Y and X of Figure 2. To allow cluster head stability, x indicates itself as delegate node for its parent clusters in the two trees where its cluster is interior. Most probably, y' will thus be y. As the previous cluster sender is piggy-backed, x can acknowledge the reciprocity of y. The identifier of x should also be piggy-backed in the packet from y' to z, so that z can acknowledge the reciprocity of x (z receives a packet because x has send a packet to y). Other nodes in the cluster of z only acknowledge z for reciprocity.

Basically, inside a cluster of size m, each node forwards a distinct packet of the stream over m to any given member. (This is done indirectly by the cluster head $d-1$ times over d.) Nodes showing good reciprocity are served before the others. Depending on its remaining bandwidth, a node may serve only some of the nodes with low reciprocity. It is important to generously serve every node from time to time. Otherwise, any network failure would result in the banishment of the node that cannot fulfill its contract. Similarly, new nodes have to receive some packets for proving their will to participate.

Concerning inter-cluster reciprocity, proper functioning is obtained when inter-cluster forwarding always occurs. When a cluster head does not receive packets back from another cluster, it selects another receiver in that cluster. Inter-cluster forwarding is thus always maintained as long as a well behaving node is found in each cluster. As the cluster heads of root clusters forward to $d-1$ clusters only, they can use their spare transmissions to acknowledge reception of the packets sent to the previous root by the source. This allows the source to select well behaving cluster heads in root clusters.

Notice that our scheme privileges nodes participating to the forwarding load of the system. This gives incentive to cooperate. However, it is not resilient to Byzantine failures in the purpose of disturbing the system. Pushing forward to such requirements is beyond the scope of this paper.

3.4 Fully balancing forwarding loads

In the above scheme, delegates nodes of a cluster have an ideal load of 1: they forward one packet of the stream over $m-1$ and forward it to $m-1$ nodes at most. On the other hand, the cluster head forwards one packet of the stream over d and forwards it to $d+1$ nodes. This results in a load of $1+1/d$. Notice that there is still some incentive to be cluster head since it is the only node that is not affected by the failure of another member of its cluster. However, we show how to obtain fully balanced loads.

The idea is to use two cluster heads with a $4d$ cyclic tree-root sequence. To avoid collisions, each cluster is virtually split into two halves according to a total order of the identifiers. When a cluster head is chosen among the nodes with smallest (resp. greatest) identifiers, it is called a low (resp. high) cluster head. The first and the third rounds of the cyclic sequence are similar to two rounds of the previous scheme and always use low cluster heads. The second and the fourth rounds are also similar but use high cluster heads.

Each cluster head should avoid to use the other cluster head as a delegate. (To agree on the choice of delegates, packet i can be delegated to node in position i modulo $m-2$ in the ordered list of other members.) To maintain its intra-cluster reciprocity, each cluster head forwards its packet to $d-1$ more members of its cluster as illustrated in cluster W of Figure 2. To make incentive more robust, these members are the $d-1$ delegate nodes that should receive a packet back from inter-cluster reciprocity. (Remember that the cluster head selects itself as delegate for the tree where its cluster is interior.) If such a delegate node does not receive later on the packet from the other cluster, it still sends an empty packet to all other members of its cluster. This proves its willingness to cooperate and it informs the cluster head that it should select another receiver in the other cluster.

Over a cycle of $4d$ packets, a cluster head now sends $2d$ packets two times and has a load of 1. An ordinary member of a cluster of size m sends $m-d-1$ packets as a direct delegate and $m-1$ packets as an indirect delegate. As it is selected once over $m-2$ packets, its load is $\frac{1}{m-2}(\frac{m-d-1}{d}+\frac{d-1}{d}(m-1))=1$. A stabilized choice of cluster heads thus results in ideally balanced loads in the cluster. When cluster heads choices are not stabilized, the load can still be maintained close to 1 by giving priority to the cluster head role and partially fulfilling the delegate role to maintain good reciprocity.

3.5 Resilience to node failure

First note that the above balanced scheme succeeds in building interior-node-disjoint trees when cluster size is greater or equal to $2d+2$. More precisely, the $2d$ trees used to multicast $2d$ consecutive packets of the stream have disjoint interior nodes. The trick is that with $m>2d+2$, an ordinary member of a cluster is interior in successively different trees allowing to fully balance loads despite a variable number of nodes. In case of node failure, a remaining node looses at most 2 packets over $4d$. As the number of ancestors of a cluster is $2d(p-1)$ at most, packet loss is likely to remain below this threshold of $1/(2d)$ as long as the number of simultaneous failures is lower than $1/(4d(p-1))$. Note that the reparation time of a tree is simply the time required to detect the node failure, selecting another forwarder in the

cluster is simply instantaneous. This typically means that with a few percent of node failures per minute the packet loss remains lower than $1/(2d)$ in the network. The above balanced scheme can be used to send $4d$ stripes guarantying that $4d-2$ at least are received most of the time. It could also be used to send $2d$ stripes guarantying that $2d-1$ at least are received most of the time. Finally, notice that this scheme could be generalized to obtain k disjoint multicast trees per cluster tree by selecting k disjoint cluster heads in each cluster. This is interesting for enabling a large number of stripes with low degree trees.

3.6 Forming clusters

Each participating node chooses a random identifier written in base d. Identifiers should be long enough to make collisions highly unlikely. We propose to form clusters by gathering together all nodes whose identifier shares a common prefix of length $p \approx \log_d(n/m)$ where n is the number of nodes and m is the desired number of nodes per cluster.

For that purpose, we use a prefix metric on identifiers. We call prefix metric a metric such that distance decreases as common prefix length increases. (A simple choice is the bitwise xor of identifiers [13].) We say that two nodes are close if the distance between their identifiers is small.

A node can estimate p as the common prefix length of its m closest contacts. Each node may get a different estimation. However, when identifiers are chosen randomly, it is a classical balls and bin result to show that with high probability, all other nodes will get an estimation greater than $p-c$ for some suitable low value c. All nodes thus gather all contacts whose identifier begins with the same $p-c$ digits as itself. (This will typically represent $O(\log n)$ contacts.) Each node can then maintain what would be its cluster member list if the prefix length was p, $p-1$,... or $p-c$. As a result, it is sufficient that the source knows the overall lower estimation p_m of p. It can piggy back p_m in each packet and all nodes operate with the clusters obtained with prefix length p_m. Notice that different packets may be multicasted along the clusters obtained with different values of p_m. To maintain p_m, a node may alert the source when the number of members in its cluster for prefix p_m is lower than m or greater than $d^c m$. Finding nodes with a given prefix is delegated to a distributed hashtable.

3.7 Distributed hashtable

Note that the de Bruijn graph has already been suggested for constructing peer-to-peer distributed hashtables [14, 7, 10, 1, 8]. Any distributed hashtable allowing to find nodes having some given prefix could be used to compute clusters as described above. For example, a distributed hashtable using prefix routing such as [17, 13] is a suitable choice. However, as inter-cluster communications will follow the de Bruijn topology, a de Bruijn based solution will certainly allow to reduce control overhead.

Practically, the de Bruijn topology between clusters is built as follows. Each node u estimates p as above. Let $u_1 \cdots u_{p-c}$ be its prefix. It then finds through the distributed hashtable all the nodes with one of the $2d+1$ following prefixes: $u_1 \cdots u_{p-c}$, and $yu_1 \cdots u_{p-c-1}$, $u_2 \cdots u_{p-c}y$ for all $y \in \{0, \ldots, d-1\}$. Several lookups may be necessary for each prefix depending on the distributed hashtable used. When the estimation p of a node increases, it simply cuts its contact lists. When p decreases, new lookups are performed. As PrefixStream still behaves normally as long as there is one node per cluster, this gives time for performing these lookups.

3.8 Balancing Clusters

As we will see in Section 4, a larger cluster incurs longer delays for its members. Nodes thus have incentive to join the smallest cluster as possible. A rather classical way to achieve this is to make new nodes choose 2 (or even $\log n$) random identifiers and to select the identifier resulting in the smallest estimation of p [3]. Such heuristics typically bound the ratio of cluster sizes by $O(\log \log n)$. To maintain balanced clusters when the overall number of nodes decreases, nodes experiencing an estimation of p much higher than the actual value p_m read in the stream packets should try to choose another identifier. (Some random decision process should be used to desynchronize such reassignments.) Alternatively, another solution [12] guarantees a $O(1)$ factor between cluster sizes. It uses a $O(\log n)$ walk after insertion and reassigns identifiers from time to time.

4 Performance analysis

We now introduce our network model and then provide an analysis of propagation delays.Packet loss has already been analyzed in Subsection 3.5.

4.1 Network Model

The source is an external node reliably furnishing the packets with a bandwidth 1, it does not duplicate packets. (The source could be a synchronized cluster of nodes acting as a single node for more reliability.) To analyze delays, we use the following model of transmission. It is similar to the LogP model [6] except that we suppose a fixed packet length. A packet requires a time T to be emitted and transits a time L in the network. (When sending i packets in a row, the last packet will thus be received after a time $iT+L$.) With a 128 kbit/s upload bandwidth and 1000 bits packets, T is typically less than 10 ms. L is half of the average round trip time on the Internet and varies typically between 20 and 150 ms. Lower bounds on the minimum delay for multicasting a packet to n nodes can be found in [11]. The normalized latency $l = L/T$ is the maximal number of packets a node can send in a row before the first packet is received.

4.2 Delay analysis

The tree height is the prefix length (in digits) minus one and thus equals $h = p-1 \approx \log_d(n/m)-1$. The maximal delay is

obtained for a leaf cluster. It is thus bounded by the source delay $T + L$ plus the tree delay $h(dT + L)$ plus the intra-cluster delay $(m-1)T+L$ where m is the cluster size. Notice that for fairness with respect to delays, cluster sizes should be well balanced. Indeed, the intra-cluster forwarding algorithm imposes a reordering buffering window of m packets. Figure 3 illustrates the normalized delay bounds obtained by PrefixStream (a scheme maintaining a large cluster size $m \in [2d + 2, 2d^2 + 2d]$ is assumed.). We see that choosing $d = 4$ allows to obtain delays within a factor of approximately 1.9 from the theoretical optimal delay for a wide range of $l = L/T$ values. Notice that we obtain large delays for $d = 8$ because the curves assume the large cluster size of $2d^2 + 2d = 144$ resulting in a very long intra-cluster delay. This is the cost for targeting $m = 2d+2$ and ensuring a low packet loss. $d = 4$ is certainly the best compromise enabling a $4d = 16$ stripes scheme with relatively small clusters.

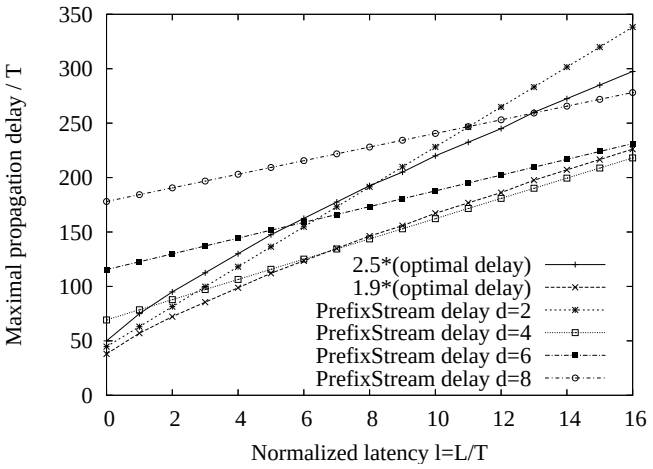

Figure 3: PrefixStream normalized delay bound as a function of $l = L/T$ for different values of d with $n = 10^6$ nodes.

In a real network, various values of l are experienced. Latency optimization can be obtained by preferring cluster heads closer in the network. PrefixStream thus allows to reduce the forwarding time of the $h-1$ first inter-cluster forwarding hops. Notice that intra-cluster optimization would not reduced notably propagation delays.

5 Conclusion

We have introduced PrefixStream, an incentive content distribution system based on end-system multicast in a clustered de Bruijn topology. PrefixStream has guaranteed delay performances with regard to the theoretical optimal. The main breakthrough of PrefixStream is to enable exchange reciprocity and thus tit-for-tat like incentive mechanisms. The system is able to reduce the forwarding load of each participating node to the stream bandwidth and the loads remain balanced under churn. PrefixStream disjoint clustering scheme enables high resilience to node failures. Besides simulating and experimenting the protocol, interesting future work resides in better tolerating large clusters with regards to delays. This could be possible through recursive use of PrefixStream inside large clusters.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proc. of the 17th Int. Symp. on Parallel and Distributed Processing (IPDPS)*, 2003.

[2] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Peer-to-Peer Systems II: Second International Workshop IPTPS*, 2003.

[4] 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 Symp. on Operating Systems Principles (SOSP)*, 2003.

[5] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. of the 4th ACM SIGPLAN Symp. on Principles and practice of parallel programming*, pages 1–12, 1993.

[7] P. Fraigniaud and P. Gauron. An overview of the content-addressable network d2b. In *Brief annoucement at 22nd ACM Symp. on Principles of Distributed Computing (PODC)*, 2003.

[8] Anh-Tuan Gai and Laurent Viennot. Broose: a practical distributed hashtable based on the de-brujin topology. In *Proc. of the 4th IEEE Int. Conf. on Peer-to-Peer Computing (P2P)*, 2004.

[9] C.-T. Ho and S.L. Johnsson. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on computers*, 38(9), 1989.

[10] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[11] R. Karp, A. Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation in the logp model. In *Proc. of ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 142–153, 1993.

[12] Gurmeet Singh Manku. A randomized id selection algorithm for peer-to-peer networks. In *Proc. of 23rd ACM Symp. on Principles of Distributed Computing (PODC)*, 2004.

[13] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric. In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[14] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proc. of the 55th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, 2003.

[15] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *ACM/IEEE NOSSDAV*, 2002.

[16] F. Pianese. Pulse: A novel unstructured approach to p2p live media streaming. In *E-Next WG3 CDN Workshop*, December 2004.

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

[18] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking 2002*, 2002.

[19] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE INFOCOM*, 2003.

Annex A

We show here that the basic algorithm of SplitStream [4] for constructing interior-node-disjoint trees produces some interior nodes with degree much larger than d. This implies the necessity for a balancing mechanism when nodes do not have spare bandwidth.

When there is no bandwidth constraint, each tree of SplitStream is built as the union of Pastry routes from all nodes to the root of the tree. Consider a tree with root $u = u_1 \cdots u_p$ (nodes are identified by a sequence of digits in base d). As Pastry uses prefix routing, the route from a node v to u follows a node with same first digit as u, a node with same 2 first digits as u, and so on. The average length of a route is $l \approx \log_d n$, the prefix length necessary to distinguish a node identifier from others.

Now consider an interior node x of the tree obtained with root u. Let i be the length of its longest common prefix with u. x is thus one of the d^{l-i} nodes with prefix $u_1 \cdots u_i$. (Our analysis assumes balanced choices of node identifiers.) Each node with shorter common prefix $u_1 \cdots u_j$ $(j < i)$ may choose such a node as a Pastry contact for its $j+1$ routing table entry. With a random choice, this occurs with probability $1/d^{i-j-1} \times 1/d^{l-i} = 1/d^{l-j-1}$. There are $(d-1)d^{l-j-1}$ such possible sons. The average tree degree of a node with prefix $u_1 \cdots u_i$ is thus roughly $\sum_{j=0}^{i-1}(d-1) = (d-1)i$. This can be as high as $(d-1)\log_d n$ for the root.

Notice that our analysis assumes balanced choices of routing table entries. $(d-1)i$ is thus a lower bound of the maximal degree of a node with prefix $u_1 \cdots u_i$. However, Pastry selects routing table entries minimizing network latency. This biased choice could produce some interior nodes with larger degree and some others with smaller degree.

Annex B

With the symmetric de Bruijn topology, any node is interior in two trees. It thus has two sets of descendants. We show that these two sets are almost disjoint for non-root nodes. This implies that a node will probably loose only one stripe over $2d$ in case of node failure in the network. (In the worst case, a root failure for example, it looses two stripes.)

A non root node u has an identifier of the form $a^i u' b^j$ where the first digit a is repeated $i \geq 1$ times and the last digit b is repeated $j \geq 1$ times. As u is not a root, it contains at least to different digits. If u' is empty then a must differ from b. If not, the first digit of u' is not a and its last digit is not b.

A descendant v in both the left and right tree where u is interior is called a double descendant. Such a node v must have an identifier of the form $v = a^{i'} u' b^j x = y a^i u' b^{j'}$ where $1 \leq |x| \leq i$, $1 \leq |y| \leq j$, $i' = i - |x|$ and $j' = j - |y|$. ($|w|$ denotes the number of digits of a sequence w and w^k denotes the concatenation of k copies of w.) x is thus a suffix of $a^i u' b^{j'}$ and y is a prefix of $a^{i'} u' b^j$. As the last digit of $a^i u'$ is not b, we have $|x| > j'$. Similarly, we have $|y| > i'$. We can thus write $x = x' b^{j'}$ and $y = a^{i'} y'$. As $v = a^{i'} u' b^j x' b^{j'} = a^{i'} y' a^i u' b^{j'}$, we deduce $u' b^j x' = y' a^i u'$.

First consider $a \neq b$. We either have $|u'| + j \leq |y'|$ or $|u'| \geq |y'| + i$. As $|y'| + i = |x'| + j$, the second case also implies $|u'| \geq |x'| + j$. We can then write $u' = y'' b^j x' = y' a^i x''$ with $|y''| = |x''|$. $u' b^j x' = y' a^i u'$ then implies $y'' b^j x' b^j x' = y' a^i y' a^i x''$. By iterating the former arguments we can prove by recurrence that $(b^j x')^k = (y' a^i)^k$ for some $k \geq 1$. As $|x'| \leq i$ and $|y'| \leq j$, we obtain $x' = a^i = x$ and $y' = b^j = y$. There is thus at most one double descendant when $a \neq b$. (As an example, $u = (ab)^i$ has $(ba)^i$ as double descendant.)

Let us now assume $a = b$. u' must be non empty and does not begin or end with a. As we have $u' a^j x' = y' a^i u'$, we thus deduce $|x'| \geq |u'|$ and $|y'| \geq |u'|$. We can then write $x' = a^{i''} u'$ and $y' = u' a^{j''}$, and any double descendant v can be written $v = a^{i'} u' a^{j+i''} u' a^{j'} = a^{i'} u' a^{j''+i} u' a^{j'}$ with $0 \leq i' < i$, $0 \leq j' < j$, $i - i' = i'' + |u'| + j'$ and $j - j' = j'' + |u'| + i'$. All values for i' and j' are possible as long as $j'' \geq 0$ and $i'' \geq 0$, i.e. $\min(i,j) \geq i' + j' + |u'|$. The number of double descendants is thus $m(m+1)/2$ with $m = \min(i,j) - |u'| + 1$. Let $p = |u|$. As $|u'| \geq 1$, the number of double descendants when $a = b$ is thus at most $(p^2-1)/8$ when p is odd and $(p^2-2p)/8$ when p is even. (This is indeed the number of double descendants for $u = a^i c a^i$ where c is a digit different from a.)

The number of double descendants of any non root node is thus bounded by $\max(1, \frac{p^2-1}{8})$. This bound is tight. It should be compared to the number of nodes in the de Bruijn graph: d^p. For example, with $4^{10} > 10^6$ nodes, p is 10 and a node has at most 10 double descendants.

Notice that the number of double ancestors of a node is at most $p - 2 + d$ including the roots (this bound is met for $v = ca^{p-2}c$).