Publishing, Retrieving and Streaming Lectures via Application Level Multicast

Marco Milanesio  
Dip. di Informatica - Università di Torino  
milane@di.unito.it

Giancarlo Ruffo  
Dip. di Informatica - Università di Torino  
ruffo@di.unito.it

Abstract

Structured peer-to-peer overlay network is an efficient solution for querying and retrieving resources spread between the peers. Unfortunately, key based routing of Distributed Hash Tables is simple when keys are known in advance, and at the service level this cannot be always assumed. In this paper, we present DHTeach, an e-learning application built on a totally distributed overlay network, where the problem of key definition is solved using an ad hoc description language. Moreover, DHTeach lays on a multicast application layer that organizes peers in order to make them able to receive the multimedia stream concurrently from a set of partners participating the class. The overall architecture is fully integrated and interoperable.

1 Introduction and Motivations

Recently, a lot of effort has been devoted to investigate several domains (e.g., distributed cooperative storage, application level multicast, and streaming) in order to convince the broadest community that the Peer-to-Peer technology is mature enough to give enhanced services to the users and/or to the providers, beyond the File-Sharing domain.

Nevertheless, except for the case of CoolStreaming, none of the many proposed systems have reached the popular success of other well-known p2p systems like BitTorrent, Gnutella, and eMule. The reasons for this apparently discouraging response could be many: e.g., (1) overlay networks based on Distributed Hash Tables (DHT) are too complex, (2) the related APIs are poorly documented and difficult to use, (3) there are few (maybe none) media providers that use these platforms to distribute their content, (4) it is not clear which alternative service can be given to the users that other client/server systems do not properly accomplish as well.

The first two argumentations have been largely confuted by many recent applications of DHTs (i.e., Kademlia, Coolstreaming, Skype). But the third and the fourth of the above reasons should be carefully taken into consideration, because it is not clear enough if overlay networks can be efficiently used outside the File-Sharing domain. In particular, before considering P2P applicable as a streaming platform, we have to answer to some questions.

By first: why overlay networks should be preferred to IP multicast; why P2P should be preferred to other overlay networks (e.g., CDN); which kind of service can motivate providers to distribute their content in a collaborative manner?

Furthermore, how is it possible to maintain an acceptable QoS in a network of highly transient nodes, and that shows Free Riding phenomena; how can a stream event be properly announced, without a central entity and without flooding the entire network and, finally, how can an user start streaming the desired event in a fully decentralized manner, considering how much is difficult to search a given object in a DHT?

The answers to the first group of questions are straightforward: IP multicast is not supported by many routers in the Internet, and as a consequence overlay networks are preferred in geographical domains. The main drawback is that server replications based methodologies are highly costly and can be adopted only by great content providers. P2P is an alternative “democratic” methodology, that enables small-medium companies and institutions to provide live and on-demand bandwidth intensive content. Moreover, application level multicast trees trivially outperform traditional TCP/IP unicast streaming, that is, at the present time, the most adopted solution in Web client-server architectures.

In order to answer to the other questions we focus on an e-learning application, that is layered on top of an application...
level multicast maintained in a P2P network. It is our opinion that an e-learning streaming platform can become one of the killer applications for DHTs, because of the efficiency of such a solution (even from an economic point of view). Moreover, social relationships between users (i.e., teachers, students and tutors) participating to the same group (e.g., to follow a given lesson, to post questions to the teacher and/or the audience, . . . ) are intrinsically collaborative.

2 Road map

As far as we know, this is the first work that proposes an E-Learning platform where every functions can be solved using a fully integrated DHT based architecture. For example, the EDUTELLA project [9] provides an open source P2P infrastructure that applies Web Semantics standards for defining a query and description language for generic educational objects. Nevertheless, this infrastructure does not define how downloading, streaming and searching can benefit of such a representation (in terms of efficiency, reliability, and so on), or which P2P topology fits at the best this environment.

A general framework (formalized in Section 3) is used to present the DHT layered architecture of our proposal. Our system architecture, namely DHTeach, is designed in Section 4, while protocol messages and primitives are showed in Section 5. The conclusions are given in Section 6, where on going work and the state of the art of the prototype is shortly described.

3 Domain Description and Definitions

A formal description of our domain is given with an abstraction of a generic overlay network (i.e., a DHT) implementing the lookup primitive. We prefer to use such a generalization because any implementation of our protocol can be made using one of the many different existing DHTs (e.g., Pastry [12], Chord [5], Kadmeilia [11]). Again, since our proposal is based on different abstraction levels (see Fig. 1), such that the value in a layer can be the key in another layer, we wanted to unambiguously define the requirements for our primitives.

Let \( N, R \) and \( K \) be respectively the set of nodes, of resources (or, even, values), and of index keys. Furthermore, we need to map \( N \) and \( R \) to an identifier space, namely \( I \). Considering that nodes, resources and keys are encoded using binary strings, we have that \( N, R, K \subseteq \{0,1\}^* \). Given a constant \( h \in \mathbb{N} \), we set \( I \equiv \{0,1\}^h \) (e.g., \( h = 160 \)). As a consequence, we can define two (hash) functions \( H_1 : N \rightarrow I \) and \( H_2 : K \rightarrow I \), that map nodes and keys in the same identifier space\(^3\).

Let \( \bar{N} \subset N \) be the set of active nodes in a given moment, and let \( O = (\bar{N}, L) \) be the graph defining the overlay topology, where the set \( L \) of links is a collection of pair \((n,m)\) of nodes in \( \bar{N} \). There is not a global representation of \( L \), but we can define the routing table of an active node \( n \) as

\[
RT(n) = \{(m, id_m) : (n, m) \in L, id_m = H_1(m)\}.
\]

Of course, \( L \) is defined by construction in terms of

\[
RT = \bigcup_{i=0}^{\bar{N}} RT(n_i).
\]

Furthermore, each node in \( \bar{N} \) is responsible for a set of key identifiers. Let \( M : I \rightarrow \mathcal{P}(\bar{N}) \), where \( \mathcal{P} \) is the power set, be the function that maps the identifier of a key to the set of active nodes responsible for it, i.e., the resource \( r \) indexed by key \( k \) with identifier \( id_k \) (i.e., \( H_2(k) = id_k \)) is replicated into the nodes in \( M(id_k) \).

3.1 Primitives of the DHT Layer

Here we define the basic operations in a Distributed Hash Table. Every operation is based on the lookup primitive, that is implemented in all the DHTs. Each primitive can be executed from any active node \( n \).

\( n.\text{lookup}(Key \ k) \): it implements the function \( M \), returning the set \( M(id_k) \) (or part of it), where \( id_k = H_2(k) \). In fact, the current node looks into \( RT(n) \), and contacts the node \( m \) whose identifier \( id_m \) is closer than the others to \( id_k \). This procedure is repeated until the nodes responsible for any of the replicas of \( id_k \) are reached. The closeness is calculated according to a metric defined in the given DHT.

\( n.\text{insert}(Key \ k, \ Resource \ r) \): it inserts a given number of replicas of the pair \((k, r)\) to all the nodes in \( M(H_2(k)) \).

\( n.\text{get}(Key \ k) \): it gets resource \( r \) indexed by key \( k \). It can be implemented by a lookup for \( k \), followed by a direct connection to any of the nodes returned in the previous step.

\[^3\]Note that many times we have that \( H_1 = H_2 = SHA_1 \).
3.2 Structures and Functions of the Application Level Multicast Layer

In the previous paragraph we introduced three functions that can be implemented in the actual overlay networks. The Application Level Multicast Layer introduces other structures (e.g., a multicast tree and/or forest interconnecting nodes) and other primitives are needed as well. For the sake of simplicity, we present this layer in terms of the requirements of the upper layers of our architecture (see Fig. 1). However, the reader can observe that many existing protocols fit this description (e.g., Scribe [8], DKS [6]). These proposals address many common problems of application level multicast (e.g., load balancing, localities, structure repairing, reliability, and so on). We refer to the original papers for in depth solutions.

We define a multicast group $MG = (\bar{N}_{mg}, L_{mg})$ as a directed graph where $\bar{N}_{mg} \subseteq \bar{N}$ and $\forall n \in \bar{N}_{mg}$ : exists a path connecting $m$ to a given rendez-vous point $rv$. Such a structure is usually a tree or a forest, where $rv$ is the root. Furthermore, each node $n \in \bar{N}_{mg}$ has a Children Table $CT(n)$, which is likely to be disjoint from $RT(n)$, that makes $n$ directly point to its adjacent nodes in $MG$. As before, the set of edges $L_{mg}$ is defined in terms of the union of all the $CT(n_i)$, with $i = 0, \ldots, |\bar{N}_{mg}|$. Messages in the Application Level Multicast Layer are routed through the Children Tables, in order to reach each member of the group.

Here we introduce also the concept of credentials: they are used for access control, and they are defined at the application level (see Section 5).

$n.createGroup$(Credentials $c$, GroupID $id_g$): When a node $n$ creates a group, it uses a given Group Identifier $id_g$. The node in the DHT layer responsible for $id_g$ is the rendez-vous point. Even if in principle the initiator $n$ of a group and the rendez-vous point can be different, no one forbids that $n$ is also the rendez-vous point. In this case, $n$ will select its identifier $id_n = H_1(n)$ as the group id. In the rest of the paper, for the sake of generality, we will not make such assumption, but the reader should always remember that the implementer of the system can force, for reliability reasons, that the group creator is always the rendez-vous point.

$n.subscribe$(GroupID $id_g$, Credentials $c$): If $n$ wants to join a group, it looks for the rendez-vous peer identified by means of $id_g$. When the message reaches a node already in the group, the latter can add $n$ in its children list, after checking the credentials.

$n.unsubscribe$(GroupID $id_g$, Credentials $c$): The node that wishes to leave a group, should send an unsubscribe message to its adjacent peer, in order to let them update their children tables. In the opposite case, repairing policies would be activated to maintain the overall consistency.

$n.sendMsg$(GroupID $id_g$, Credentials $c$, Text $msg$): A given message $msg$, described in terms of a grammar that can be defined at the application level, can be broadcasted to the members of a group identified by $id_g$. This message is simply routed all the way to the rendez-vous root. During the path, it is forwarded to all the unseen children. Each node should check the credentials before forwarding the message.

3.3 Observations

In our application, the streams from a node (i.e., teacher) to its group(s) (i.e., a classroom initiated by the teacher) use the multicast structure previously defined for delivering service messages, but not for transferring audio/video payload. It is important to mark such a difference, because many nodes can be involved in the multicast structure just as forwarders, and not as members of the group (e.g., Scribe [8]). It is unfair (and also unprofitable for the entire system) to consume the bandwidth of a node that it is not interested in a given service. Moreover, it is inefficient to use, in a dynamic domain, a fixed structure to make the information flow from a source to many nodes: it is not reliable and it does not cope against peers’ transience phenomenon. Furthermore, the bandwidth usage is not optimized when nodes use asymmetric connection like aDSL.

The architecture of the streaming layer, based on the idea of multi-source downloading, is similar to the one adopted in DoNet/coolStreaming [10]. It uses a bittorrent-like incentive mechanism [3], in which a node streams to and from a set of partners, discriminately choking the outbound bandwidth when the corresponding inbound traffic decreases4.

Finally, the most important challenge of the DHTeach layer is to define a proper scheme that enables the many participants (i.e., teachers, students, administrators, and so on) to access and enjoy the system in a fully decentralized fashion. The goal is to allow the users to announce new lectures, to retrieve and to forward the sources of a stream event, and to update previously inserted information, without a central gateway service. At the core of this proposal is the definition of a semantic for the keys to be used to publish and to retrieve the many kind of (meta-)resources that are inserted in the underlying DHT.

---

4Fairness in a peer-to-peer domain is also obtained by means of micro-payment schemes (e.g., [2]), but in this context the incentive scheme is devoted to improve efficiency against arbitrary strategies of peers that do not forward the stream they receive. As a consequence, we need a differential service that adopt real-time measures against free riders, while micro-payment schemes guarantee fairness in the long term.
4 Architecture of DHTeach

In our scheme, we have three different kind of users:

Administrators: They are authorized to insert and to update the list of teachers that can create new classes and give real-time lectures. Moreover, an administrator can also update the protocol template used for creating messages.

Teachers: They can create a (multicast) group, that will be used to provide the lessons for a given class. They are responsible for a set of classes, and they can create groups pertaining only to their lectures. Moreover, they publish information and announcements (e.g., the date of a new lesson), that can be modified presenting the right credentials.

Students: Students can join classes, attend lectures, and ask questions to other users.

In this domain, we are assuming that each user $n$ is identified by way of an asymmetric cryptography key pair, made of a public $PK_n$ and a secret $SK_n$ counterpart. These keys are used to authenticate messages, and to let the other users verify the given signatures. For simplicity, we assume a bijection between the set of active nodes and the set of users. As a consequence, we will equally refer to nodes and to users.

We refer to Section 5 for the description of the messaging system, that is based on an original key description language that hierarchically encapsulates higher levels’ keys into lower levels’ messages. In the following, we describe the streaming architecture.

Let $t$ be a teacher, that creates a multicast group $MG = (N_{mg}, L_{mg})$. The streaming layer maintains a new directed graph $SG = (N_{sg}, L_{mg})$, where $N_{sg} \equiv N_{mg}$. For each $n \in N_{sg}$, there is a Partnership Table $PT(n)$, that is initialized to the Children Table $CT(n)$, but that is dynamically updated during the transmission of the given stream event. Two nodes are partners if they exchange other payload packets. The identifier $id_m$ of a partner $m$ is stored in a given entry of table $PT(n)$, together with a value $bw_{nm} \in [0, 1]$ which is the percentage of the bandwidth that $n$ reserves to $m$.

When $t$ starts transmitting, it divides the stream $S$ in constant sized segments $s_1, s_2, \ldots, s_l$, that are sequentially sent to its partners. Each node can be both supplier and receiver of a segment $s_l$, except $t$ that is exclusively a supplier. Observe that incoming segments are sent to the player of a node in sequential order, and that node $t$ starts streaming after a brief delay, in order to create the former segments.

4.1 Streaming Layer

The Streaming Layer is responsible for the splitting of the stream and for the scheduling on each node, so, in order to do that, it must perform some methods to make the peers able to set some sort of priority to the stream segments and to their partners.

Three primary issues are concerned with this layer: the partnership management, the exchange of information between peers and the bandwidth usage.

Partnership: We told that for each $n \in N_{sg}$, a Partnership Table is provided. In fact, $\forall m \in PT(n)$, a keep-alive TCP connection is established between $n$ and $m$. When a node $n$ subscribes a multicast group, its partnership table is initialized to its children table $CT(n)$. During the streaming event, the set of partners can change dependently on the segments $s_i$ needed, as explained later.

Information exchange: Given a stream $S$, a node $n$ should also have some sort of representation of the progress of the stream itself, meaning the number of ordered segments $s_1, \ldots, s_l$ already received. This can be easily implemented by a binary array $A$. It is a fact that, given a segments sequence $s_1, s_2, \ldots, s_l$, and a binary array of length $l$, we can define a bijection between the stream’s segments and the element of the array: $A[i] \leftrightarrow s_i$.

Such a relation tells us that if the $i$-th element of the array is equal to 1 ($A[i] = 1$), then the $i$-th segment in the sequence ($s_i$) was already received by the node. Periodically, partners exchange their arrays (and their partners’ arrays) each other. This can yield the partnership table to be updated; in fact, a scheduler will force the node to select the best peer, namely $m$, for the next needed segment (i.e. the $s_{i+1}$ segment), depending on the available bandwidth (i.e. $bw_{nm}$). Peer $m$ is added to $PT(n)$, while partners supplying segments with low priority are candidates to be removed from $PT(n)$.

This periodical exchange between partners it’s similar to the DONet [10] Buffer Map usage, in which the Buffer Map, similar to our array, points out the needed segments for a node.

Bandwidth reservation: It is useful also a choking procedure similar to the BitTorrent’s [3] one, in order to make the peers unwillingly donate less bandwidth than the requested. The bandwidth choking is an incentive for the good behaving of the peers, and, therefore, for a higher QoS for all the peers in the system.

5 Description of DHTeach Layer

Each node runs a DHTeach client that stores locally an XML name space defining the tags used in our message description language. Messages are simply resources that can be published in (and retrieved from) the overlay network.

---

5 Indeed, we have another kind of user, the **tutor**, that substitutes the teacher when he is not available, and that also manages class forums. Nevertheless we will not give any further details for lack of space.

6 The size of the segments is not defined here. The reader should observe that an empirical evaluation is needed to find the best length, that can be, in principle, as small as a TCP packet (or even of an RTP packet, if Real Time Protocol is used between TCP/UDP and the DHT).
Each message is split in three tags, i.e., an header, a body and a signature (see Fig. 2). The main idea is that the header contains information knowable a priori, and that every user can build them by her own. The header is the key for the message, and it is passed to the $H_2$ hash function to obtain a key identifier. With such a key, it is possible to insert and to retrieve the associated resource, i.e., the body of the message, that contains further information. Then, the authenticity of the message can be verified by means of the attached signature (that can be described using the W3C XML Signature scheme). The signatures are also used for checking credentials.

$$k \rightarrow H^{-1}_2(id_k)$$

$$<DHT \text{ Message Name}>$$
$$\begin{array}{l}
\langle \text{Header}\rangle \\
\ldots \\
\langle \text{Body}\rangle \\
\ldots \\
\langle \text{Signature}\rangle \\
\ldots \\
\langle \text{DHT \text{ Message Name}}\rangle
\end{array}$$

Figure 2. The scheme of a DHTeach message

5.1 Administrators Operations and Messages

The following operations force the insertion of messages only if they are produced (and signed) by an administrator. Observe that DHTeach clients are distributed with a certified list of current administrators’ public keys. Moreover, each message has a TTL (Time To Live): when it expires, then involved nodes will delete their local replica.

$n$.publishTeacherList(TeacherList $tl$): message $tl$ contains a list of users, that are authorized to provide lectures and to create multicast groups. Each teacher is identified by means of a name and an unique code (e.g., the social security number). Let $k$ be the header of message $tl$, i.e., $k = \langle\text{Header}\rangle <\text{Desc}>Teachers</\text{Desc} :</\text{Header}>"$; this procedure simply calls $n$.insert($k, tl$). Note that the node responsible for $id_k = H_2(k)$, should check that $tl$ has been signed by an administrator. In the opposite case, it will return an error message to $n$.

$n$.publishClassList(ClassList $cl$): The administrator is responsible for the insertion of classes announcements. Each class is identified by a unique code, and is associated to a teacher’s code. Again, the pair is $(k, cl)$ is inserted in the DHT, where $k = \langle\text{Header}\rangle <\text{Desc}>Classes</\text{Desc} :</\text{Header}>"$. Furthermore, for enhancing retrieving of classes’ announcements and information, it is possible also, thanks to polymorphism, to call the present procedure in the form $publishClassList(TeacherInfo \ ti)$, where $TeacherInfo$ is defined as the extended version. In other words, this enables the student to search a given class from the teacher code, because we can build the indexing key (i.e., the header), without other codes. Please also observe that this kind of message contains other information regarding a teacher, as her public key, that is useful to check credentials passed to restricted operations (e.g., announceLecture, startLecture, createGroup, and so on).

Updating Information and Language: For brevity, we cannot show all the procedures. In short, administrators can update previously inserted messages, even if they are not expired yet (i.e., updateTeacherList, updateClassList, updateNameSpace), simply removing the key-value binding and inserting a new pair. In particular, the message definition language can be substituted by inserting a new XML name space (in this case, $k = "\text{NameSpace}"$, and the resource is the DTD file). This is a particularly interesting feature of the system, because part of the protocol can be redefined without updating the software.

Finally, administrators public keys are periodically updated, and the message bound to key $k = "\text{AdminSignatures}"$ is signed with the one of the previous (and not black listed) signatures.

5.2 Teachers Operations and Messages

When a teacher announces a lecture, she must (1) create a group, (2) binds this lecture to a class, (3) publish the group identifier, together with temporal information (e.g., when the lesson is planned). After the lecture information has been inserted, the teacher can start streaming at the publicly announced date.

$n$.announceLecture(Lecture $lm$): The teacher $n$ creates a multicast group with identifier $id_g$, passing to $createGroup$ her credentials, too (i.e., she signs her own teacher code and $id_g$ with her private key $SK_n$). Hence, she inserts the Lecture message $lm$, that announces a lesson for a given class (stated by the class code). The key $k$ is the header of such a message. Finally, the message is inserted into the DHT ($insert(k, lm)$). Before expiration, the teacher can also update the announcement (with $updateLecture(lm)$).

$n$.startLecture(Credentials $\epsilon$, GroupId $id_g$): This procedure activates the streaming application, that starts transmitting the live lesson to nodes in the group previously created. Only the teacher responsible for the course can stream the

---

7http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/Overview.html
8The XML schemes of the messages are available on the extended version of this paper, that can be required to the authors
lesson to others. Credentials \(c\) are passed to underlying layers for permissions checking. Teacher \(n\) can stop streaming, too (with \(\text{stopLecture}(\text{Credentials } c, \text{ GroupId } id_g)\)).

### 5.3 Students Operations and Messages

\(n\).getTeacherList(): A student \(n\) can retrieve the current list of teachers (with their own codes) simply calling the DHT level procedure \(\text{get}(k)\), where \(k = "<\text{Header}>\text{<Desc>Teachers</Desc>}/\text{Header}>"\). This operation, like the following four, is not restricted, and every node can call it.

\(n\).getClassList(): Similarly, it is possible to retrieve the list of all the available classes. In such a case, \(k = "<\text{Header}>\text{<Desc>Classes</Desc>}/\text{Header}>"\).

\(n\).getTeacherInfo(Code \(tc\)): If a student wants to retrieve only the classes of a particular teacher, as well as retrieving the teacher’s public key, then he can get a TeacherInfo message. In this case, the indexing key should contain the teacher code \(tc\).

\(n\).getLectureAnnouncement(Code \(tc\), Code \(cc\)): Before joining a classroom, student \(n\) should subscribe to the related multicast group. He will search the lecture announcement creating an indexing key by means of the class code \(cc\) and the teacher code \(tc\).

\(n\).joinClassRoom(GroupId \(id_g\)): The previous operation returns a Lecture message, that contains the group identifier \(id_g\) bound to the wanted classroom. After calling the present procedure, \(n\) runs the Multicast Application Level procedure \(\text{subscribe}\). Hence, the player embedded to the DHTTeach client will wait for the incoming stream. The student can of course unsubscribe and leave the session (i.e., \(\text{leaveClassRoom}\)).

For example, if a node \(n\) wants to retrieve information about the lectures of a given teacher, then the header of the specific message has to be assembled with key \(k\) equal to "<\Header><Code>JHML75F67N</Code></Header>". If he wants to know more about the next lesson of the class Life of Conan the Barbarian of Prof. J. Milius, then he will create the key: "<\Header><TeacherCode>JHML75F67N</TeacherCode>-<ClassCode>CTB82rjc</ClassCode></Header>".

### 5.4 Other Operations

The following procedures can be called by every users.

\(n\).getNameSpace(): Periodically, the namespace can be renewed. Nodes can get the new description using the key "NameSpace".

\(n\).tell(NodeId \(id_u\), Text \(msg\)): If \(n\) wants to send a message \(msg\) to a given user \(u\), he should know identifier \(id_u = H_1(u)\). Otherwise, he can send a message to the group he subscribed to with \(\text{tellAll}(\text{GroupId } id_g)\). In the last case, the Multicast Application Level \(\text{sendMsg}\) procedure is called.

### 6 Conclusion and Ongoing Work

In this article we introduced an e-learning application built on a totally distributed overlay network. Using a DHT to address the peers and the resources, and using a Multicast layer to build our \textit{classroom}, we can separate the specific functions of each layer. This paper characterizes basically our approach to the problem of building a Streaming Application on a DHT. Presently we are developing a prototype, adopting several DHT-based APIs to investigate which one would be the more suitable for implementing this kind of application.

### References