DEVS Simulation of Peer-To-Peer File-Sharing

[SYSC 5104 Term Project Report]

Alan Davoust
Carleton University
Dept of Systems and Computer Engineering
Network Management and Artificial Intelligence Lab
adavoust@sce.carleton.ca

ABSTRACT

We present a set of tools to simulate peer-to-peer (P2P) file-sharing networks based on a discrete-event model. Unlike existing P2P simulation tools, that are mostly geared towards performance evaluation, we focus on the network content evolution, and include a model of the queries and documents shared by the peers. We describe the underlying modeling of file-sharing networks, and show how these models were converted into DEVS models and implemented with the tool CD++. Our system, can be decomposed into two main models, a peer model and a network model. Using scripting techniques we were able to combine thenetwork model with many instances of the peer model, and thus generate large-scale network models (several hundred peers) and execute them in reasonable time. Among our tools, we implemented a visualization tool that can show a simulation run as an animated graph.

Keywords

Peer-To-Peer, Simulation, DEVS

1. INTRODUCTION

This work is part of a broader research project about peer-to-peer file-sharing.

Most recent work in the field of peer-to-peer (P2P) systems has focused on two main characteristics of the P2P paradigm. The distributed infrastructure offers on one hand potentially high performance and scalability, both in terms of storage space and in terms of computational power, and on the other hand distributed control, without need for global coordination, and no single point of failure.

The performance aspects motivate a large body of work related to structured P2P network systems, such as Distributed Hash Tables (DHT). DHT aim to organize the network structure and file storage in order to make provide the peer community with a very large storage space, while optimizing the access time to the resources. Files are typically stored in a single place (or with just one backup copy), as multiple copies imply a waste of space, and the peer network is organized to minimize both the time for any peer to access any piece of data in the network, and the bandwidth used by the query system. Minimizing these parameters maximizes the scalability of the network.

The distributed control aspects are mostly explored in the database research community, specifically in the area of P2P data integration. P2P data integration is a general decentralized approach to data integration, where database management systems are interconnected in an arbitrary graph, rather than through a central mediator (which is the more traditional daa integration approach). Research in P2P data integration then examines the formal properties of query algorithms over these distributed systems, and related problems such as consistency issues between databases.

P2P file-sharing is an application that typically runs over unstructured networks, where peers find files of interest by means of very simple queries (typically keyword queries), and download them. P2P file-sharing is notorious for being both hugely popular, accounting by some estimates to more than 50% of internet traffic, and mostly illegal, as the shared files are often copyright-protected music and video.

In P2P file-sharing, no attempt is made to optimize the storage of files : users download resources from the network with the purpose of using a local copy (i.e. listening to music, viewing the video, etc.). The replication of files thus naturally follows the popularity of files : popular files are more frequently downloaded and widely available through the network, while obscure files that few people are interested in may be much more difficult to come across, as only a handful of copies will exist in the network, and only be available when their owner peers are online.

Another interesting characteristic then emerges, which is a natural filtering of the network content, whereby the most popular files also have the highest availability.

Furthermore, some protocols, such as the Gnutella protocol, offer only a limited view of the network : queries are only propagated to a certain horizon (normally seven network hops for Gnutella), and such protocols thus contribute to the filtering effect.

This emergent filtering can be compared to Google's pagerank algorithm, which addresses the problem of sorting the (typically millions of) web pages potentially relevant to a search engine keyword query : pagerank, essentially, pushes to the top the of the result list the pages that are the most linked to, which is a way of measuring their reputation or popularity.

In the information retrieval world, this type of filtering has proved to be of utmost importance, and the pagerank example is certainly relevant, as it made the reputation and fortune of its authors.

In our research, we are interested in applying the P2P file-sharing paradigm to other applications than simply sharing music and video. One aspect of this research is to study how this emergent ranking of content can be interpreted in the context of these new P2P applications. For example, within a network of interconnected databases, there could arise an inconsistency between a data entry found in one particular database, and a contradictory data entry found many times all over the network: in this case, the popularity of the latter data entry could be a determining factor to resolve the inconsistency.

Clearly, the emergent ranking of data that we envision cannot easily be studied in a real deployment: our prototype applications are currently only deployed in a handful of nodes within our lab. Ideally, by simulating a large-scale P2P network, we can produce some interesting emergent group phenomena, that result from reasonable assumptions on the individual peers' behavior.

The primary goal of this simulation is thus to simulate the behavior of peers in a network, with respect to particular queries and documents. Most existing simulation tools for P2P networks are mostly designed to generate realistic traffic in terms of volume and network latency.

This paper describes describes the current state of our project. So far, we have simulated a network with a large number of peers (up to 150), where peers exhibit a simple file-sharing behavior and their queries are properly routed through the network. We have also implemented a tool to visualize the output of a simulation in an animated graph.

The rest of this paper is organized as follows. After a brief description of related work in Section 3, we describe in Section 2 the general formal model of a file-sharing network, obtained in previous work, that we used as a basis for our DEVS modeling. Then in Section 4 we provide detailed specifications of the DEVS models that we have implemented in our P2P simulator.

In section 5 we describe the other tools that we have implemented for this project, including our visualization tool. In Section 6we present some preliminary results and directions for future work, and finally in Section 7 we draw a few conclusions.

2. BACKGROUND: FORMAL MODEL OF P2P FILE-SHARING

This work builds on previous work [3], where we introduced a formal model of a P2P file-sharing system. We briefly describe here this file-sharing model, which was used as a basis for the DEVS models developed in this project.

2.1 Modeling an Unstructured Network

A network formed by a set of peers $\{P_i\}$ can be modeled by a Labelled Transistion System (LTS), which is a special case of a deterministic automaton, where there is no notion of "accepting state" (i.e. all states are accepting).

Each state of the network is a graph where the nodes of the graph are a subset of $\{P_i\}$ and the edges represent the connections between the peers. The initial state of the LTS is an empty graph.

The possible events in a given state are addition or removal of a vertex or edge in the graph, i.e. one of the following :

- a (previously offline) peer goes online
- a (previously online) peer goes offline
- an online peer connects to another online peer
- an online peer drops its connection to another online peer

The conditions indicated for a given event (e.g. a peer can only go offline if it was previously online) indicate which state this event may occur in, and the resulting state after the event is fairly straightforward : if a peer goes online, the next state is the same graph with a new vertex associated with the peer.

2.2 File-Sharing Community

We now define the concept of a file-sharing community.

A file-sharing community is defined by :

- a set of peers
- a network formed by this set of peers (i.e. in any state, a subset of these peers forms the graph)
- a query protocol
- a data schema : we will ignore this component for now

In this community, each peer stores some data and is capable of outputting queries.

In terms of data and queries, we can consider the data items $\{d \in D\}$ and the queries $\{q \in Q\}$ to be two disjoint sets of abstract entities, and we have a binary relation which we will call *match* on these two sets (i.e. on the cartesian product $Q \times D$).

The community protocol is a function that takes a graph and a particular node of this graph as input, and returns a subset of nodes of this graph. This abstract function can

be applied to a particular state of a network: if a peer (connected in the network) outputs a query, the protocol specifies which peers receive the query. In this model we ignore time considerations (i.e. network delays) and network reliability issues (such as the possibility of messages being lost).

A peer has access to the following data operations :

- publish (d: data item) : add d to the peer's local repository;
- remove (d: data item) : if d is in the peer's local repository, then remove d;
- output (q: query) : output the query q to the network. The peer receives as a response the list of data items matching the query, and stored by the peers that were reached by the query (this being determined by the protocol);
- download (d: data item) : this operation assumes as a precondition that the peer does not store d locally, and that d was a response to a previous query by the peer. The post-condition of the operation is that the peer publishes a copy of d to its local repository.

3. RELATED WORK

There have been many other approaches to modeling P2P networks. Mostly the purpose of these efforts have been to accurately model aspects such as network traffic and bandwidth use, with the goal of estimating the scalability and efficiency (in terms of query response time) of P2P networks with particular structures and query protocols. Recent modeling efforts have modeled P2P networks as queueing systems (See for example [6]), a model that completely abstracts from the documents that are shared in the network.

Simulation tools are also mostly geared towards evaluating protocol performance, and most existing simulators thus focus on overlay protocols for structured P2P networks, such as Chord, CAN, Pastry, etc. A good example of such a simulation engine is Peersim [4], and another one built using a discrete-event modeling approach is Oversim [1]. As these systems are focused on performance evaluation, we speculate [1] that tracking particular documents and extending the systems towards managing, for instance, multiple types of documents and relationships between the documents is quite orthogonal to the development of these projects.

4. DEVS MODEL

In this section we describe in detail the different DEVS models that were implemented in this project.

4.1 Overview

The diagram in figure 1 shows the high-level design of the coupled model.

Our coupled model includes a variable number of sub-components: many peers and a single network component.

[1]We cannot be assertive as we have not properly investigated these simulators' inner design...

A peer is an autonomous component that attempts to make connections to its acquaintances (other peers), publishes documents, and outputs queries. We will consider downloads as particular cases of publishing. A peer's activity in the network starts with the peer going *online*, and ends when the peer goes *offline*.

Our coupled model includes a fixed number of peers, but not all of them are necessarily active (online) at the same time.

In our model, the peers are all connected to a single, relatively complex, Network coupled model. The Network maintains the state of the network connections and the data stored by all the peers all the peers, and uses this information to route queries in the network, and answer queries. The network is itself divided into sub-components, to maintain the network connection graph, to manage the query routing, to maintain the peers' databases, etc.

We note that as the Network is connected to each and every peer (with a different output port for each peer) some files within the network model must be generated by the same script that generates all the peers.

This coupled model does not have any overall inputs or outputs. In any simulation run, all the peers will go online, output some queries, then go offline again. The simulation ends when all peers have terminated their sessions.

4.2 Peer DEVS Model

A peer is a coupled model composed of two atomic models : The Connection Manager and the Session Manager. The outputs of the peer represent its network-related behavior, such as going online or trying to connect to other peers (through the "connect" output ports), its querying behavior (through the "query" output port). The peer also publishes documents (and removes them) through its "data_ops" ports. The detailed list of all input and output ports for the coupled model is given in Table 1.

Table 1: Peer coupled model : Input and Output ports

Port name	I / O	Comments
query	output	(file-sharing) queries
publish	output	files published by the peer
remove	output	removes a published file
online	output	peer goes online
offline	output	peer goes offline
out_connect	output	peer attempts a connection
out_disconnect	output	peer disconnects from a neighbor
in_connect	input	notifies of a connection being established
in_disconnect	input	notifies a connection being dropped
queryhit	input	queryhit (response to a query)

Each peer has a set of acquaintances, that it attempts connect with while it is online. The acquaintance graph is generated by a separate tool (see section 5 for more details), and its topology has the properties known as "small world" that characterize real social networks [2]. In most traditional peer-to-peer networks, peers do not normally connect

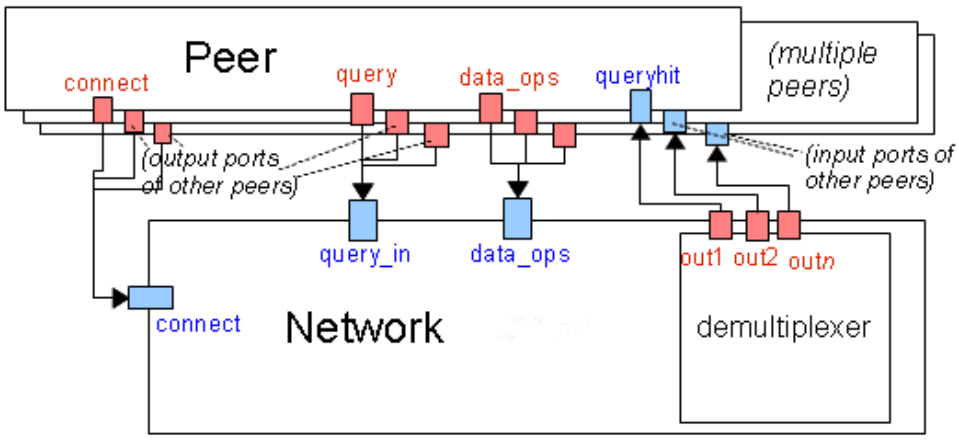

Figure 1: P2P System Coupled Model

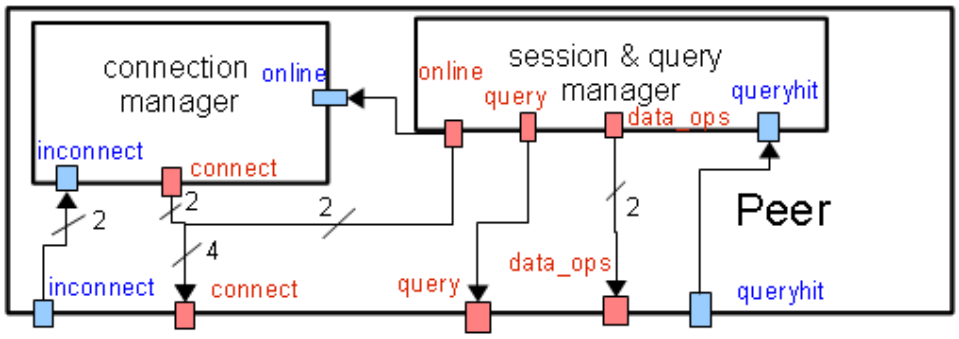

Figure 2: Peer Coupled Model

to "friends" but rather to "well-known" nodes that act mainly as access points to the network and are found in listings on the web. We are interested (in future work) in including some identification of peers and a social aspect to our P2P systems, as that will allow more possibilities to model trust relationships between peers.

4.2.1 Connection Manager

The connection manager is a component that simulates the behavior of the peer attempting to connect to the peers that it is acquainted with; this behavior, in a real system, would be built into the P2P client application.

The input and output ports are listed in Table 2. The component loads a list of peer identifiers from a file : these are the peers acquaintances. The connection manager starts in a passive state, then goes online when an input messages arrives on the "online" input port. It then attempts to connect to all its acquaintances, repeatedly, until all the connections are established. It goes passive again when it is notified to be offline.

Table 2: Connection Manager model : Input and Output ports

Port name	I / O	Comments
online	input	peer goes online
offline	input	peer goes offline
out_connect	output	peer attempts a connection
out_disconnect	output	peer disconnects from a neighbor
in_connect	input	notifies of a connection being established
in_disconnect	input	notifies of a connection being dropped

4.2.2 Session Manager

The Session Manager simulates the conscious behavior of the human user behind the peer : the user goes online, offline, outputs queries, and downloads files.

The Session Manager's input and output ports are listed in Table 3.

Table 3: Session Manager model : Input and Output ports

Port name	I / O	Comments
online	output	peer goes online
offline	output	peer goes offline
query	output	(file-sharing) queries
publish	output	files published (or downloaded) by the peer
remove	output	removes a published file

The session manager's behavior is loosely based on the study [5]. This study provides some "ready-to-use" models for the peer's behavior. In those model the peers are mostly inactive, and the few peers that are active only output a handful of queries (a mean of about three) during a session. This behavior is of course realistic, and is found in P2P networks where the connex components of the connection graph are very large (thousands of nodes). In our case, as discussed previously, we have a limited number of peers and they each only connect to their acquaintances; and the connex components of our graph are small. For the sake of observing some activity in the graph, we have made the peers mostly active. Each peer has an assigned list of initial documents, and a list of queries that it will make while it is online. These lists are randomly generated, using an external tool(see section 5 for details).

The behavior of a peer can be described as follows:

1. the peer waits a random time before going online (using an exponential probability distribution)
2. the peer goes online, publishes all its initial documents, and starts outputting queries from its list of queries
3. each time the peer gets a queryhit, it downloads (publishes) the corresponding document, unless it already has it.
4. once the peer has finished querying, it waits a random time, then goes offline.

In future extensions of this work, we may need to make the query generator more complex as it may need to simulate the behavior of a user faced with a more complex application (such as a wiki, where there are more data operations, and there may exist relations between different data items).

4.3 Network DEVS Model

4.3.1 Coupled Model Design

The network model will be a coupled model with the structure illustrated in figure 3.

The Input and Output ports of the Network Coupled Model are listed in Table 4.

Table 4: Network Coupled Model : Input and Output ports

Port name	I / O	Comments
query	input	a peer makes a query
publish	input	a peer publishes a file
remove	input	a peer removes a published file
peer_online	input	a peer goes online
peer_offline	input	a peer goes offline
peer_connect	input	a peer attempts a connection
peer_disconnect	input	a peer disconnects from a neighbor
out_[N]	output	used to notify peer number N of a queryhit
out_c[N]	output	used to notify peer number N of a connection or a disconnection

4.3.2 Dispatcher

The role of the dispacher is to generate a unique identifier for each query that is input into the network. The purpose this identifier is to track queries through the different network components. The dispatcher receives messages through its

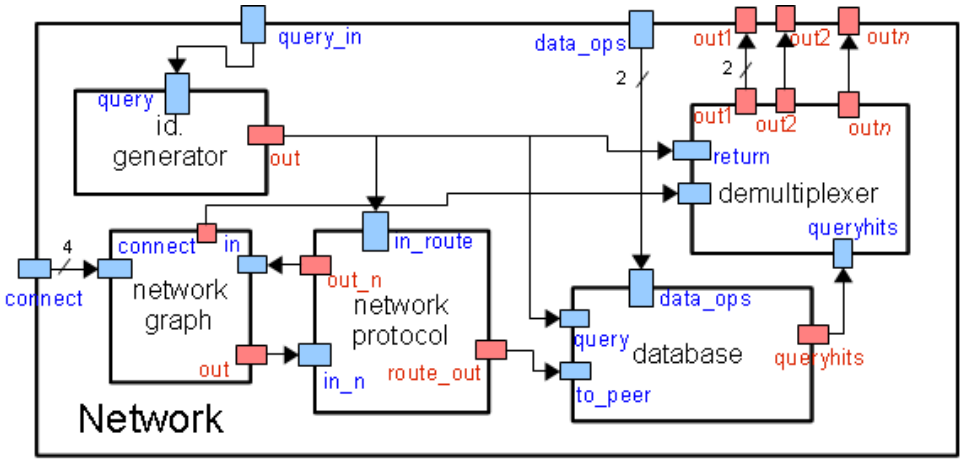

Figure 3: Network Coupled Model

input port, that encode a query and an identifier of the peer that output that query. Upon receiving this message, the dispatcher generates a unique message identifier, then generates two separate messages: one encodes the identifier with the query term, and one encodes the identifier with the query originator (i.e. the number of the peer that sent the query). These two messages are output through two output ports, to be used by the other components of the network.

This atomic model does not simulate a particular system within a real P2P network, and its state transitions are instantaneous.

The Input and Output ports of the Dispatcher Model are listed in Table 5.

Table 5: Dispatcher Model : Input and Output ports

Port name	I / O	Comments
msg_in	input	a query is output by a peer
peer_id	output	the peer number and message id
query_id	output	the query term and message id

4.3.3 *LTS Network*

The LTS network atomic model maintains the connection graph of the network, and closely matches the Labelled Transition Model described in section 2.1.

The model receives messages from the peers that go online or attempt to connect to other peers. Some transitions are illegal (two peers cannot connect if one is offline, for example), and the LTS component only changes the graph state following legal transitions. If a peer goes offline, all its connections are automatically dropped : the "disconnect" input port is therefore not used in practice, but we leave it for future use and consistency with the theoretical model.

When connections between peers are established, or dropped, the model outputs a message on its output port "out_connect" that is used to notify the peers about the state of the connection. This allows peers to know the state of their direct connections.

The model can also provide information about the connectivity in the network by the "in" and "out" ports: messages input on the "in" port indicate a peer identifier (the sender of the message) and a sequence of messages is output from the "out" port indicating which peers were reached by the message, i.e. which peers were connected to the sender. This behavior is currently quite tighly tied to the Gnutella protocol implemented in the Gnutella atomic model, as the messages passed include the time-to-live, and are "broadcast" from a peer to all its connections. However, with only minor changes it could be adapted to be used for other protocols.

The inputs and outputs of this model are summarized in the Table 6.

Table 6: LTS Atomic Network Model : Input and Output ports

Port name	I / O	Comments
peer_online	input	a peer goes online
peer_offline	input	a peer goes offline
peer_connect	input	a peer attempts a connection
peer_disconnect	input	a peer disconnects from a neighbor
out_connect	output	notify of a connection
out_disconnect	output	notify of a disconnection
inroute	input	a message is broadcast by a peer
route_out	output	a message reaches other peers

4.3.4 *Gnutella Protocol*

This atomic model implements the Gnutella protocol.

The model's external transition function can be described by the activity diagram in Figure 4.

Messages to be routed in the network are input on the input port "in_route", and are assigned a certain time-to-live (TTL): the TTL determines how many hops the message will follow in the network before being dropped. Each hop is

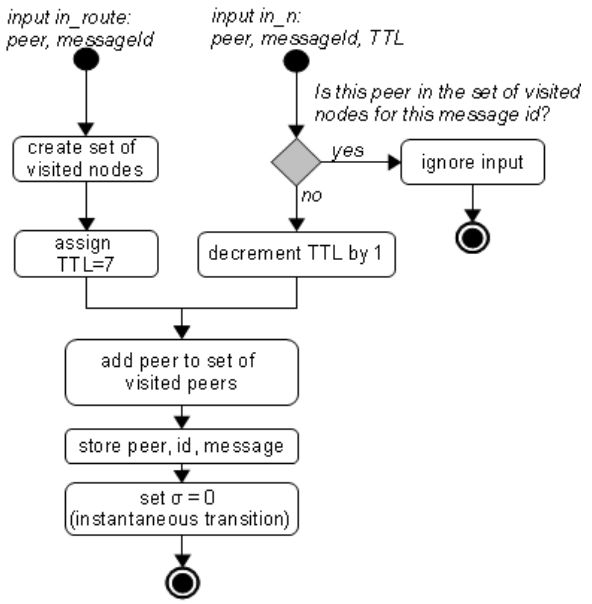

Figure 4: Gnutella Model : External Transition Function

simulated by passing the message to the LTSNetwork atomic model, through the output port "out_n" which then inputs new messages to the input port "in_n", one message for each new peer that the message reaches.

Each time the message reaches a new peer (a peer that the message hasn't previously reached), this event is marked by an output message on the "route_out" output port. The inputs and outputs of this model are summarized in the Table 7.

Table 7: Gnutella Model : Input and Output ports

Port name	I / O	Comments
route_in	input	a query is input to be routed
route_out	output	a query is output by a peer
out_n	output	the message is output from a particular peer
in_n	input	the message reaches other peers

4.3.5 *Database*

The database component maintains the data stored by all the peers. When a query is input to the network model, the database is notified of the query and its message identifier through its input port "query_in". The query is then routed through the network and successively reaches a number of peer. Each time a peer is reached, the database is notified on its "peer_in" port. These messages include a message identifier and can thus be correlated with a particular query that was input earlier. The query term is then matched to the documents stored by the peer that is reached by the query, and any queryhits are output on the "queryhit" port.

The database model's inputs and outputs are summarized in Table 8.

4.3.6 *Demultiplexer*

Table 8: Database Model : Input and Output ports

Port name	I / O	Comments
query_in	input	a query is input
peer_in	input	a peer is reached by a query
queryhit	output	a queryhit is generated as an answer to a query

The demultiplexer's role is to output messages to their destination peers. The demultiplexer has separate output ports for each peer that is connected in the network, and messages that are sent from any component in the network (in our case the database and the LTSNetwork) are input to the demultiplexer, then forwarded to the destination peers through the appropriate output ports.

The demultiplexer maintains a routing table where messages identifiers are listed with the peer identifiers that the corresponding messages should eb forwarded to.

The Demultiplexer's ports are listed in Table 9.

Table 9: Database Model : Input and Output ports

Port name	I / O	Comments
table_in	input	input to the routing table : a message id is associated to a peer
message_in	input	a message is input to be routed
connect_in	output	a connection notification is input
out_[N]	output	a queryhit for peer [N]
out_c[N]	output	a connection notification for peer [N]

4.4 Message Encoding

Messages exchanged by these different models mostly contain more than atomic pieces of information. Usually they are meant to represent network messages, and have two or up to three components to them. We encode the messages by encoding each piece of information on three digits. For example, if a message passed from the Gnutella model to the LTS network model must encode a peer identifier, a message identifier, and a time-to-live, then the message identifier is encoded in the three least significant digits, the peer identifier is encoded in digits 4 through 6, and the time-to-live is encoded in the most significant digits.

The different encodings used across the models are listed in Table 10

Table 10: Database Model : Input and Output ports

Message type	digits		
	low	mid	high
publish	doc	peer	-
query (before dispatcher)	query	peer	-
queryhit	msg id	doc	peer
peer and id	msg id	peer	-
query and id	msg id	query	-
connect or disconnect	peer1	peer2	1 for connect, 0 for disconnect

5. SUMMARY OF EXTERNAL TOOLS

This section describes the external (non-CD++) tools that we have written for this project.

- a Python script to generate the peer acquaintance graph and the files necessary for the simulation.
- a Python script to pre-process log files.
- a Java Applet to view the simulation graphically.

5.1 Generation of Simulation Files

The script inputs the number of peers, and generates a (partly random) social acquaintance graph, using the algorithm presented in [7]. The social aquaintance graph is saved in a file, and, in addition, the list of each peer's acquaintances is also saved in a specific file, to be loaded by that peer.

Then the lists of documents and queries are generated for each peer. We use a collection of ten different queries and ten different documents, such that each query matches three documents, and each document is matched by three queries. This graph of queries and matches is defined in an external file and can be easily modified for some more interesting behaviors.

The available documents are each randomly assigned to either one or two (0.5 probability each) peers; the peers chosen uniformly at random among all the peers in the network. Each peer is assigned a sequence of queries (the number of queries follows a Poisson distribution with a mean of six, and are not necessarily all different). The sequence of queries is then repeated once.

5.2 Log File Processing

A log file for a simulation run can be quite large (around 30 to 50Mb), and for this reason we have written a "preprocessing" script that extracts only the meaningful events that can be visualized in our visualizer (see below). A processed log file will only be a small fraction of the original log file (less than 1%).

5.3 Simulation Visualization

This java applet allows simulation runs to be viewed in real time in an animated graph. The graph of active peers is displayed, with the documents that they store. In addition to these nodes appearing, disappearing, and becoming connected in the graph, other events such as queries and queryhits are viewed by the relevant peers changing colors for brief durations. Please note that this tool is still under construction. Videos of the animations happening on screen can be recorded using third-party tools. This is a goal for future work, once the animations will be show more clearly interesting emergent behaviors, such as the spreading of files, etc.

6. RESULTS AND ANALYSIS

6.1 Preliminary Results, and Open Research Questions

In the current state of our project, we have a full set of tools to easily generate network models of different sizes, run simulations and visualize results.

We have generated networks and ran simulations with several hundred nodes (up to 500), and have found that the simulation is very efficient: a full-scale simulation of a 500-node network can be run in a few minutes, when the simulation time is close to one hour. This leads us to believe we can probably tweak our models to simulate much higher scale networks – several thousand peers – at a reasonable computational cost.

However, our experiments used the same minimal initial file distribution as in smaller networks (see section 5), which at this scale proved very insufficient for there to be any meaningful propagation of the files. In fact, this may be an interesting result in itself: the absolute number of files, as propagated by the activity of 90 peers, is much higher than when the files are propagated by the activity of 500 peers.

We can interpret this result as the effect of two main factors that compete to determine the dissemination of files. On one hand, a high number of peers querying for the files should create more replication from a fixed number of original copies; and network distance between querying peers and peers with the matching files only increases at a logarithmic rate with respect to the number of peers (a property of small-world networks). On the other hand, as the network distance (in terms of peer acquaintances) becomes higher, the chance that there exists a path of active nodes over that distance decreases geometrically, as it is a product of probabilities. These two factors, combined with the limited-horizon effect that the Gnutella protocol creates, seem to result in files not being propagated when there are too many peers.

We can thus identify several interesting research questions : is there a tipping point, in terms of network size, before which the replication of files increases with the peers, and after which the network becomes somehow fragmented ? If so, how can this tipping point be characterized ? Another interesting experiment would be to attempt to leverage this possible fragmentation effect, and introduce different files in different parts of the network: would they spread, within a bounded area, as files do in a smaller network ?

6.2 Future Work

As this paper describes a work in progress, we have included some weaknesses that we have identified in our existing design, which we will fix in later development stages, as well as the general direction for future evolutions.

Analyzing the structure of our models, we have found several design flaws, that we intend to fix during the next stages of development. One potential problem is that our network model is probably too closely tied to the Gnutella protocol. A network layer that would only model the transportation and connectivity between pairs of nodes might be a better approach, in case we want to experiment with other protocols. Some aspects such as the query management (in fact, the purely Gnutella aspects) could be pushed up into the peers, and the database component might be better as an entirely separate module. Fortunately, the modularity of the

modeling framework make these changes appear fairly easy.

In addition, our current approach of designing the peers' acquaintance graph, the document distribution, and more importantly the query behavior of the peers in a static, external tool before the simulation is run was a simple solution to begin, but we see necessary improvements. For example, there are good reasons to make the peers' query behavior part of the peer model. Peers are likely to change their querying behavior depending on what results they obtained form previous queries; and this is even clearer if we extend the peer behavior to include such actions as editing an existing document.

Currently, our visualization tool lets us see mostly the network graph evolve, and the query and query hits. However, these mostly serve as an intuitive validation that the model is somewhat realistic : we see, for instance, how peers with few acquaintances may stay isolated for a long time just because their acquaintances happen to be offline. This is a result of relatively sparse networks, and explains for instance why "social" applications such as instant messaging clients are so sensitive to critical masses of users, and their usage appears to follow fads more than the technical appeal of the tools themselves.

Evolution. Our long-term goal, of simulating applications such as wikis, or database-like systems, mostly requires the development of the database component, to include relationships between documents, or document metadata. For example, a wiki page may be an edit of another, and two web pages may link to one another : these would be two examples of relationships between documents.

One aspect of our research will be to experiment with parameters such as trust-related peer behavior, and the filtering effect of file-sharing networks (as discussed in section 1), to identify applications where the file-sharing paradigm may have strong added value over centralized approaches, or over structured network-based approaches.

7. CONCLUSION

The DEVS formalism has proved here to be an efficient tool to model network applications. Its modularity has made the generation of large-scale networks an easy task, using a scripting approach. With the experience that we have built up during this project we have identified some improvements in our design, that should make our simulation tool on par with other tools developed over many years by teams of dedicated researchers.

8. REFERENCES

[1] I. Baumgart, B. Heep, and S. Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, pages 79–84, May 2007.

[2] J. Davidsen, H. Ebel, and S. Bornholdt. Emergence of a Small World from Local Interactions: Modeling Acquaintance Networks. *Physical Review Letters*, 88(12):128701+, March 2002.

[3] A. Davoust. Collaborative knowledge construction in a peer-to-peer file sharing network. Master's thesis, Carleton University, 2009.

[4] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[5] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst. Characterizing the query behavior in peer-to-peer file sharing systems. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 55–67, New York, NY, USA, 2004. ACM.

[6] T. Li, M. Chen, D.-M. Chiu, and M. Chen. Queuing Models for Peer-to-peer Systems. April 2009.

[7] R. Toivonen, J.-P. Onnela, J. Saramaki, J. Hyvonen, and K. Kaski. A model for social networks. *Physica A: Statistical and Theoretical Physics*, 371(2):851–860, 2006.