Simulating Gnutella P2P Network

Student Project (Studienarbeit) Final Report

Wei Lu (w.lu@tu-harburg.de)

Matrikel-Nr. 27462

Master of Information and Communication Systems

Communication Networks

Technical University of Hamburg-Harburg

25/02/2005
Abstract

In this project, we implement a typical peer-to-peer (P2P) protocol, Gnutella, for ns-2 and Ptolemy simulator. The protocol and implementation details are discussed.

Many single user related and entire network related performance characters, e.g. relationships between traffic volume and increasing size of network, are studied by simulations. Great difference between Gnutella version 0.4 and 0.6 is found and analyzed. Specific characteristics of structured P2P network, Gnutella 0.6, are investigated, e.g. Ultraceer election and organization.

Finally, limits of our approach are discussed. Promising future work to overcome the limits is also proposed.

Keywords:

Peer-to-Peer, Gnutella, Simulation
Contents

1 P2P network and Gnutella protocol .......................................................... 6

2 Gnutella protocol version 0.6 ..................................................................... 7
  2.1 Ultrapeer Election .............................................................................. 8
    Leaf to Ultrapeer .............................................................................. 10
    Leaf to Shielded Leaf ..................................................................... 10
    Leaf to Unshielded Leaf .................................................................. 11
    Ultrapeer to Ultrapeer ................................................................... 11
    Ultrapeer to Ultrapeer, with Leaf Guidance ................................... 11
  2.2 Message Dispatching ...................................................................... 12
  2.3 Query Routing .............................................................................. 12

3 Implementation ...................................................................................... 13
  3.1 ns-2 ............................................................................................... 13
    3.1.1 ns-2 Introduction .................................................................. 13
    3.1.2 Architecture of Gnutella protocol implementation .............. 14
  3.2 Ptolemy ........................................................................................ 16
  3.3 Simulation setup and data process scripts ...................................... 19
    3.2.1 Simulation setup .................................................................. 19
    3.3.2 Data processing .................................................................... 24

4 Simulations .......................................................................................... 27
  4.1 Configurations .............................................................................. 27
4.1.1 Network ..............................................................................................27
4.1.2 User Behavior Model .........................................................................28
4.2 Ultrappeers .............................................................................................29
4.3 Total traffic ............................................................................................32
4.4 Single traffic ..........................................................................................35
4.5 Conclusions ..........................................................................................38

References ......................................................................................................39

List of Figures

Figure 1, general concept of P2P network .......................................................6
Figure 2, Schematic drawing of Gnutella 0.4 ...................................................7
Figure 3, Gnutella 0.6 network topology .......................................................7
Figure 4, Runtime object layout of a simple simulation .................................13
Figure 5, UML of main classes of Gnutella protocol implementation ..........15
Figure 6, Architecture of Gnutella on Ptolemy .............................................17
Figure 7, Sequence Diagram of Timer handling on Ptolemy port ..................18
Figure 8, UML class diagram of main classes of Ptolemy port .....................18
Figure 9, ns-2 Trace Format Example ..........................................................24
Figure 10, data process in message level ......................................................25
Figure 11, data process in packet level ........................................................26
Figure 12, Germany network (17 nodes) ..............................................................27

Figure 13, Number of ultrapeers vs. simulation time elapsed. .........................30

Figure 14, Number of ultrapeers vs. network size (total number of peers) ........30

Figure 15 Variation of network structure with different peer availability ..........32

Figure 16, Average signaling traffic per MODEM user .....................................33

Figure 17, Average signaling traffic per DSL user .............................................33

Figure 18, Average signaling traffic per Bootstrap user ..................................34

Figure 19, Total signaling traffic comparison ..................................................35

Figure 20, Signaling traffic of a single Gnutella 0.4 user ....................................36

Figure 21, Signaling traffic of a single Gnutella 0.6 user ....................................37

Figure 22, Signaling and download traffic of a single Gnutella 0.6 user. (Log-log
scaled) .............................................................................................................37

Figure 23, comparison of signaling and synthesis (signaling + download) traffic
of single Gnutella 0.6 user. (Log-log scaled) ..................................................38
1 P2P network and Gnutella protocol

A Peer-to-Peer (P2P) network is an overlay network over the existing physical internet in application layer. The main purposes include file sharing and media content delivering.

![Figure 1, general concept of P2P network](image)

Since Napster music sharing network, many P2P networks are created, like Gnutella, KaZaa, Freenet, JXTA, etc. Each network develops its own protocol. Because of its open nature, the Gnutella network and protocol are frequently targeted by academic research. Measurements and statistics are widely available as bases.

There are two versions of Gnutella protocol. The basic and widely deployed one is version 0.4. Gnutella 0.4 is a Distributed Index Flooding Architecture (DIFA). When a peer joins the network, it attempts to connect to Bootstrap servers to fetch initial peer address list. Once knew other peers, PING messages are sent to each of them. The latter relays PING messages to all known peers in a flooding manner. The flooding scope is controlled by Time-To-Live (TTL). PONG messages are reversely routed to the PING initiator. The initiator establishes connections to replied peers. When a user wants to find shared files, Query messages carrying keywords are flooded to all known-hosts. Once a peer has a matched file, a QUERYHIT message is reversely routed to the query initiator. The matched file can be downloaded via HTTP/TCP
protocol from QueryHit sender by Query sender directly.

![Gnutella 0.4 Network Diagram](image1)

Figure 2, Schematic drawing of Gnutella 0.4

### 2 Gnutella protocol version 0.6

As the signaling message load in the Gnutella 0.4 network is very high and doesn’t scale when network size increases, protocol enhancements to reduce the signaling overhead have been proposed and resulted in a new version of the Gnutella protocol, Gnutella 0.6.

The main solution is to create an additional hub-like network layer as follows.

![Gnutella 0.6 Network Topology](image2)

Figure 3, Gnutella 0.6 network topology

Peers in Gnutella 0.6 are divided into Leaf and Ultapeer roles to provide the hub
functionality. Ultradees establish a higher hierarchy level in which they form a pure
unstructured P2P network, i.e. are connected to each other directly via TCP
connections. Each leaf-node maintains in general only one connection to one
Ultradeer, whereas one Ultradeer is connected to several leaf nodes and several
Ultradeers.

Besides the enhancements on Bootstrapping, Querying and Flow Control,
obviously, what could affect our interested merits most is the introduction of Ultradeer
mode.

The ultradeer extension is implemented based on Gnutella 0.4 for ns-2 simulator
referring to the Ultradeer Proposal v1.0\(^2\) discussed as follows.

2.1 Ultradeer Election

Gnutella 0.6 peers regularly determine whether they are eligible to be ultradeers
(“ultradeer capable”) by looking at uptime, operating system functionality, bandwidth,
etc. Except uptime, other factors are not feasible to evaluate in our simulation. We
decide to ignore them to simplify the problem. Another solution is to ignore the
election protocol but assign peer role manually. The obstacle is that Gnutella 0.6 is not
deployed widely and lack of measurements and statistics as references. Analytical
model to derive a theoretical structure is not available either. So a simplified criteria is
used. Because every host is in an ON-OFF process except Bootstrap hosts with
bandwidth greater than 1000Mbps, the criteria is that the host who’s going to be ON
for longer than mean up-time (60 sec) is eligible to be ultradeer.

In the beginning, the number of leaves of an ultradeer is not limited. However, at
the first moment of a simulation, most hosts connect to few bootstraps for a start, then
they become the leaves of the bootstraps because it’s obviously that a bootstrap has
higher qualification to be ultradeer than other hosts. It results that all bootstraps
become ultradeers but none of other hosts. Of course, this doesn’t reflect the real
world. The reason is partly because of the small size of simulated network. However,
computer memory limits the size of simulated network, e.g. 2GB RAM supports simulation of about 1000 hosts at most. In order to mimic the real world, the maximum number of leaves of an ultrapeer is limited to 32 according to an empirical suggestion. I ever tried a number of 20. The simulation result looks similar to 32’s. Further study on this could be interesting for no literature on this topic can be found yet.

It is important to distinguish between hosts that are ultrapeer capable and hosts that are actually ultrapeers:

- Ultrapeer: hosts that are ultrapeer capable and not a shielded leaf node to an ultrapeer. Note that an ultrapeer does not necessarily have leaf connections.

- Leaf: hosts that have only a single, routed connection to an ultrapeer. Note that leaves may actually be ultrapeer capable.

There are some header fields for Gnutella 0.6 hosts to indicate capabilities and role intents when connect to each other. They are not implemented in the Gnutella messages for ns-2 simulator. For efficiency, corresponding methods to enquire the host status are instead. We use the header field name in the specification as identifications in the following discussion.

Important header fields are:

- **X-Ultrapeer**, tells whether a host plans on acting as an ultrapeer (if true) or a shielded node (if false). It’s implemented as
  Ultrapeer Capable (up time > mean up time) && not leave yet.

- **X-Ultrapeer-Needed**, used to balance the number of ultrapeers. This is discussed in detail in section “Ultrapeer to Ultrapeer, with Leaf Guidance” below. It’s implemented as
  if (X-Ultrapeer)
  X-Ultrapeer-Needed = (number_of_leaves<=0);
  else
**X-Ultraceer-Needed = is not leave.**

When hosts connect, they express their capabilities and intents to negotiate the mode of ultrapeer or leafnode. The various cases implemented are as follows.

**Leaf to Ultrapeer**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>iX-Ultraceer: False</td>
<td>X-Ultraceer: True</td>
</tr>
<tr>
<td></td>
<td>X-Ultraceer-Needed: false</td>
</tr>
</tbody>
</table>

If client is already leaf node, stop the connection simply. If not, continue the process like in Gnutella 0.4 and the server becomes ultrapeer of client. We say it that client is shielded by server.

**Leaf to Shielded Leaf**

<table>
<thead>
<tr>
<th>Client (A)</th>
<th>Server (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultraceer: False</td>
<td>X-Ultraceer: False</td>
</tr>
<tr>
<td></td>
<td>[terminates connection]</td>
</tr>
</tbody>
</table>

B is a leaf node, the connection terminates. In future, an X-Try-Ultrapeers option should be implemented for B to redirect A to B’s ultrapeer.

---

1 Other fields are omitted.
Leaf to Unshielded Leaf

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultrapeer: False</td>
<td>X-Ultrapeer: False</td>
</tr>
<tr>
<td></td>
<td>X-Ultrapeer-Needed: true</td>
</tr>
</tbody>
</table>

Sometimes nodes will be ultrapeer-incapable and unable to find an ultrapeer. In this case, they behave exactly like old, unrouted Gnutella 0.4 connections.

Ultrapeer to Ultrapeer

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultrapeer: True</td>
<td>X-Ultrapeer: True</td>
</tr>
<tr>
<td>X-Ultrapeer-Needed: false</td>
<td>X-Ultrapeer-Needed: false</td>
</tr>
</tbody>
</table>

When two ultrapeers meet, both set X-Ultrapeer to true. If both have leaf nodes, they will remain ultrapeer mode after the interaction.

Ultrapeer to Ultrapeer, with Leaf Guidance

<table>
<thead>
<tr>
<th>Client (A)</th>
<th>Server (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultrapeer: True</td>
<td>X-Ultrapeer: True</td>
</tr>
<tr>
<td>X-Ultrapeer-Needed: true</td>
<td>X-Ultrapeer-Needed: false</td>
</tr>
</tbody>
</table>

Consider the case of an ultrapeer, A connecting to an ultrapeer B, or the opposite. A’s X-Ultrapeer-Needed is false, B’s X-Ultrapeer-Needed is true. X-Ultrapeer-Needed
indicates if host has no leaves currently. The one who has no leaves will become leaf node of the other. When both already have leaves, the client becomes leaf of server and drops original connections. Its previous leaves become unshielded nodes. When the one chosen to be ultrapeer has already too many leaves, both peers remain ultrapeer mode and the connection is established.

2.2 Message Dispatching

In Gnutella 0.4, Ping messages are flooded to every connected host. In Gnutella 0.6 implementation, leaf node will not send Ping message to other hosts except its shielding ultrapeer. Ultrapeer only dispatches Ping message to other connected ultrapeers.

2.3 Query Routing

In our Gnutella 0.6 implementation, leaf node will never spread Query message. Ultrapeer only dispatches Query message to his leaf nodes that definitely have a match to the query. This is a little different to the actual situation because the query routing algorithm can never be 100% precise. And the simulation code checks the database of leaf nodes to know if they have matched files. This process can be done by C/C++ function invoking without actually generating network traffic. This part of network traffic between ultrapeer and leaf nodes is ignored because it should be a very little portion.
3 Implementation

We have implemented Gnutella 0.4 and Gnutella 0.6 protocol for ns-2\textsuperscript{4} network simulator and Ptolemy\textsuperscript{5} multi-purpose simulator framework. Both simulators provide an open source Object-Oriented framework in C++ to be extended by sub-classing in C++. Comparing to commercial simulation solutions like OPNET\textsuperscript{6}, ns-2 and Ptolemy are lack of ready-for-use result processing and analyzing functionalities. Quite a lot post simulation processing scripts in Perl, Bash and Matlab are written in our work.

3.1 ns-2

3.1.1 ns-2 Introduction

Ns-2 simulator is a rich featured, well tuned and widely accepted discrete event simulator for communication network research. It’s equipped with a well designed heap based event scheduler that is very suitable for handling huge amount of events in complex simulations. Its Object-Oriented architecture in C++ and scripting interface in TCL make it easy to extend new functions. Further more, it includes a fully implemented TCP/IP stack which is rarely seen in other simulation tools. This is of great help for our work.

Figure 4, Runtime object layout of a simple simulation.
The internal structure of ns-2 is depicted as above. The bi-directional arrows represent the flow of the control messages. The thick unilateral arrows are the data flow between software objects. The thin unilateral arrows represent the virtual simulated network transmission path.

In every simulation task, there is only one global Simulator object. It controls the whole simulation. Class Application is the representation of the application layer of the network. It generates the traffic and consumes it. Classes Agent is the representation of the protocol stack. It can be understood as the protocol software in the hosts. Class Node is the presentation of physical network nodes, e.g. networked computer, switch, router, etc. Link presents the physical network link with packet queues.

Simulation scenario is described by TCL scripts. Main program of ns-2 is implemented as a TCL interpreter indeed. The interested metrics in the simulation can be recorded by tracing packet events in Links and state variable changes in Agents and saved to disk. Customer extensions can of course record their own information in any ways. For example, our Gnutella protocol implementation outputs internal state information to standard output and redirect to a text file for post analysis.

3.1.2 Architecture of Gnutella protocol implementation

Gnutella protocol is implemented as an application layer component.
It’s a bit uncommon that GnutellaApp is not implemented as a child class of ns-2 Application class like what usual ns-2 extension does. The main reason is that the TcpAgent is not created in TCL simulation setup script for each Application as usual. Each GnutellaApp maintains multiple GnutellaConnection objects and the latter maintains TcpAgent objects those are created dynamically in runtime to represent multiple TCP connections. GnutellaApp still extends TclObject to be created in setup scripts. GnutellaServent concretely implements the Gnutella protocol and invoked by GnutellaApp when packet event is received. GnutellaGlobalRegistrationOffice is a singleton helper class to generate shared file related random parameters.

Another unusual point is that Gnutella Message is not implemented as ns-2 Packet and delivered in simulation events. Each GnutellaApp contains a local message list. The message receiver can get the pointer of sender GnutellaApp via message incoming GnutellaConnection and its Node pointer and then Node’s app_ pointer. The receiver can access the sender GnutellaApp’s local message list to get the message content directly. The original ns-2 Node class has to be modified to add pointer to GnutellaApp object. This is not a neat solution but gives better performance. Another disadvantage is that standard ns-2 tracing mechanism can not apply on Gnutella Message statistic because message content is not transmitted in the simulation at all.
In GnutellaServent class, information about Gnutella protocol state and message transmission is output to stdout with one second interval. We can execute the simulation and redirect the output to a text file for post-processing. Some of the post-processing scripts will be discussed later.

Differences between Gnutella 0.4 and Gnutella 0.6 mainly exist in message receiving and relaying routines. Two versions can be selected by conditional compiling option GNUTELLA_V06. Download traffic is injected by invoking TCL scripts inside C++ codes after receive a QueryHit message.

### 3.2 Ptolemy

Ns-2 is very suitable for background the batch execution with a high efficient scripting interface. However, there is a disadvantage that it has no interactive graphic interface. Only an animation program for replaying finished simulation is provided. This gives not much help for simulation scenario creation and debug.

Ptolemy is another simulation framework including discrete event scheme. An interactive GUI is supported by Ptolemy. And the communication networks department in TUHH developed a simple TCP/IP stack and other protocol implementations for it. It’s valuable to port the Gnutella protocol to Ptolemy too.

With an adaptor pattern, Gnutella implementation can be compiled with Ptolemy without modification to major part of the source code.

The following figure depicts the over architecture of the Gnutella protocol in Ptolemy.
There are some difficulties for the implementation.

Firstly, the scheduling and event mechanism is quite different between two simulators. In ns-2, there is only one central schedule controlling the simulation. It’s a Singleton pattern and can be accessed by static method like Scheduler::instance(). In Ptolemy, every Star (basic simulation element in Ptolemy concept) may have a scheduler because it may belong to different Domains in which the scheduling scheme can be different. The current solution is to assign the pointer of the scheduler belonging to current Star to a static variable to be accessed. This is just a temporal solution before we find a better way.

Secondly, the timer required by the protocol is supported by scheduler in discrete event simulation. We implement it in Ptolemy by sub-classing from DERepeatStar that has a feedback wire to itself to implement a self-timer periodically. A sequence diagram for this scenario can be represented as follows.
DEGnutellaServent (DEGnutellaServent.pl) and DETCPMultiSocket (DETCPMultiSocket.pl) are two new Ptolemy Stars. Every DETCPMultiSocket can handle multiple active opened connections and multiple passive opened connections at one server port. The connections_ is an array of TCPSocket class containing the pointer of active opened TCP connections, except the last item points to a passively opened server port listening for client connections. The array index is the local port of the connection. The passively opened TCP connections are maintained in a map serverConns as values. The keys are the remote address and remote port pairs. A 64
bit integer array remoteAddrPorts is used to allocate IDs for passive opened connections. The index is the ID value. When the item value is 0, the ID is not allocated, when the value is the remote address and remote port pair, the index is allocated as the ID for corresponding connection. DEGnutellaServent is a Ptolemy operational component (Star in Ptolemy concepts) as proxy of the concrete implementation of Gnutella protocol, GnutellaServent (gnutella_protocol.cc/.h)

GnutellaApp (gnutella_app.cc/.h) is the interface connecting DEGnutellaServent and GnutellaServent. In ns-2 simulator, it’s application layer representation and attaches to Agent. The reason why this class still exists in Ptolemy implementation is to provide an identical interface for GnutellaServent. Eventually, the DEGnutellaServent Star will implement the interface of GnutellaApp and expose to GnutellaServent. The GnutellaApp will be removed to save memory.

Main logic of Gnutella protocol exists in GnutellaServent. This class can be compiled in both ns-2 and Ptolemy without specific modification. To achieve this, the ns-2 classes those are used by GnutellaServent are re-implemented with the facility of Ptolemy, e.g. Node, Scheduler, TimerHandler, etc.

3.3 Simulation setup and data process scripts

3.2.1 Simulation setup

Simulation scenarios are constructed in TCL scripts. Snippets are discussed as follows.

```tcl
# parameter to roughly control the network size
set networkscale [lindex $::argv 0]
# random seed
if { $::argc > 1 } {
    set rndseed [lindex $::argv 1]
}
# record physical topology
set topfile [open 17nodes_19_$networkscale-trace_some.top w]
# Create a new simulator object.
set ns [new Simulator]
```
# Heap scheduler is high efficient in complex simulation
\$ns use-scheduler Heap

# Create wired core nodes.
set node(1) [\$ns node]
set node(2) [\$ns node]
...
set node(17) [\$ns node]

# Create links between nodes.
\$ns simplex-link $node(17)$ node(16) 1000.000Mb 20.000000ms DropTail
\$ns simplex-link-op $node(17)$ node(16) queuePos 0.5
\$ns simplex-link-op $node(17)$ node(16) color black
\$ns simplex-link-op $node(17)$ node(16) orient 108.9deg

# Set Queue Properties for link 17->16
[[\$ns link $node(17)$ node(16)]] queue] set limit_ 500
\$ns simplex-link $node(16)$ node(17) 1000.000Mb 20.0000000ms DropTail
\$ns simplex-link-op $node(16)$ node(17) queuePos 0.5
\$ns simplex-link-op $node(16)$ node(17) color black
\$ns simplex-link-op $node(16)$ node(17) orient 288.9deg

# Set Queue Properties for link 16->17
[[\$ns link $node(16)$ node(17)]] queue] set limit_ 500

# Set Queue Properties for link 2->1
[[\$ns link $node(2)$ node(1)]] queue] set limit_ 500
\$ns simplex-link $node(2)$ node(1) 1000.000Mb 20.0000000ms DropTail
\$ns simplex-link-op $node(2)$ node(1) queuePos 0.5
\$ns simplex-link-op $node(2)$ node(1) color black
\$ns simplex-link-op $node(2)$ node(1) orient 87.9deg

# Set Queue Properties for link 1->2
[[\$ns link $node(1)$ node(2)]] queue] set limit_ 500

#setup trace on links
set trfile25 [open 17nodes_19_$networkscale-trace_some_1-3.tr w]
\$ns trace-queue $node(1)$ $node(3)$ $trfile25
set trfile26 [open 17nodes_19_$networkscale-trace_some_1-2.tr w]
\$ns trace-queue $node(1)$ $node(2)$ $trfile26
set trfiler25 [open 17nodes_19_$networkscale-trace_some_r1-3.tr w]
\$ns trace-queue $node(3)$ $node(1)$ $trfiler25
set trfiler26 [open 17nodes_19_$networkscale-trace_some_r1-2.tr w]
\$ns trace-queue $node(2)$ $node(1)$ $trfiler26
# set up the Gnutella Global office
set go [new GnutellaGlobalRegistrationOffice]
if { $#::argc > 1 } {
    $go seed $rndseed
    puts "ItINFO: set seed to $rndseed\n"
} else {
    $go seed 32145
}

# load emperical file size distribution
$go readFile gnutella/gnutella.files

# To implement the second layer of topology, implement power-law distribution
set u [new RandomVariable/Pareto]
$u set avg_ $networkscale
$u set shape_ 1.5

# initialize a general uniform variable to randomly select access link speed
set uniform [new RandomVariable/Uniform]
$uniform set min_ 0
$uniform set max_ 1

# The following loop implements the 2nd layer of the heirarchy. The number of nodes per core-node is power-law distributed, the power is controlled with the shape parameter
set counter 18
set counter2 1
set appcounter 1
set numConnections 0
for {set x 1} {x <= 17} {incr x} {
    puts "node([expr ($x) ]).\[$node([expr ($x) ]) address?] link to:"
    set z [expr round([u value])]
    for {set y 1} {y <= $z} {incr y} {
        set node([expr ($counter)]) [ns node]
        puts " _node([expr ($counter)]),\[$node([expr ($counter) address?])"
        set emp [expr [$uniform value]]
        # monitor specific MODEM and DSL user
        if {$counter == 18} {
            set userSpeed 100.000kb
            set numConnections 12
        } else {
            if {$counter == 22} {
                set userSpeed 1.000Mb
                set numConnections 30
            } else {
                if {$emp <= 0.70} {
                    set userSpeed 100.000kb
                } else {
                    if {$emp < 0.70} {
                        set userSpeed 100.000kb
                    } else {
                        # code snippet...
                    }
                }
            }
        }
    }
}

set numConnections 12
} else {
    if {$emp <= 0.94 } {
        set userSpeed  1.000Mb
        set numConnections 30
    } else {
        if {$emp <= 0.985 } {
            set userSpeed  2.000Mb
            set numConnections 75
        } else {
            if {$emp <= 0.996 } {
                set userSpeed  4.000Mb
                set numConnections 120
            } else {
                if {$emp <= 1.000 } {
                    set userSpeed  1000.000Mb
                    set numConnections 800
                }
            }
        }
    }
}

# set up the link
$ns duplex-link $node($x) $node([expr ($counter)]) $userSpeed 20.000000ms
DropTail
puts " link node([expr ($x)]).[$node($x)] address? and node([expr ($counter)]).[$node($counter)] address? with delay $userSpeed"
# Set Queue Properties for the link with QueueSize = 500
[[[$ns link $node($x) $node([expr ($counter)])]] queue] set limit_ 500
# Attach GnutellaApp agent
set app([expr ($appcounter)]) [new GnutellaApp $ node([expr ($counter)])]
$numConnections $go
if {$numConnections > 120} {
    $go regBootStrapServent $node([expr ($counter)])
    puts " app([expr ($appcounter)]) registered for bootstrap"
    puts Stopfile "$node([expr ($counter)]) address?] 1 bootstrap [$node([expr ($x)]) address?]"
} else {
    if {$numConnections<=12} {
        puts Stopfile "$node([expr ($counter)]) address?] 3 modem [$node([expr ($x)]) address?]"
    } else {
        puts Stopfile "$node([expr ($counter)]) address?] 2 DSL [$node([expr ($x)]) address?]"
    }
}
# monitor selected MODEM and DSL user

```tcl
if {$counter == 18} {
    set trmodem [open 17nodes_19_${networkscale}-trace_some_r1-modem.tr w]
    $ns trace-queue $node($counter) $node(1) $trmodem
    $app($appcounter) monitor-this #$app($appcounter) set-min-on 100000
} else {
    if {$counter == 22} {
        set trdsl [open 17nodes_19_${networkscale}-trace_some_r1-dsl.tr w]
        $ns trace-queue $node($counter) $node(1) $trdsl
        $app($appcounter) monitor-this #$app($appcounter) set-min-on 100000
    }
    incr counter
    incr appcounter
}
```

close $topfile

for {set a 1} {$a < $appcounter} {incr a} {
# Traffic with download or singlaing only
    $app($a) enable-download
# Start Gnutella protocol
    $ns at [expr ($a)] "$app($a) start"
}

# clean up routine when simulation complete
proc finish {} {
    $ns flush-trace
    close $trfile25
    close $trfile26
    close $trfiler25
    close $trfiler26
    exit 0
}

# Run the simulation
$ns at 500.00010 "finish"
$ns run
```

List 1, Simulation setup script
3.3.2 Data processing

There are 3 sources of data to be processed after simulation to get interested metrics. As we can see in List 1, Simulation setup script, physical topology information including access speed, linkages, etc. is output to stdout by TCL statements. GnutellaServer outputs the internal state information including counts of sent messages, number of opened connections, etc., to stdout. We capture these two sources into text file by output redirection in command line. The 3rd type of trace data is recorded by Link trace supported by ns-2 build-in function, as we can see in above "$ns trace-queue" statements.

Physical topology information is used to calculate traffic rates of given access type. Gnutella internal states can be used to calculate traffic metrics in message level. Awk is used to filter the desired columns of data, like the following command:

```bash
cat $1 | grep "ON" | grep "bootflag = 1" | awk '{printf "%s \n",$28}' > $1_time_1.dat
```

Matlab is used to calculate mean and variance values. However, it’s not easy to pass parameters in command line for Matlab scripts execution, Perl is utilized as complement.

Ns-2 link trace file is in a format like this:

<table>
<thead>
<tr>
<th>event</th>
<th>time</th>
<th>from node</th>
<th>to node</th>
<th>pkt type</th>
<th>pkt size</th>
<th>flags</th>
<th>fid</th>
<th>src addr</th>
<th>dst addr</th>
<th>seq num</th>
<th>pkt ld</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>1.3556</td>
<td>3</td>
<td>2</td>
<td>ack 40</td>
<td>1</td>
<td>3.0</td>
<td>0.0</td>
<td>15</td>
<td>201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1.3556</td>
<td>2</td>
<td>0</td>
<td>ack 40</td>
<td>1</td>
<td>3.0</td>
<td>0.0</td>
<td>15</td>
<td>201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1.3556</td>
<td>2</td>
<td>0</td>
<td>ack 40</td>
<td>1</td>
<td>3.0</td>
<td>0.0</td>
<td>15</td>
<td>201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>1.3557</td>
<td>6</td>
<td>2</td>
<td>tcp 1000</td>
<td>1</td>
<td>0.0</td>
<td>3.0</td>
<td>29</td>
<td>199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1.3557</td>
<td>6</td>
<td>2</td>
<td>tcp 1000</td>
<td>1</td>
<td>0.0</td>
<td>3.0</td>
<td>29</td>
<td>199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1.3557</td>
<td>6</td>
<td>2</td>
<td>tcp 1000</td>
<td>1</td>
<td>0.0</td>
<td>3.0</td>
<td>29</td>
<td>199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1.3561</td>
<td>2</td>
<td>1</td>
<td>cbr 1000</td>
<td>2</td>
<td>1.0</td>
<td>3.1</td>
<td>157</td>
<td>207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1.3561</td>
<td>2</td>
<td>1</td>
<td>cbr 1000</td>
<td>2</td>
<td>1.0</td>
<td>3.1</td>
<td>157</td>
<td>207</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9, ns-2 Trace Format Example
Besides Perl scripts, trpr tool is used to extract information from trace files.

Each round of our execution includes iterations of several simulations and each simulation produces several text and trace files. We use a batch file to arrange the simulation automatically. The simulation results in huge volume of Giga-bytes and distributes in several files. The complete data processing is separated into several scripts in Bash, Perl and Matlab. It’s shown in the following figures.

Figure 10, data process in message level
Figure 11, data process in packet level

1. alltr2dat.sh
2. trs2dat.pl
3. mean_var_links.pl
4. outproc_time.sh
5. mean_variance_link_1
   7node19.m
6. plot_link_traffic_peers.m

1. trproc.pl
2. trs2dat.pl
3. outproc_time_link.pl 1 [11 14]
4. mean_var_links.pl 1 [11 14]
5. mean_variance_corelink_1
   7nodes_19.m
6. mean.pl var.pl
4 Simulations

With Gnutella 0.6 implementation on ns-2 simulator, we have done several simulations which validate the implementation and reveal some Gnutella traffic characteristics.

4.1 Configurations

4.1.1 Network

The simulated network is based on a Germany Network with 17 core nodes depicted as follows.

![Germany network (17nodes)](image)

Between core nodes, duplex links with 1000.000Mbps bandwidth and 20ms propagation delay are maintained. The queue capacity is set to 500 packets. On core nodes, only IP packet switching functionality is configured. Secondary edge nodes
which are not depicted in above figure are connected to core nodes with one duplex link and host Gnutella protocol clients. According to measurement\textsuperscript{10} of Kazza network, link capacity and Gnutella connection threshold are set as follows.

Table 1 User access model

<table>
<thead>
<tr>
<th></th>
<th>MODEM</th>
<th>DSL1</th>
<th>DSL2</th>
<th>DSL3</th>
<th>Bootstrap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage</td>
<td>70%</td>
<td>24%</td>
<td>4.5%</td>
<td>1.1%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Capacity</td>
<td>100kbps</td>
<td>1Mbps</td>
<td>2Mbps</td>
<td>4Mbps</td>
<td>1000Mbps</td>
</tr>
<tr>
<td>Max Connections</td>
<td>12</td>
<td>30</td>
<td>75</td>
<td>120</td>
<td>800</td>
</tr>
</tbody>
</table>

According to study\textsuperscript{11}, node connectivity follows a power law for nodes with more than 10 connections. Number of edge node connected to each core nodes are drawn by Pareto distribution with shape parameter of 1.5.

4.1.2 User Behavior Model

According to study\textsuperscript{12}, all peers are set in ON/OFF process. Peer on-time is log-quadratically distributed with mean value of 60 second. Peer off-time is negative exponentially distributed with mean value of 105 second. Every 25 seconds, peers that maintain many connections drop oldest connections.

The file sizes shared by each peer are determined by an empirical distribution\textsuperscript{13}. Generally, 88% files are in between 0 to 10 Mega Bytes, 5% are in between 10 to 100 MB, 7% are in between 100MB to 1GB. Small (< 10MB) files (music) make up of less than 15% of total bytes but more than 88% request, large (>100MB) files (movies) compose more than 65% bytes but less than 5% request.

Study\textsuperscript{14} gives that 66% peers are free-riders sharing nothing, 73% peers share 10 or less files, top 1% peers share 35% files.
The following results are based on above configurations.

4.2 Ultraceers

Our main objective is to comparing the difference between Gnutella 0.4 and Gnutella 0.6. The source of the difference is the introduction of ultrapeer. First of all, we analysis the number of ultrapeers evolving in the simulations.

As discussed in early section, the election and organization of ultrapeers are fully decentralized. There is no central control of the role of a specific peer, or the global percentage of ultrapeer. Everything is determined by the election protocol according to the capability of a specific peer in respect of computing power, shared resource, network connectivity and the regional network status like role of neighbors. However, the published Gnutella 0.6 protocol specification is quite open in this area. Most concrete algorithm, criteria, parameters are created by implementers. For example, the number of leaves of an ultrapeer should be limited obviously. This parameter affects the cluster (An ultrapeer with its leaves) size reasonably. The specification gives no suggestion. The most important two clients, LimeWire and BearShare set it to 30, 45 respectively.

Our implementation is much simplified. Ultraceer criteria is simply uptime above average. The maximum leave number is reduced to 20 considering the small network size. Later we found that the number of 30 produces similar simulation result.

The following figure depicts the variation of number of ultrapeers with simulation time elapsed. Network size is 1452 nodes. With different network size, the curves look similar. It shows that the process takes about 100 seconds to reach stable. The spikes are caused by the fix interval of 25 seconds with which rich connected peers, usually being ultrapeer, clean their oldest connections. Once ultrapeers drop connections to leaves, some of these leaves who meet the ultrapeer criteria change their roles to ultrapeer. So the total number of ultrapeer increases suddenly. After negotiate with previous rich connected ultrapeers, some of these new ultrapeers give
up their ultrapeer role and become leaves again. Then the network evolves to stable state.

Figure 13, Number of ultrapeers vs. simulation time elapsed.

Mean numbers of ultrapeers are plotted comparing to the total number of peers as follows. The percentage of ultrapeer in total peers keeps constant to 5% roughly.

Figure 14, Number of ultrapeers vs. network size (total number of peers)

This phenomenon could be further explained by developing an analytical model. However, under the time constrain, it can only be done in future. Lack of well established analytical model is a main obstacle of current P2P protocol design and network study. The capability of simulation methodology is limited because it’s
difficult to simulate large networks under current CPU speed and RAM volume constrains. A good analytical model may help to answer two questions.

1. What’s the globally optimal topology? (Expressed by Percentage of Ultrapeer, Cluster size, Degree distribution, etc.)

2. How to locally approach the optimal topology? (Giving Ultrapeer criteria, Signaling/Content Query routing algorithm)

There is no well accepted analytical model for structured P2P network because of the complexity of the problem. For example, there are many parameters and variables including ultrapeer percentage/cluster size, degree distribution, message TTL, Query rate, content replication ratio, etc. The optimization objects are not simple either. The single host loading, total network loading, quality of content query result, etc., should be considered. An extended Random Graph model\textsuperscript{16} looks promising. Another model\textsuperscript{17} gives a recursive approach that could be solved by simulated annealing method. Our simulation result could be evaluated too.

Besides our simple algorithm, most implementations are based on intuition and experience. The local ultrapeer election process and organization protocol could be supervised by a good analytical model. Yang\textsuperscript{18} gives rules of thumb for network and protocol design. Cluster head selection in ad-hoc and sensor networks\textsuperscript{19} \textsuperscript{20} can be referenced too.

Most of previous models are based on static network. Our little experiment as follows indicates that the dynamic factors of network also matter.
Figure 15 Variation of network structure with different peer availability

The upper plot shows that when mean uptime increases, number of ultrapeers increases. The lower plot shows that the network takes more time and produces more traffic to reach stable. Then the total traffic in stable network slightly increases when number of ultrapeer increases.

4.3 Total traffic

We repeated the simulations of Gnutella 0.4 and Gnutella 0.6 for various network sizes from 300 to 1000 peers. Total number of Gnutella Messages, including Ping, Pong, Query and QueryHit, sent during 500 seconds simulation time is measured to compare the traffic amount in Gnutella 0.4 and Gnutella 0.6.

In the following figures, different access speeds, MODEM, DSL and Bootstrap, are measured individually. Messages sent by each type of access are counted and averaged to number of online users with corresponding access type. The curves of Gnutella 0.4 are shorter because the simulations with larger networks are out of
memory on our computers. Gnutella 0.6 produces much less traffic. It’s possible to do simulation of larger network with the same computer facility.

![Figure 16, Average signaling traffic per MODEM user](image16)

In Gnutella 0.6, average MODEM user produces only about 1/35 traffic amount of in Gnutella 0.4. When the network size becomes larger than 500 peers, MODEM user fully opens its maximum number of connections and approaches its capacity.

![Figure 17, Average signaling traffic per DSL user](image17)

The difference between Gnutella 0.6 and Gnutella 0.4 are greater for DSL user
because Gnutella 0.4 exhausts the connections and capacity while Leaves, to which most DSL and MODEM users are belong, are common in low traffic without strong correlation to access speed in Gnutella 0.6.

![Figure 18, Average signaling traffic per Bootstrap user](image)

Bootstrap users are equipped with broadband link and keep online during the whole simulation period. They are usually ultrapeers in Gnutella 0.6. The behavior of ultrapeer in Gnutella 0.6 is similar to peers in Gnutella 0.4 by flooding traffic to all known ultrapeer neighbors. The difference is not so great but it still proves that Gnutella 0.6 scales better.
Finally, we put all peers together. We can calculate that $\text{Mean}(\frac{\text{MsgRate}_{0.4}}{\text{MsgRate}_{0.6}}) = 3.5$.

Remembering that we got 5% peers being ultradeers that behave similar in Gnutella 0.4, 35 is greater than $1/5\%=20$ because not only that less peers flood messages but also that messages are flooded in smaller scope.

**4.4 Single traffic**

We also monitored one selected DSL and one selected MODEM user at their outgoing port of the link during all simulations with increasing network size and two protocol versions.
DSL utilization is limited to about 46% to 50% for the maximum number of connections is opened. MODEM utilization is 60%~70%, up to its capacity while the connections are not fully opened. We have to mention that these curves are plotted by single round of simulations. More reliable mean values and intervals based on multiple rounds of simulation are preferred however impossible under the limited time for such a round takes more than 7 days.

Same measurements in Gnutella 0.6 are plotted as follows. Both peers keep as leaves with constant one connection to ultrapeer. The Ping and Pong traffic with a fixed interval of 20 seconds and stochastic traffic of Query/QueryHit traffic compose very lower traffic and is not strongly correlated to network size or access type.
Once received a QueryHit message, the first matched file is downloaded via a TCP connection in the following simulations. The synthesis traffic, signaling plus downloading, is measured and plotted in the following figures.

The downloading traffic should mainly relate to query hit rate that is a complex function of content duplicated ratio, network topology and message TTL, but not only to network size. The curve even slightly declines because while number of peers
increases, but most of peers are free-riders or share few contents, the query hit rate still decreases. The utilizations of MODEM and DSL are about 60% and 46% respectively.

Figure 23, comparison of signaling and synthesis (signaling + downloading) traffic of single Gnutella 0.6 user. (Log-log scaled)

Finally, the pure signaling traffic is compared to synthesis traffic. Signaling traffic is only a minor part of the synthesis traffic. For MODEM, synthesis traffic is 1000 times of signaling traffic. For DSL, the number is still 100. Considering that a nontrivial part of the signaling traffic, Ping messages, is deterministic, the signaling traffic could be ignored when consider a single Gnutella 0.6 traffic source.

4.5 Conclusions

With several simulations, our implementation is validated. The traffic characters of Gnutella 0.6 are investigated by comparing to Gnutella 0.4. Structured P2P network of Gnutella 0.6 shows much better scalability than unstructured Gnutella 0.4 network.

The limits of simulation methodology for P2P network study are also found in
respect of very limited number of peers and short duration possible to be studied.
Further study on aggregate synthetic traffic would be valuable for developing
aggregate source to replace multiple single peers for simulation of much larger
network. On the other hand, an analytical model should be established in
collaboration with simulation methodology.

References

1 RFC-Gnutella 0.6, http://rfc-gnutella.sourceforge.net/developer/testing/index.html
2 Anurag Singla, Christopher Rohrs, Lime Wire LLC, Ultrapeers: Another Step
Towards Gnutella Scalability Version 1.0,
http://rfc-gnutella.sourceforge.net/src/Ultrapeers_1.0.html
4 http://www.isi.edu/nsnam/ns/
5 http://ptolemy.eecs.berkeley.edu/
6 http://www.opnet.com/
9 Andreas Betker, Christoph Gerlach, Ralf Hülsermann*, Monika Jäger, etc.,
Reference Transport Network Scenarios
11 Matei, et.al., Mapping the gnutella network: properties of large scale peer-to-peer
systems and implications
12 J.Chu, K.Lebonte, B. Levine, Availability and Locality Measurements of Peer-Peer
Files, 2002.
13 Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M.


15 Daniel Stutzbach, Reza Rejaie, *Charactering Today’s Gnutella Topology*.

16 Ruediger Schollmeier, Gero Schollmeier, *An analytic model for the behavior of arbitrary peer-to-peer networks*.


18 Beverly Yang, Hector Garcia-Molina, *Designing a Super-Peer Network*

19 Mario Gerla, Jack Tzu-Chieh Tsai, *Multicluster, mobile, multimedia radio network*

20 Ossama Younis, Sonia Fahmy, *Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach*