Exhibit 1

chapter eight

Gnutella

Gene Kan, Gnutella and GoneSilent.com

> When forced to assume [self-government], we were novices in its science. Its principles and forms had entered little into our former education. We established, however, some, although not all its important principles.
>
> —*Thomas Jefferson, 1824*

> Liberty means responsibility. That is why most men dread it.
>
> —*George Bernard Shaw*

Gnutella is among the first of many decentralized technologies that will reshape the Internet and reshape the way we think about network applications. The traditional knee-jerk reaction to create a hierarchical client/server system for any kind of networked application is being rethought. Decentralized technologies harbor many desirable qualities, and Gnutella is a point of proof that such technologies, while young, are viable.

It is possible that Gnutella has walked the Earth before. Certainly many of the concepts it uses—even the unconventional ones—were pioneered long ago. It's tricky to determine what's brand-new and what's not, but this is for certain: Gnutella is the successful combination of many technologies and concepts at the right time.

Gnutella in a gnutshell

Gnutella is a citizen of two different worlds. In the popular consciousness, Gnutella is a peer-to-péer, techno-chic alternative to Napster, the popular Internet music swapping service. To those who look past the Napster association, Gnutella is a landscape-altering technology in and of itself. Gnutella turned every

academically correct notion of computer science on its head and became the first large-scale, fully decentralized system running on the wild and untamed public Internet.

Roughly, Gnutella is an Internet potluck party. The virtual world's equivalents of biscuits and cheese are CPU power, network capacity, and disk space. Add a few MP3s and MPEGs, and the potluck becomes a kegger.

On the technical side, Gnutella brings together a strange mix of CDMA, TCP/IP, and lossy message routing over a reliable connection. It's a really strange concept.

Contrary to popular belief, Gnutella is not branded software. It's not like Microsoft Word. In fact, Gnutella is a language of communication, a protocol. Any software that speaks the language is Gnutella-compatible software. There are dozens of flavors of Gnutella compatibles these days, each catering to different users. Some run on Windows, others on Unix, and others are multiplatform Java or Perl. And as Gnutella's name implies, many of the authors of these Gnutella compatibles have contributed to the open source effort by making the source code of their projects freely available.

A brief history

Besides its impact on the future of intellectual property and network software technology, Gnutella has an interesting story, and it's worth spending a little time understanding how something this big happens with nobody writing any checks.

Gnutella's first breath

Gnutella was born sometime in early March 2000. Justin Frankel and Tom Pepper, working under the dot-com pen name of Gnullsoft, are Gnutella's inventors. Their last life-changing product, Winamp, was the beginning of a company called Nullsoft, which was purchased by America Online (AOL) in 1999. Winamp was developed primarily to play digital music files. According to Tom Pepper, Gnutella was developed primarily to share recipes.

Gnutella was developed in just fourteen days by two guys without college degrees. It was released as an experiment. Unfortunately, executives at AOL were not amenable to improving the state of recipe sharing and squashed the nascent Gnutella just hours after its birth. What was supposed to be a GNU General Public License product when it matured to Version 1.0 was never allowed to grow beyond Version 0.56. Certainly if Gnutella were allowed to develop further under the hands of Frankel and Pepper, this chapter would look a lot different.

At least Gnutella was born with a name. The neologism comes from ramming GNU and Nutella together at high speed. GNU is short for GNU's Not Unix, the geekish rallying cry of a new generation of software developers who enjoy giving free access to the source code of their products. Nutella is the hazelnut and chocolate spread produced by Italian confectioner Ferrero. It is typically used on dessert crepes and the like. I think it's great, and chocolate is my nemesis.

Anyway, Gnutella was declared an "unauthorized freelance project" and put out to pasture like a car that goes a hundred miles on a gallon of gas. Or maybe like a technology that could eliminate the need for a physical music distribution network. Cast out like a technology that could close the books on a lot of old-world business models? Well, something like that, anyway.

Open source to the rescue

It was then, in Gnutella's darkest hour, that open source developers intervened. Open source developers did for Gnutella what the strange masked nomads did for George Clooney and friends in *Three Kings*. Bryan Mayland, with some divine intervention, reverse engineered Gnutella's communication language (also known as "Gnutella protocol") and posted his findings on Gnutella's hideout on the Web: *gnutella.nerdherd.net*. Ian Hall-Beyer and Nathan Moinvaziri created a sort of virtual water cooler for interested developers to gather around. Besides the protocol documentation, probably the most important bit of information on the Nerdherd web site was the link to Gnutella's Internet Relay Chat (IRC) channel, *#gnutella*. *#gnutella* had a major impact on Gnutella development, particularly when rapid response among developers was required.

What makes Gnutella different?

Gnutella has that simple elegance and minimalism that marks all great things. Like Maxwell's equations, Gnutella has no extraneous fluff. The large amount of Gnutella-compatible software available is testimony to that: Gnutella is small, easy, and accessible to even first-time programmers.

Unlike the Internet that we are all familiar with, with all its at signs, dots, and slashes, Gnutella does not give meaningful and persistent identification to its nodes. In fact, the underlying structure of the Internet on which Gnutella lives is almost entirely hidden from the end user. In newer Gnutella software (Gnotella, Furi, and Toadnode, for example), the underlying Internet is completely hidden from view. It simply isn't necessary to type in a complex address to access infor-

mation on the Gnutella system. Just type in a keyword and wait for the list of matching files to trickle in.

Also unlike standard Internet applications such as email, Web, and FTP, which ride on the bare metal of the Internet, Gnutella creates an application-level network in which the infrastructure itself is constantly changing. Sure, the wires stay in the ground and the routers don't move from place to place, but which wires and which routers participate in the Gnutella network changes by the second. The Gnutella network comprises a dynamic *virtual infrastructure* built on a fixed physical infrastructure.

What makes Gnutella different from a scientific perspective is that Gnutella does not rely on any central authority to organize the network or to broker transactions. With Gnutella, you need only connect to one arbitrary host. Any host. In the early days, discovery of an initial host was done by word of mouth. Now it is done automatically by a handful of "host caches." In any case, once you connect with one host, you're in. Your Gnutella node mingles with other Gnutella nodes, and pretty soon you're in the thick of things.

Contrast that to Napster. Napster software is programmed to connect to *www.Napster.com*. At *www.Napster.com* is a farm of large servers that broker your every search and mouse click. This is the traditional client/server model of computing. Don't get me wrong: client/server is great for many things. Among its positive qualities are easy-to-understand scalability and management. The downside is that by being the well-understood mainstay of network application science, client/server is boring, inflexible, and monolithic. Those are bad words in the Internet lexicon.

Gnutella works like the real world

So far, we know that Gnutella is an Internet potluck. We know it's impossible to stop. But how does it actually work all this magic?

In its communication, it's like finding the sushi tray at a cocktail party. The following is a loose description of the interaction on the Gnutella network.

A Gnutella cocktail party

The concepts introduced in this example, primarily the idea that a request is repeated by a host to every other host known by that host, is critical to understanding how Gnutella operates. In any case, you can see that Gnutella's communication concepts closely reflect those of the real world:

Cocktail party	Gnutella
You enter at the foyer and say hello to the closest person.	You connect to a Gnutella host and issue a PING message.
Shortly, your friends see you and come to say hello.	Your PING message is broadcast to the Gnutella hosts in your immediate vicinity. When they receive your PING, they respond with a PONG, essentially saying, "Hello, pleased to meet you."
You would like to find the tray of sushi, so you ask your nearby friends.	You would like to find the recipe for strawberry rhubarb pie, so you ask the Gnutella nodes you've encountered.
None of your drunken friends seem to know where the sushi is, but they ask the people standing nearby. Those people in turn ask the people near them, and so on, until the request makes its way around the room.	One of the Gnutella nodes you're connected to has a recipe for strawberry rhubarb pie and lets you know. Just in case others have a better recipe, your request is passed on to other hosts, which repeat the question to all hosts known to them. Eventually the entire network is canvassed.
A handful of partygoers a few meters away have the tray. They pass back the knowledge of its location by word of mouth.	You get several replies, or "hits," routed back to you.
You walk over to the keepers of the tray and partake of their sushi.	There are dozens of recipes to choose from. You double-click on one and a request is issued to download the recipe from the Gnutella node that has it.

A client/server cocktail party

In contrast, centralized systems don't make much sense in the real world. Napster is a good example of a client/server system, so let's look at how things would be if there were a real-life cocktail party that mimicked Napster's system:

Cocktail party	Napster
You enter at the foyer and the host of the party greets you. Around him are clustered thirty-five million of his closest friends.	You connect to Napster and upload a list of files that you are sharing. The file list is indexed and stored in the memory of the party host: the central server.
Your only friend at this party is the host.	The Napster server says, "File list successfully received."
You would like to find the tray of sushi, so you find your way back to the foyer and ask the host where exactly the tray has gone.	You would like to find the recipe for strawberry rhubarb pie. So you type "rhubarb" into the search box, and the request is delivered to the central server.

Cocktail party	Napster
The host says, "Oh, yes. It's over there."	You get several replies, or "hits," from the Napster server that match your request.
You hold the tray and choose your favorite sushi.	You decide which MP3 file you want to download and double-click. A request is issued to the Napster server for the file. The Napster server determines which file you desire and whose computer it is on, and brokers a download for you. Soon the download begins.

As you can see, the idea of a central authority brokering all interaction is very foreign to us. When I look at what computer science has espoused for decades in terms of real-world interactions, I wonder how we got so far off track. Computer science has defined a feudal system of servers and slaves, but technologies like Gnutella are turning that around at long last.

Client/server means control, and control means responsibility

As it relates to Napster, the server is at once a place to plant a business model and the mail slot for a summons. If Napster threw the switch for Napster subscriptions, they could force everyone to pay to use their service. And if the RIAA (Recording Industry Association of America) wins its lawsuit, Napster just might have to throw the switch the other way, stranding thirty-five million music swappers. We'll see how that suit goes, but whether or not Napster wins in United States Federal Court, it will still face suits in countless municipalities and overseas. It's the Internet equivalent of tobacco: the lawsuits will follow Napster like so many cartoon rain clouds.

Gnutella, on the other hand, is largely free of these burdens. In a decentralized world, it's tough to point fingers. No one entity is responsible for the operation of the Gnutella network. Any number of warrants, writs, and summons can be executed, and Gnutella will still be around to help you find recipes for strawberry rhubarb pie and "Oops, I Did It Again" MP3s.

Thomas Hale, CEO of WiredPlanet, said, "The only way to stop Gnutella is to turn off the Internet." Well, maybe it's not the only way, but it's really hard to think of a way to eliminate every single cell of Gnutella users, which is truly the only way to wipe Gnutella off the planet.

The client is the server is the network

Standard network applications comprise three discrete modules. There is the server, which is where you deposit all the intelligence—the equivalent of the television studio. There is the client, which typically renders the result of some action on the server for viewing by the user—the equivalent of the television. And there is the network, which is the conduit that connects the client and the server—the equivalent of the airwaves.

Gnutella blends all that into one. The client is the server is the network. The client and server are one, of course. That's mainly a function of simplification. There could be two processes, one to serve files and another to download files. But it's just easier to make those two applications one; easier for users and no more difficult for developers.

The interesting thing is that the network itself is embedded in each Gnutella node. Gnutella is an internet built on top of the Internet, entirely in software. The Gnutella network expands as more nodes connect to the network, and, likewise, it does not exist if no users run Gnutella nodes. This is effectively a software-based network infrastructure that comes and goes with its users. Instead of having specialized routers and switches and hubs that enable communication, Gnutella marries all those things into the node itself, ensuring that the communication facilities increase with demand. Gnutella makes the network's users the network's operators.

Distributed intelligence

The underlying notion that sets Gnutella apart from all other systems is that it is a system of distributed intelligence. The queries that are issued on the network are requests for a response, any kind of response.

Suppose you query the Gnutella network for "strawberry rhubarb pie." You expect a few results that let you download a recipe. That's what we expect from today's Gnutella system, but it actually doesn't capture the unique properties Gnutella offers. Remember, Gnutella is a distributed, real-time information retrieval system wherein your query is disseminated across the network in its raw form. That means that every node that receives your query can interpret your query however it wants and respond however it wants, in free form. In fact, Gnutella file-sharing software does just that.

Each flavor of Gnutella software interprets the search queries differently. Some Gnutella software looks inside the files you are sharing. Others look only at the filename. Others look at the names of the parent directories in which the file is

contained. Some Gnutella software interprets multiword queries as conjunctions, while others look at multiword queries as disjunctions. Even the results returned by Gnutella file-sharing software are wildly different. Some return the full path of the shared file. Others return only the name of the file. Yet others return a short description extracted from the file. Advertisers and spammers took advantage of this by returning URLs to web sites completely unrelated to the search. Creative and annoying, yet demonstrative of Gnutella's power to aggregate a collective intelligence from distributed sources.

To prove the point once and for all that Gnutella could be used to all kinds of unimagined benefit, Yaroslav Faybishenko, Spencer Kimball, Tracy Scott, and I developed a prototype search engine powered by Gnutella that we called InfraSearch. The idea was that we could demonstrate Gnutella's broad power by building a search engine that accessed data in a nontraditional way while using nothing but pure Gnutella protocol. At the time, InfraSearch was conceived solely to give meat to what many Gnutella insiders were unable to successfully convey to journalists interested in Gnutella: that Gnutella reached beyond simple file swapping. To illustrate, I'll use the examples we used in our prototype.

InfraSearch was accessed through the World Wide Web using a standard web browser. Its interface was familiar to anyone who had used a traditional web search engine. What happened with the query was all Gnutella. When you typed a search query into InfraSearch, however, the query was not answered by looking in a database of keywords and HTML files. Instead, the query was broadcast on a private Gnutella network comprising a few nodes. The nodes themselves were a hodgepodge of variegated data sources. A short list of the notables: Online Photo Lab's image database, a calculator, a proxy for Yahoo! Finance, and an archive of MoreOver.com's news headlines.

When you typed in "MSFT" the query would be broadcast to all the nodes. Each node would evaluate the query in relation to its knowledge base and respond only if the node had relevant information to share. Typically, that would mean that the Yahoo! Finance node would return a result stating Microsoft's current stock price and the MoreOver.com node would return a list of news stories mentioning Microsoft. The results were just arbitrary snippets of HTML. The HTML fragments would be stitched together by a Gnutella node, which also doubled as a web server, and forwarded on to the web browser. Figure 8-1 shows the results of a search for "rose."

The real power of this paradigm showed itself when one entered an algebraic expression into the search box, say, "1+1*3" for instance. The query would be disseminated and most nodes would realize that they had nothing intelligent to

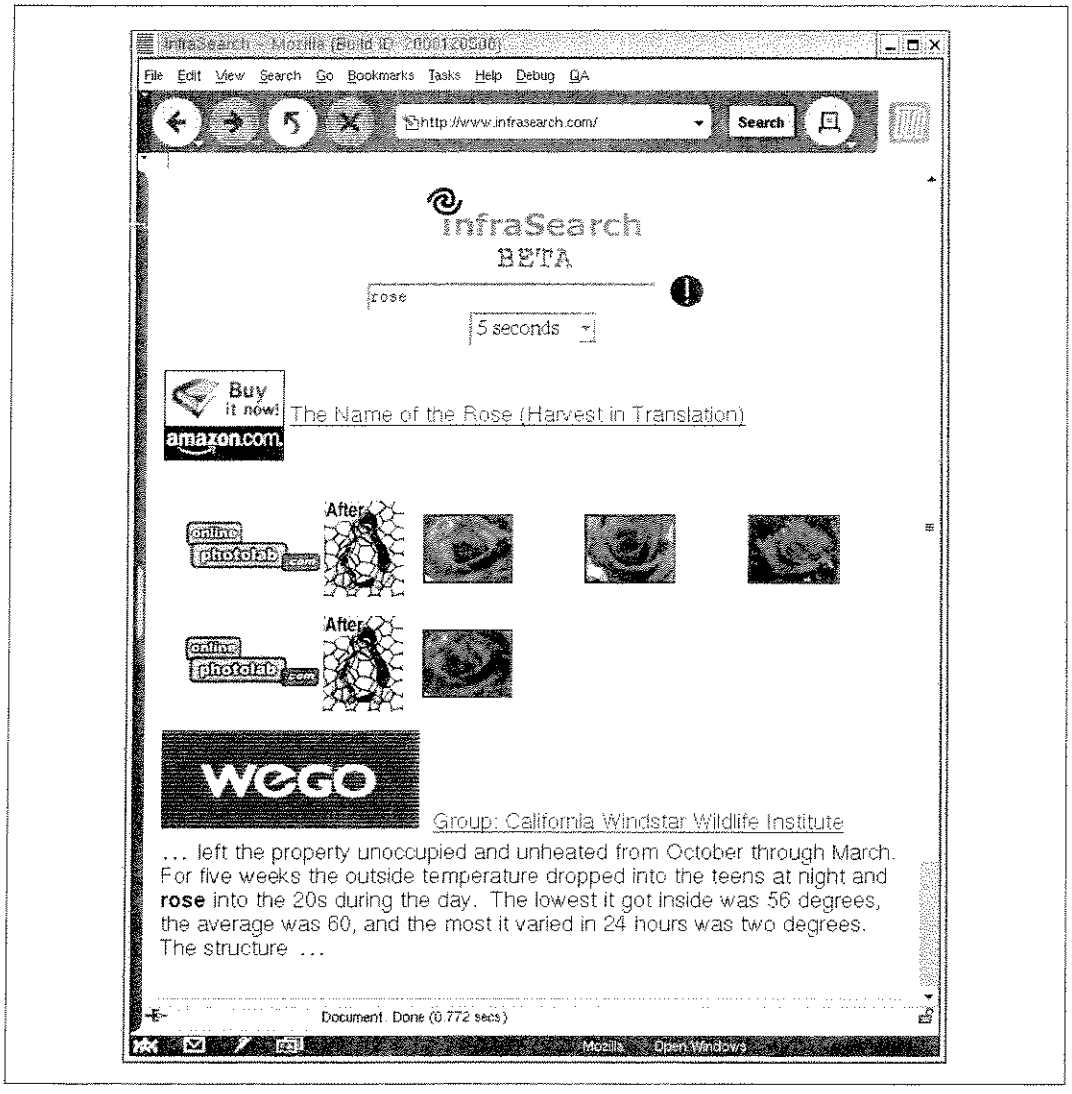

Figure 8-1. Results displayed from Gnutella search

say about such a strange question. All except the calculator node. The calculator was a GNU *bc* calculator hacked to make it speak Gnutella protocol. Every time the calculator received a query, it parsed the text to see if it was a valid algebraic expression. If it was not, then the calculator remained silent. If the query was an algebraic expression, however, the calculator evaluated the expression and returned the result. In this case, "1+1*3 = 4" would be the result.*

* Some creative users would search on ridiculously complex algebraic expressions, causing the calculator node to become overburdened. Gnutella would then simply discard further traffic to the calculator node until it recovered from figuring out what "98791283741984719798797123 4*12341837437488457 65" was. The other nodes continued on unaffected.

One potential application of this is to solve the dynamic page problem on the World Wide Web. Instead of trying to spider those pages as web search crawlers currently do, it would be possible to access the information databases directly and construct a response based upon data available at the time the query was issued. Possibilities that reach even further are within sight. The query could become structured or parameterized, making a huge body of data available through what effectively becomes a unified query interface. The possibilities for something like that in the enterprise are enormous. When peer-to-peer systems take off, accessing data across heterogeneous information stores will become a problem that Gnutella has already demonstrated it can solve.

What we realized is that this aggregation of intelligence maps very closely to the real world. When you ask a question of two different people, you expect two different answers. Asking a question about cars of a mechanic and a toy shop clerk would expectedly yield two very different answers. Yet both are valid, and each reflects a different sort of intelligence in relation to the topic. Traditional search technologies, however, apply only one intelligence to the body of data they search. Distributed search technologies such as Gnutella allow the personality of each information provider and software developer to show through undiluted.

Different from Freenet

Oftentimes Gnutella and Freenet are lumped together as decentralized alternatives to Napster. True, Gnutella and Freenet are decentralized. And it's true that one can share MP3 files using either Gnutella or Freenet. The technical similarities extend further in various ways, but the philosophical division between Gnutella and Freenet picks up right about here.

Freenet can really be described as a bandwidth- and disk space–sharing concept with the goal of promoting free speech. Gnutella is a searching and discovery network that promotes free interpretation and response to queries. With Freenet, one allocates a certain amount of one's hard drive to the task of carrying files which are in the Freenet. One shares bandwidth with others to facilitate the transport of files to their optimal localities in the Freenet. In a sense, Freenet creates a very large and geographically distributed hard drive with anonymous access. The network is optimized for computerized access to those files rather than human interaction. Each file is assigned a complex unique identification that is obscure in its interpretation. The only way to search for files is by searching via that unique identification code.

In contrast, Gnutella is a distributed searching system with obvious applications for humans and less obvious applications for automatons. Each Gnutella node is free to interpret the query as it wants, allowing Gnutella nodes to give hits in the form of filenames, advertising messages, URLs, graphics, and other arbitrary content. There is no such flexibility in the Freenet system. The Japanese Gnutella project, *http://jnutella.org*, is deploying Gnutella on i-Mode mobile phones, where the results of a search are tailored to mobile phone interfaces. Freenet's highly regimented system of file location based upon unique identification is about cooperative distribution of files. There is nothing wrong with this. It's just a different approach with different effects which I'll leave to Freenet's authors to explain.

Gnutella's communication system

With the basic understanding that Gnutella works the way real-world interpersonal communication works, let's take a look at the concepts that make it all possible in the virtual world. Many of these concepts are borrowed from other technologies, but their combination into one system makes for interesting results and traffic jams.

Message-based, application-level routing

Traditional application-level networks are circuit-based, while Gnutella is message-based. There is no idea of a persistent "connection," or circuit, between any two arbitrary hosts on the Gnutella network. They are both on the network but not directly connected to each other, and not even indirectly connected to each other in any predictable or stable fashion. Instead of forcing the determinism provided by circuit-based routing networks, messages are relayed by a computerized bucket-brigade which forms the Gnutella network. Each bucket is a message, and each brigadier is a host. The messages are handed from host to host willy-nilly, giving the network a unique interconnected and redundant topology.

TCP broadcast

Another unconventional approach that Gnutella uses is a broadcast communication model over unicast TCP. Contrast this to a traditional system such as Napster, where communication is carefully regulated to minimize traffic to its absolute lowest levels, and even then to only one or two concerned parties. Traditional networking models are highly regimented and about as natural as formal gardens.

The broadcast mechanism is extremely interesting, because it maps very closely to our everyday lives. Suppose you are standing at a bus stop and you ask a fellow when the next bus is to arrive: "Oi, mate! When's the next bus?" He may not know, but someone nearby who has heard you will hopefully chime in with the desired information. That is the strength behind Gnutella: it works like the real world.

One of the first questions I asked upon learning of Gnutella's TCP-based broadcast was, "Why not UDP?" The simple answer is that UDP is a pain. It doesn't play nicely with most firewall configurations and is tricky to code. Broadcasting on TCP is simple, and developers don't ask questions about how to assess "connection" status. Let's not even start on IP multicast.

Message broadcasting

Combining the two concepts of message-based routing and broadcast gives us what I'll term message broadcasting. Message broadcasting is perfect for situations where more than one network participant can provide a valid response to a request. This same sort of thing happens all the time. Auctions, for example, are an example of message broadcasting. The auctioneer asks for bids, and one person's bid is just as good as another's.

Gnutella's broadcasting mechanism elegantly avoids continuous echoing. Messages are assigned unique identifiers (128-bit unique identifiers, or UUIDs, as specified by Leach and Salz's 1997 *UUIDs and GUIDs Informational Draft* to the IETF). With millions of Gnutella nodes running around, it is probably worth answering the question, "How unique is a UUID?" Leach and Salz assert uniqueness until 3400 A.D. using their algorithm. Anyway, it's close enough that even if there were one or two duplicated UUIDs along the way nobody would notice.

Every time a message is delivered or originated, the UUID of the message is memorized by the host it passes through. If there are loops in the network then it is possible that a host could receive the same message twice. Normally, the host would be obligated to rebroadcast the message just like any other that it received. However, if the same message is received again at a later time (it will have the same UUID), it is not retransmitted. This explicitly prevents wasting network resources by sending a query to hosts that have already seen it.

Another interesting idea Gnutella implements is the idea of decay. Each message has with it a TTL* number, or time-to-live. Typically, a query starts life

* TTL is not unique to Gnutella. It is present in IP, where it is used in a similar manner.

with a TTL of 7. When it passes from host to host, the TTL is decremented. When the TTL reaches 0, the request has lived long enough and is not retransmitted again. The effect of this is to make a Gnutella request fan out from its originating source like ripples on a pond. Eventually the ripples die out.

Dynamic routing

Message broadcasting is useful for the query, but for the response, it makes more sense to route rather than to broadcast. Gnutella's broadcast mechanism allows a query to reach a large number of potential respondents. Along the way, the UUIDs that identify a message are memorized by the hosts it passes through. When Host A responds to a query, it looks in its memory and determines which host sent the query (Host B). It then responds with a reply message containing the same UUID as the request message. Host B receives the reply and looks in its memory to see which host sent the original request (Host C). And on down the line until we reach Host X, which remembers that it actually originated the query. The buck stops there, and Host X does something intelligent with the reply, like display it on the screen for the user to click on (see Figure 8-2).

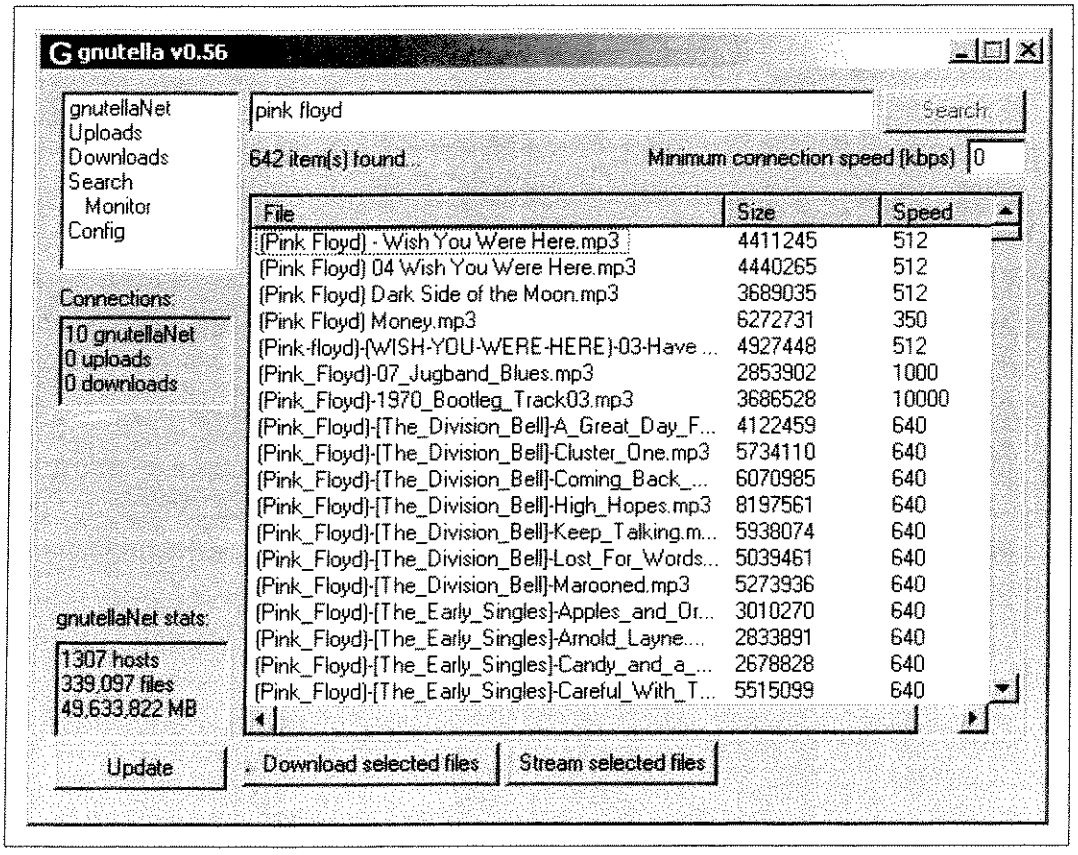

Figure 8-2. Results displayed from a Gnutella query

The idea to create an ephemeral route as the result of a broadcast for discovery is not necessarily novel, but it is interesting. Remember, a message is identified only by its UUID. It is not associated with its originator's IP address or anything of the sort, so without the UUID-based routes, there is no way for a reply to be delivered to the node that made the request.

This sort of dynamic routing is among the things that make Gnutella the intriguing technology that it is. Without it, there would need to be some kind of fixed Gnutella infrastructure. With dynamic routing, the infrastructure comes along with the nodes that join the network, in real time. A node brings with it some network capacity, which is instantly integrated into the routing fabric of the network at large.

When a node leaves the network, it does not leave the network at large in shambles, as is typical for the Internet. The nodes connected to the departing node simply clean up their memories to forget the departed node, and things continue without so much as a hiccup. Over time, the network adapts its shape to long-lived nodes, but even if the longest-lived, highest-capacity node were to disappear, there would be no lasting adverse effects.

Lossy transmission over reliable TCP

A further unconventional notion that is core to Gnutella's communication mechanisms is that the TCP connections that underlie the Gnutella network are not to be viewed as the totally reliable transports they are typically seen as. With Gnutella, when traffic rises beyond the capacity that a particular connection can cope with, the excess traffic is simply forgotten. It is not carefully buffered and preserved for future transmission as is typically done. Traffic isn't coddled on Gnutella. It's treated as the network baggage that it is.

The notion of using a reliable transport to unreliably deliver data is notable. In this case, it helps to preserve the near-real-time nature of the Gnutella network by preventing an overlong traffic backlog. It also creates an interesting problem wherein low-speed Gnutella nodes are at a significant disadvantage when they connect to high-speed Gnutella nodes. When that happens, it's like drinking from a fire hose, and much of the data is lost before it is delivered.

On the positive side, loss rates provide a simple metric for relative capacity. If the loss rate is consistently high, then it's a clear signal to find a different hose to drink from.

Organizing Gnutella

One of the ways Gnutella software copes with constantly changing infrastructure is by creating an ad hoc backbone. There is a large disparity in the speeds of Internet connections. Some users have 56-Kbps modems, and others have, say, T3 lines. The goal is that, over time, the T3-connected nodes migrate toward the center of the network and carry the bulk of the traffic, while the 56-Kbps nodes simultaneously move out toward the fringes of the network, where they will not carry as much of the traffic.

In network terms, the placement of a node on the network (in the middle or on the fringes) isn't determined geographically. It's determined in relation to the topology of the connections the node makes. So a high-speed node would end up being connected to potentially hundreds of other high-speed Gnutella nodes, acting as a huge hub, while a low-speed node would hopefully be connected to only a few other low-capacity nodes.

Over time this would lead the Gnutella network to have a high concentration of high-speed nodes in the middle of the network, surrounded by rings of nodes with progressively decreasing capacities.

Placing nodes on the network

When a Gnutella node connects to the network, it just sort of parachutes in blindly. It lands where it lands. How quickly it is able to become a productive member of Gnutella society is determined by the efficacy of its network analysis algorithms. In the same way that at a cocktail party you want to participate in conversations that interest you, that aren't too dull and aren't too deep, a Gnutella node wants to quickly determine which nodes to disconnect from and which nodes to maintain connections to, so that it isn't overwhelmed and isn't too bored.

It is unclear how much of this logic has been implemented in today's popular Gnutella client software (Gnotella, Furi, Toadnode, and Gnutella 0.56), but this is something that Gnutella developers have slowly educated themselves about over time. Early Gnutella software would obstinately maintain connections to nodes in spite of huge disparities in carrying capacity. The effect was that modem nodes acted as black holes into which packets were sent but from which nothing ever emerged.

One of the key things that we* did to serve the surges of users and new client software was to run high-speed nodes that were very aggressive in disconnecting nodes which were obviously bandwidth disadvantaged. After a short time, the only active connections were to nodes running on acceptably high-speed links. This kind of feedback system created an effective backbone that was captured in numerous early network maps. A portion of one is shown in Figure 8-3.

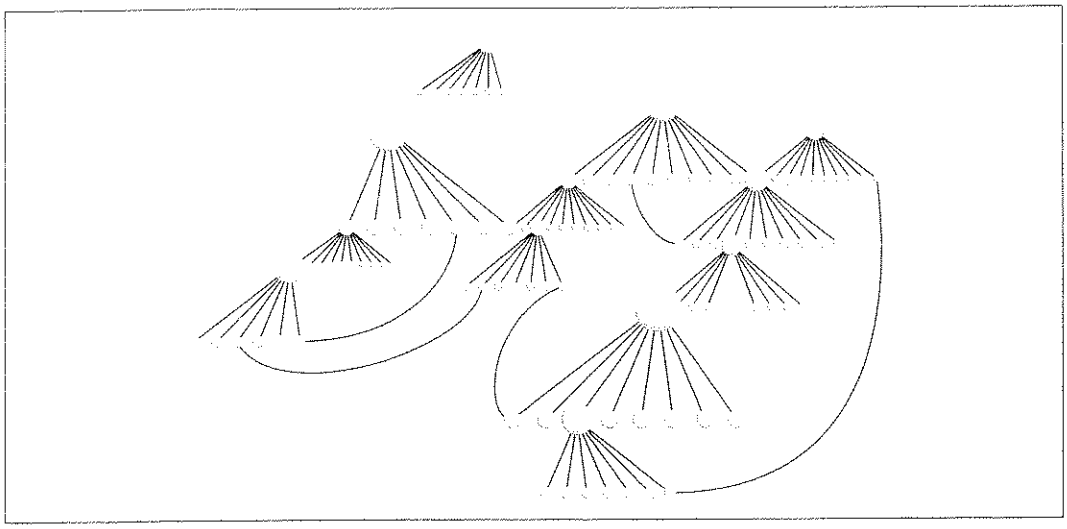

Figure 8-3. Snapshot of effective Gnutella network structure

Gnutella's analogues

The first thing that technologists say when they think about how Gnutella works is, "It can't possibly scale." But that is simply not the case. Gnutella is an unconventional system and as such requires unconventional metrics. Millions of users may be using Gnutella simultaneously, but they will not all be visible to one another. That is the basic nature of a public, purely peer-to-peer network. Because there is no way to guarantee the quality of service throughout the network, it is impossible to guarantee that every node on the network can be reached by every other node on the network. In spite of that, Gnutella has many existing analogues.

Of all the analogues that exist, the most interesting two are cellular telephony and Ethernet.

---

* Bob Schmidt, Ian Hali-Beyer, Nathan Moinvaziri, Tom Camarda, and countless others came to the rescue by running software which made the network work in its times of need. This software ranged from standard Gnutella software to host caches to so-called Mr. Clean nodes, which aggressively removed binary detritus from the network.

The Gnutella horizon

In Gnutella, there is a concept of a horizon. This is simply a restatement of the effect the TTL has on how far a packet can go before it dies, the attenuation of ripples on a pond. Gnutella's standard horizon is seven hops. That means that from where you stand, you can see out seven hops. How far is that? Typically, a seven-hop radius combined with network conditions means about ten thousand nodes are within sight.

When Gnutella was younger, and the pond analogy hadn't yet crossed my mind, I explained this effect as a horizon, because it was just like what happens when you are at the beach and the world seems to disappear after some distance (approximately five kilometers if you're two meters tall). Of course, that is due to the curvature of the earth, but it seemed like a pretty good analogy.

A slightly better one is what happens in a mob. Think first day of school at UC Berkeley, or the annual Love Parade in Germany. You stand there in the middle of the mob, and you can only see for a short distance around you. It's obvious that there are countless more people outside your immediate vision, but you can't tell how many. You don't even really know where you are in relation to the crowd, but you're certainly in the thick of it. That's Gnutella.

Each node can "see" a certain distance in all directions, and beyond that is a great unknown. Each node is situated slightly differently in the network and as a result sees a slightly different network. Over time, as nodes come and go and the network shifts and morphs, your node gets to see many different nodes as the network undulates around it. If you've used Gnutella, you've seen this happen. Initially, the host count increases very rapidly, but after a minute or two, it stabilizes and increases much more slowly than it did at the outset. That is because in the beginning your node discovers the network immediately surrounding it: the network it can see. Once that is done, your node discovers only the nodes that migrate through its field of view.

Cellular telephony and the Gnutella network

In the technological world, this concept is mirrored exactly by cellular telephony cell sites (cellular telephony towers). Each site has a predetermined effective radius. When a caller is outside that radius, his telephone cannot reach the site and must use another if a nearer one is available. And once the caller is outside the operating radius, the site cannot see the caller's telephone either. The effect is the irksome but familiar "no coverage" message on your phone.

Cellular network operators situate cell sites carefully to ensure that cell sites overlap one another to prevent no-coverage zones and dropped calls. A real coverage map looks like a Venn diagram gone mad. This is, in fact, a very close analogue of the Gnutella network. Each node is like a cell site in the sense that it has a limited coverage radius, and each node's coverage area overlaps with that of the nodes adjacent to it. The key to making cellular telephony systems scale is having enough cells and enough infrastructure to connect the cells. It's a similar story with Gnutella.

Cell sites are not all that one needs to build a successful cellular network. Behind all those cell towers is a complex high-bandwidth packet switching system, also much like Gnutella. In the cellular world, this network is very carefully thought out and is a piece of physical infrastructure. As with everything else, the infrastructure comes and goes in the Gnutella network, and things are constantly changing shape.

So then the goal is to find a way to create cells that are joined by a high-speed backbone. This is entirely what would happen in the early Gnutella network. Gnutella nodes would gather around a local hub, forming a cluster. There were numerous clusters interconnected by high-speed lines. All this happened in an unplanned and dynamic way.

Ethernet

Gnutella is also similar in function to Ethernet. In fact, Ethernet is a broadcast network where each message has a unique identifier. Like Gnutella, its scalability metrics are unconventional. The question most people ask about Gnutella is, "How many users are on Gnutella?" The answer is complicated.

Millions of users have Gnutella on their computers. One node can only see about ten thousand others from where it stands in the network. So what is the answer? Ten thousand, or several million?

We could ask the same question about Ethernet, and we'd get the same duality in answer. Hundreds of millions of computers have Ethernet, yet only a few dozen can share an Ethernet "segment" before causing network gridlock. The solution for Ethernet was to develop specialized hardware in the form of Ethernet bridges, switches, and routers. With that hardware, it became possible to squeeze all those millions of computers onto the same network: the Internet.

Cultivating the Gnutella network

Similar development is underway for Gnutella. Fundamentally, each Gnutella node can contain enough logic to make the Gnutella network grow immensely. Broadening the size of a Gnutella cell, or segment, is only a matter of reducing the network traffic. A minor reduction by each node can translate into a huge reduction in traffic over all nodes. That is what happens with distributed systems: a minor change can have a huge effect, once multiplied over the number of nodes.

There is at least one effort underway to create a specialized Gnutella node which outwardly mimics a standard Gnutella node but inwardly operates in a dramatically different manner. It is known as Reflector and is being developed by a company called Clip2. The Reflector is effectively a miniature Napster server. It maintains an index of the files stored on nodes to which it is connected. When a query is issued, the Reflector does not retransmit it. Rather, it answers the query from its own memory. That causes a huge reduction in network use.*

Anyone can run a Reflector, making it an ideal way to increase the size of a Gnutella cluster. Connecting Reflectors together to create a super high-capacity backbone is the obvious next step. Gnutella is essentially an application-level Internet, and with the development of the Gnutella equivalent of Cisco 12000s, Gnutella will really become what it has been likened to so many times: an internet on the Internet.

Gnutella's traffic problems

One place where the analogy drawn between Gnutella and cellular telephony and Ethernet holds true down to its last bits is how Gnutella suffers in cases of high traffic. We know this because the public Gnutella network at the time of this writing has a traffic problem that is systemic, rather than the standard transient attack. Cellular telephones show a weakness when the cell is too busy with active calls. Sometimes there is crosstalk; at other times calls are scratchy and low quality. Ethernet similarly reaches a point of saturation when there is too much traffic on the network, and, instead of coping gracefully, performance just degrades in a downward spiral. Gnutella is similar in almost every way.

---

* Depending on your view, the benefit, or unfortunate downside, of Reflector is that it makes Gnutella usable only in ways that Reflector explicitly enables. To date, Reflector is chiefly optimizing the network for file sharing, and because it removes the ability for hosts to respond free-form and in real time, it sacrifices one of the key ideas behind Gnutella.

In terms of solutions, the bottom line is that when too many conversations take place in one cell or segment the only way to stop the madness is to break up the cell.

On the Gnutella network, things started out pretty peacefully. First a few hundred users, then a few thousand, then a few hundred thousand. No big deal. The network just soldiered along. The real problem came along when host caches came into wide use.

Host caches

In the early days of Gnutella, the way you found your way onto the network was by word of mouth. You got onto IRC and asked for a host address to connect to. Or you checked one of the handful of web pages which maintained lists of hosts to connect to. You entered the hosts into your Gnutella software one by one until one worked. The the software took care of the rest. It was tedious, but it worked for a long while.

Before host caches, it was fairly random what part of the network you connected to. Ask two different people, and they would direct you to connect to hosts on opposite sides of the Gnutella network. Look at two different host lists, and it was difficult to find any hosts in common. Host lists encouraged sparseness and small clusters. It was difficult for too many new hosts to be concentrated into one cell. The cells were sparsely connected with one another, and there wasn't too much crosstalk. That created a nearly optimal network structure, where the Gnutella network looked like a land dotted by small cities and townships interconnected by only a few roads.

Users eventually became frustrated by the difficulties of getting onto Gnutella. Enter Bob Schmidt and Josh Pieper. Bob Schmidt is the author of GnuCache, a host caching program. Josh Pieper also included host caching logic in his popular Gnut software for Unix. Host caches provide a jumping off spot for Gnutella users, a host that's always up and running, that gives a place for your Gnutella software to connect to and find the rest of the Gnutella network.* The host cache greets your node by handing off a list of other hosts your node should connect to. This removes the uncertainty from connecting to Gnutella and provides a more friendly user experience. We were all very thankful for Schmidt and Pieper's efforts until host caches became a smashing success.

* Actually, Gnutella was born with a ready host cache located at *findshit.gnutella.org*. Unfortunately, the same people who took away Gnutella also took away *findshit.gnutella.org*, leaving us with a host-cacheless world until GnuCache and Pieper's Gnut software came along.

An unexpected consequence evidenced itself when waves of new Gnutella users logged on in the wake of the Napster injunction on July 26, 2000. Everyone started relying on host caches as their only means of getting onto the Gnutella network. Host caches were only telling new hosts about hosts they saw recently. By doing that, host caches caused Gnutella nodes to be closely clustered into the same little patch of turf on the Gnutella network. There was effectively only one tightly clustered and highly interconnected cell, because the host caches were doling out the same list of hosts to every new host that connected. What resulted was overcrowding of the Gnutella airwaves and a downward spiral of traffic.

Oh well. That's life in the rough-and-tumble world of technology innovation.

To draw an analogy, the Gnutella network became like a crowded room with lots of conversations. Sure, you can still have a conversation, but maybe only with one or two of your closest friends. And that is what has become frustrating for Gnutella users. Whereas the network used to have a huge breadth and countless well-performing cells of approximately ten thousand nodes each, the current network has one big cell in which there is so much noise that queries only make it one or two hops before drowning in overcrowded network connections.

Effectively, a crowded network means that cells are only a few dozen hosts in size. That makes the network a bear to use and gives a disappointing user experience.

Returning the network to its natural state

Host caches were essentially an unnatural addition to the Gnutella network, and the *law of unintended consequences* showed that it could apply to high technology, too. Improving the situation requires a restoration of the network to its original state, where it grew organically and, at first glance, inefficiently. Sometimes, minor inefficiency is good, and this is one of those cases.

Host lists, by enforcing a sparse network, made it so that the communities of Gnutella nodes that did exist were not overcrowded. Host caches created a tightly clustered network, which, while appearing more efficient, in fact led to a major degradation in overall performance. For host caches to improve the situation, they need only to encourage the sparseness that we know works well.

Sort of. An added complication is that each Gnutella host maintains a local host *catcher*, in which a long list of known hosts (all hosts encountered in the node's travels) is deposited for future reference. The first time one logs into the Gnutella network, a host list or a host cache must be used. For all future logins,

Gnutella softwares refer to their host catcher to connect into the Gnutella network. This creates a permanent instability in the network as nodes log on and connect to hosts they remember, irrespective of the fact that those hosts are often poor choices in terms of capacity and topology. The problem is compounded by the reluctance of most Gnutella software to "forget" hosts that are unsuitable.

Turning the network around is technically easy. Host caches can listen to the levels of traffic on each cell they want to serve and distribute new hosts among those cells until the traffic levels become high enough to warrant establishment of new cells. By purposely separating nodes into distinct cells, traffic in each cell can be reduced to a manageable level. With those well-planted seeds and periodic resets of the collective memories of host catchers to allow smart host caches to work their magic, the network can be optimized for performance. The trouble with host catchers, though, is that they are seldom reset, because that requires manual intervention as well as some understanding of the reset mechanism.

Private Gnutella networks

One feature that some Gnutella client software implements is the notion of private Gnutella networks. To join a private network, one needs to know the secret handshake or password. That enforces manageably-sized networks with a predetermined community membership, and it is a pretty good way to ensure a high quality of service no matter what is happening out in the wilds of the public Gnutella network.

Reducing broadcasts makes a significant impact

Broadcasts are simultaneously the most powerful and the most dangerous feature of the Gnutella protocol. Optimally, there are two broadcast packet types: PING and QUERY. PING packets are issued when a node greets the network and wants to learn what other nodes are available to connect to. QUERY packets are issued when a search is conducted. Some Gnutella developers also implemented the PUSH REQUEST packet as a broadcast packet type. PUSH REQUEST packets are used to request files from hosts which are protected by firewalls. The concept is one of the unsung heroes of Gnutella, making it work in all but the most adverse of Internet environments (the double firewall, arch enemy of productivity).

Unfortunately, the PUSH REQUEST packet should be implemented as a routed packet rather than a broadcast packet. At times, PUSH REQUEST packets com-

prised 50% of all Gnutella network traffic. Simply routing those packets rather than broadcasting them would reduce the overall network traffic dramatically.

Reductions can also be made in the number of queries that are broadcast to large expanses of the network by intelligently caching results from similar searches. Clip2's Reflector software is an example of such a product. Portions of Reflector can be integrated into each Gnutella client, leading to a small increase in the software's internal complexity (the user need not concern herself with this behind-the-scenes activity) in exchange for a massive improvement in network performance.

The final broadcast packet that was the carrier of some early abuses is the PING packet. In early Gnutella software, PING packets could have a payload, even though it was not clear what that payload might contain. It was subsequently abused by script kiddies to debilitate the Gnutella network. Gnutella developers responded immediately by altering their software to discard PING packets with payloads, causing a several thousand–fold traffic reduction on the Gnutella network and simultaneously foiling what amounted to a denial of service attack.

What developers have been debating ever since is how to reduce the level of traffic usurped by PING packets. Suggestions have ranged from eliminating PING packets to reducing the allowed number of retransmissions of a PING packet. Personally, I favor something in the middle, where every host on the network behaves as a miniature host cache for its locality, returning proxied greetings for a few nearby hosts in response to a PING and only occasionally retransmitting the PING. That would allow the PING to continue to serve its valuable duty in shaping the network's structure and connectivity, while reducing the network's traffic levels dramatically. Figure a thousand-fold reduction in traffic.

One thing to consider in distributed applications is that, no matter how difficult the code is to write and how much it bloats the code (within reason), it's worth the trouble, because the savings in network utilization pay dividends on every packet. As an example, just consider that if a PUSH REQUEST packet is broadcast, it may reach 1,000 hosts. If it is routed, it may reach four or five. That is a 200-fold reduction in traffic in exchange for a dozen lines of code.

The analogy that Gnutella is like an Internet potluck rings true. Everyone brings a dish when they join the Gnutella network. At a minimum, the one dish that everyone brings is network capacity. So then there is definitely enough bandwidth to go around. The only matter is to organize the combined capacity and manage the traffic to make sure the network operates within its limits.

We just covered what is probably Gnutella's biggest problem, so if this was a corporate memorandum, this would be the perfect point at which to introduce the engineering organization. This isn't a memo, but let's do it anyway. Keep in mind the Thomas Jefferson quotation at the beginning of the chapter: none of us knew what we were doing, but we got our hands dirty and took responsibility for what we did.

The policy debates

Napster and Gnutella have really been at the center of the policy debate surrounding the new breed of peer-to-peer technologies. For the moment, let's forget about the debate that's burning in the technology community about what is truly peer-to-peer. We'll get back to that later and tie all these policy questions back to the technology.

Napster wars

There is only one thing that gets people more riled up than religion, and that is money. In this case, the squabble is over money that may or may not be lost to online music swaps facilitated by services such as Napster and systems like the Gnutella network. This war is being fought by Napster and the RIAA, and what results could change the lives of everyone, at least in the United States.

Well, sort of. The idea that lawsuit or legislature can stop a service that everyone enjoys is certainly a false one. Prohibition was the last real effort (in the U.S.) by the few against the many, and it was a dismal failure that gave rise to real criminal activity and the law's eventual embarrassing repeal. We have an opportunity to see all that happen again, or the recording industry could look at what's coming down the road and figure out a way to cooperate with Napster before the industry gets run down by next-generation peer-to-peer technologies.

Napster, at least, provides a single place where file swappers can be taxed. With Gnutella and Freenet, there is no place to tax, no person to talk to about instituting a tax, and no kinds of controls. The recording industry may hope that Gnutella and Freenet will "just go away," but that hope will probably not materialize into reality.

Anonymity and peer-to-peer

One of the big ideas behind peer-to-peer systems is their potential to provide a cloak under which users can conduct information exchanges without revealing their identities or even the information they are exchanging. The possibility of

anonymity in nearly every case stems from the distribution of information across the entire network, as well as the difficulty in tracking activities on the network as a whole.

Gnutella provides some degree of anonymity by enabling an essentially anonymous searching mechanism. It stops there, though. Gnutella reveals the IP address of a downloading host to the uploading host, and vice versa.

Gnutella pseudoanonymity

Gnutella is a prime example of peer-to-peer technology. It was, after all, the first successful, fully decentralized, peer-to-peer system. But in the policy debate, that's not a huge matter. What does matter is that Gnutella's message-based routing system affords its users a degree of anonymity by making request and response packets part of a crowd of similar packets issued by other participants in the network.

In most messages that are passed from node to node, there is no mention of anything that might tie a particular message to a particular user. On the Internet, identity is established using two points of data: An IP address and the time at which the packet containing the IP address was seen. Most Gnutella messages do not contain an IP address, so most messages are not useful in identifying Gnutella users. Also, Gnutella's routing system is not outwardly accessible. The routing tables are dynamic and stored in the memory of the countless Gnutella nodes for only a short time. It is therefore nearly impossible to learn which host originated a packet and which host is destined to receive it.

Furthermore, Gnutella's distributed nature means that there is no one place where an enforcement agency can plant a network monitor to spy on the system's communications. Gnutella is spread throughout the Internet, and the only way to monitor what is happening on the Gnutella network is to monitor what is happening on the entire Internet. Many are suspicious that such monitoring is possible, or even being done already. But given the vastness of today's Internet and its growing traffic, it's pretty unlikely.

What Gnutella does subject itself to, however, are things such as Zeropaid.com's Wall of Shame. The Wall of Shame, a Gnutella Trojan Horse, was an early attempt to nab alleged child pornography traffickers on the Gnutella network. This is how it worked: a few files with very suggestive filenames were shared by a special host. When someone attempted to download any of the files, the host would log the IP address of the downloader to a web page on the Wall of Shame. The host obtained the IP address of the downloader from its connection information.

That's where Gnutella's pseudoanonymity system breaks down. When you attempt to download, or when a host returns a result, identifying information is given out. Any host can be a decoy, logging that information. There are systems that are more interested in the anonymity aspects of peer-to-peer networking, and take steps such as proxied downloads to better protect the identities of the two endpoints. Those systems should be used if anonymity is a real concern.

The Wall of Shame met a rapid demise in a rather curious and very Internet way. Once news of its existence circulated on IRC, Gnutella users with disruptive senses of humor flooded the network with suggestive searches in their attempts to get their IP addresses on the Wall of Shame.

Downloads, now in the privacy of your own direct connection

So Gnutella's message-based routing system and its decentralization both give some anonymity to its users and make it difficult to track what exactly is happening. But what really confounds any attempt to learn who is actually sharing files is that downloads are a private transaction between only two hosts: the uploader and the downloader.

Instead of brokering a download through a central authority, Gnutella has sufficient information to reach out to the host that is sharing the desired file and grab it directly. With Napster, it's possible not only to learn what files are available on the host machines but what transactions are actually completed. All that can be done easily, within the warm confines of Napster's machine room.

With Gnutella, every router and cable on the Internet would need to be tapped to learn about transactions between Gnutella hosts or peers. When you double-click on a file, your Gnutella software establishes an HTTP connection directly to the host that holds the desired file. There is no brokering, even through the Gnutella network. In fact, the download itself has nothing to do with Gnutella: it's HTTP.

By being truly peer-to-peer, Gnutella gives no place to put the microscope. Gnutella doesn't have a mailing address, and, in fact, there isn't even anyone to whom to address the summons. But because of the breakdown in anonymity when a download is transacted, Gnutella could not be used as a system for publishing information anonymously. Not in its current form, anyway. So the argument that Gnutella provides anonymity from search through response through download is impossible to make.

Anonymous Gnutella chat

But then, Gnutella is not exclusively a file-sharing system. When there were fewer users on Gnutella, it was possible to use Gnutella's search monitor to chat with other Gnutella users. Since everyone could see the text of every search that was being issued on the network, users would type in searches that weren't searches at all: they were messages to other Gnutella users (see Figure 8-4).

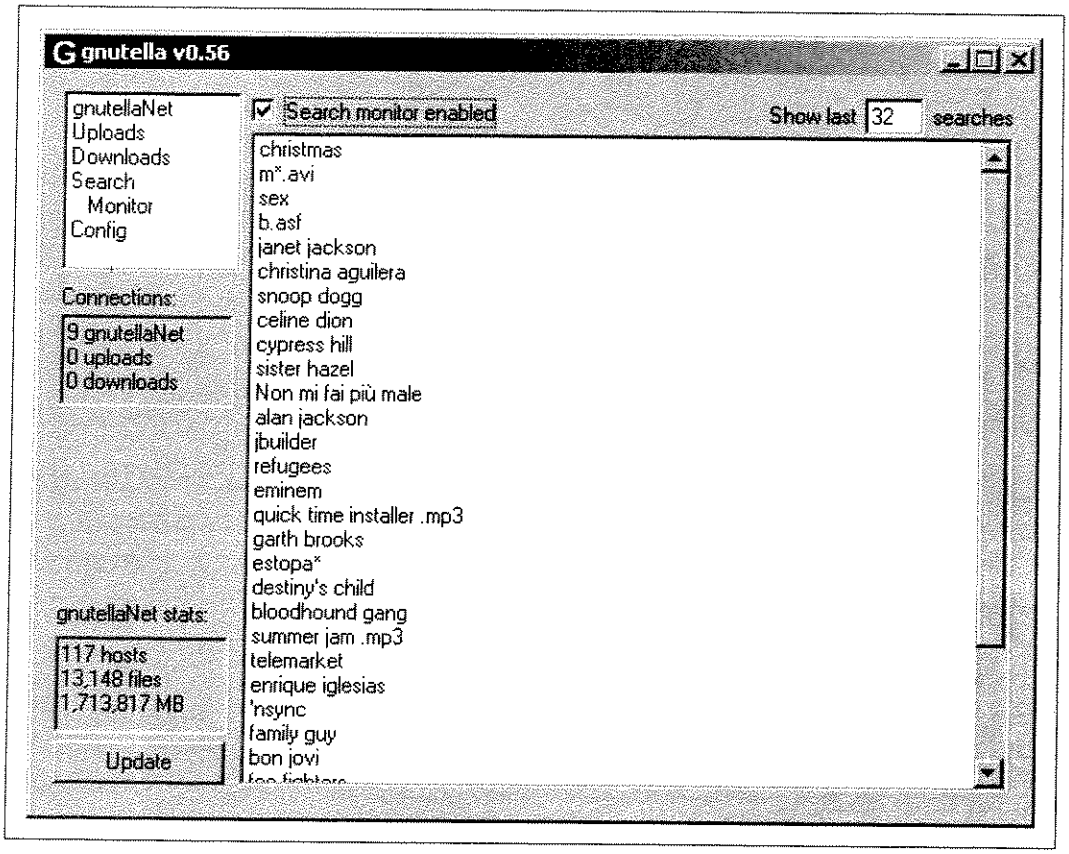

Figure 8-4. Gnutella search monitor

It was impossible to tell who was saying what, but conversations were taking place. If you weren't a part of the particular thread of discussion, the messages going by were meaningless to you. This is an excellent real-world example of the ideas behind Rivest's "Chaffing and Winnowing."* Just another message in a sea of messages. Keeping in mind that Gnutella gives total anonymity in searching, this search-based chat was in effect a totally anonymous chat! And we all thought we were just using Gnutella for small talk.

---

* Ronald L Rivest (1998), "Chaffing and Winnowing: Confidentiality without Encryption," *http://www.toc.lcs.mit.edu/~rivest/chaffing.txt*.

Next-generation peer-to-peer file-sharing technologies

No discussion about Gnutella, Napster, and Freenet is complete without at least a brief mention of the arms race and war of words between technologists and holders of intellectual property. What the recording industry is doing is sensitizing software developers and technologists to the legal ramifications of their inventions. Napster looked like a pretty good idea a year ago, but today Gnutella and Freenet look like much better ideas, technologically and politically. For anyone who isn't motivated by a business model, true peer-to-peer file-sharing technologies are the way to go.

It's easy to see where to put the toll booths in the Napster service, but taxing Gnutella is trickier. Not impossible, just trickier. Whatever tax system is successfully imposed on Gnutella, if any, will be voluntary and organic—in harmony with Gnutella, basically. The same will be true for next-generation peer-to-peer file-sharing systems, because they will surely be decentralized.

Predicting the future is impossible, but there are a few things that are set in concrete. If there is a successor to Gnutella, it will certainly learn from the lessons taught to Napster. It will learn from the problems that Gnutella has overcome and those that frustrate it today. For example, instead of the pseudoanonymity that Gnutella provides, next generation technologies may provide true anonymity through proxying and encryption. In the end, we can say with certainty that technology will outrun policy. It always has. The question is what impact that will have.

Gnutella's effects

Gnutella started the decentralized peer-to-peer revolution.* Before it, systems were centralized and boring. Innovation in software came mainly in the form of a novel business plan. But now, people are seriously thinking about how to turn the Internet upside down and see what benefits fall out.

Already, the effects of the peer-to-peer revolution are being felt. Peer-to-peer has captured the imagination of technologists, corporate strategists, and venture

* The earliest example of a peer-to-peer application that I can come up with is Zephyr chat, which resulted from MIT's Athena project in the early 1990s. Zephyr was succeeded by systems such as ICQ, which provided a commercialized, graphical, Windows-based instant messaging system along the lines of Zephyr. Next was Napster. And that is the last notable client/server-based, peer-to-peer system. Gnutella and Freenet were next, and they led the way in decentralized peer-to-peer systems.

capitalists alike. Peer-to-peer is even getting its own book. This isn't just a passing fad.

Certain aspects of peer-to-peer are mundane. Certain other aspects of it are so interesting as to get notables including George Colony, Andy Grove, and Marc Andreessen excited. That doesn't happen often. The power of peer-to-peer and its real innovation lies not just in its file-sharing applications and how well those applications can fly in the face of copyright holders while flying under the radar of legal responsibility. Its power also comes from its ability to do what makes plain sense and what has been overlooked for so long.

The basic premise underlying all peer-to-peer technologies is that individuals have something valuable to share. The gems may be computing power, network capacity, or information tucked away in files, databases, or other information repositories, but they are gems all the same. Successful peer-to-peer applications unlock those gems and share them with others in a way that makes sense in relation to the particular applications.

Tomorrow's Internet will look quite different than it does today. The World Wide Web is but a little blip on the timeline of technology development. It's only been a reality for the last six years! Think of the Web as the Internet equivalent of the telegraph: it's very useful and has taught us a lot, but it's pretty crude. Peer-to-peer technologies and the experience gained from Gnutella, Freenet, Napster, and instant messaging will reshape the Internet dramatically.

Unlike what many are saying today, I will posit the following: today's peer-to-peer applications are quite crude, but tomorrow's applications will not be strictly peer-to-peer or strictly client/server, or strictly anything for that matter. Today's peer-to-peer applications are necessarily overtly peer-to-peer (often to the users' chagrin) because they must provide application and infrastructure simultaneously due to the lack of preexisting peer-to-peer infrastructure. Such infrastructure will be put into place sooner than we think. Tomorrow's applications will take this infrastructure for granted and leverage it to provide more powerful software and a better user experience in much the same way modern Internet infrastructure has.

In the short term, decentralized peer-to-peer may spell the end of censorship and copyright. Looking out, peer-to-peer will enable crucial applications that are so useful and pervasive that we will take them for granted.