Peer-to-Peer Content Delivery Using Information Additive Codecs

This whitepaper is derived from an engineering notebook started September 15, 2001 that documented my thought processes in anticipation of filing for patent protection. As of this date, I'm no longer pursuing the patent. I've decided to publish this material in hope that those who are in a position to implement and commercialize it may benefit from my efforts.

—Doug Kaye, 29 January 2002 (doug@rds.com)

Revision History

29 January 2002	Initial Publication

Contents

1 Introduction

In June, 2001, I completed writing *Strategies for Web Hosting and Managed Services* and began work on a second book on the topic of "Content Delivery." During my research for the second book, I discovered an audio recording of a presentation entitled *Meta-Content--A New Way of Dealing With Content* given by Michael Luby of Digital Fountain at *CDN Spring 2001*. (This recording is apparently no longer available on line.)

From Luby's presentation I learned that his technology requires a multicast network, or at least an *application-level multicast overlay* network. So while the technology offers tremendous advantages, it cannot be fully exploited on the public Internet or on private networks that are not multicast-enabled.

It occurred to me that combining a peer-to-peer content delivery network (P2PCDN) and an encoding technology similar to Digital Fountain's meta-content would be a better way to delivery content than either of the two technologies on their own. I also realized that other P2P applications could benefit from the use of a technology similar to meta-content. For instance, Napster (the music-sharing system) allows the user to download an individual file from only one source at a time, and if packets are dropped during the transfer or the source disappears (e.g., the remote user disconnects from the Internet), the file transfer slows or stops altogether. A technology such as meta-content would allow one to download simultaneously from multiple remote sources.

I visited the Digital Fountain web site (www.digitalfountain.com) and downloaded and read the whitepaper entitled *A Quantum Leap in Content Delivery: Digital Fountain's Meta-Content™ Technology*. Digital Fountain's meta-content technology is but one of a family of data encoding techniques that could be used in conjunction with a P2P network. The concepts contained herein don't depend on only a single algorithm or technology for either. For instance, a *Reed-Solomon* or *Tornado* encoding scheme (or perhaps others) could be used in lieu of the Luby-Transform. The term *Information Additive Codec* (IAC) refers to the set of suitable encoding techniques.

In my on-going research to improve the concept, I also encountered OpenCola (http://www.opencola.org:8080/). Their Swarmcast protocol is similar to what I've developed. Unlike the Digital Fountain's concept, Swarmcast is truly peer-to-peer, and it uses the same mathematical content abstractions (IACs, or more generically called FECs, forward error-correction codes) that I described. Swarmcast is optimized for large (>1 megabyte) objects.

1.1 Objectives

The proposed system overcomes many of the cost, scalability, performance and reliability problems of transmitting digital content over a data network. Some of these problems have previously been addressed by the technologies of (a) content abstraction, (b) content delivery networks (CDNs), and (c) peer-to-peer (P2P) networks, but all of these approaches have failed to provide a complete solution. The system combines all three

technologies in a unique and novel manner to create a solution that is superior to that offered by any of the previous technologies on its own or in earlier combinations.

1.2 Scope and Application

This system solves the problem of how to efficiently, reliably and cost-effectively deliver digital content over a data network from one or more servers to multiple clients (client/server) or between multiple clients without servers (peer-to-peer).

Applications for the system include the following:

- Transmission of live streaming data such as news and events (live audio and video streams)
- Transmission of on-demand streaming data
- File downloads and data distribution
- Content sharing (pictures, music, videos, etc.)
- On-line collaboration
- Delivery of web pages and other static content
- Delivery of partially-dynamic data (e.g., stock ticker feeds)

2 Existing Technologies

The standard technology for reliable delivery of digital content from a server to one or more clients via a data network is based on the creation and utilization of a *connection* between the server and each client. This is typically accomplished through the use of a connection-oriented protocol such as the Transmission Control Protocol (TCP) to manage the transmission of data between the server and each client, as illustrated in Figure 1

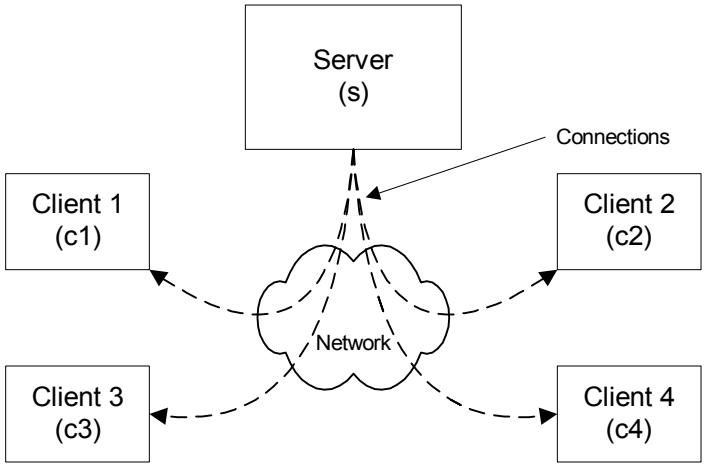

Figure 1

There are two problems with this technique:

- The costs of both server capacity and data transmission increase in direct proportion to the number of clients. This is unlike broadcast or cable television, for instance, in which the incremental cost to transmit to an additional client (receiver) is minimal.
- Because all data must traverse the entire length of the connection from server to client including passing through potentially many different networks and routers owned and operated by multiple parties, the performance and quality of the network may be inadequate for technical or aesthetic reasons.

Note that some systems use a connectionless protocol such as the User Datagram Protocol (UDP) instead of TCP, but they still stuffer from the above problems and furthermore don't provide a reliable transport mechanism.

Many solutions to these problems have been developed and deployed, and three are particularly relevant to the new system. The first is content delivery networks (CDNs); the second is peer-to-peer (P2P) networks; and the third is Information Additive Codec (IACs).

2.1 Content Delivery Networks

CDNs solve the problems of cost, performance and quality by introducing edge servers that are topologically closer to the clients than is the primary or origin server. This is illustrated in Figure 2.

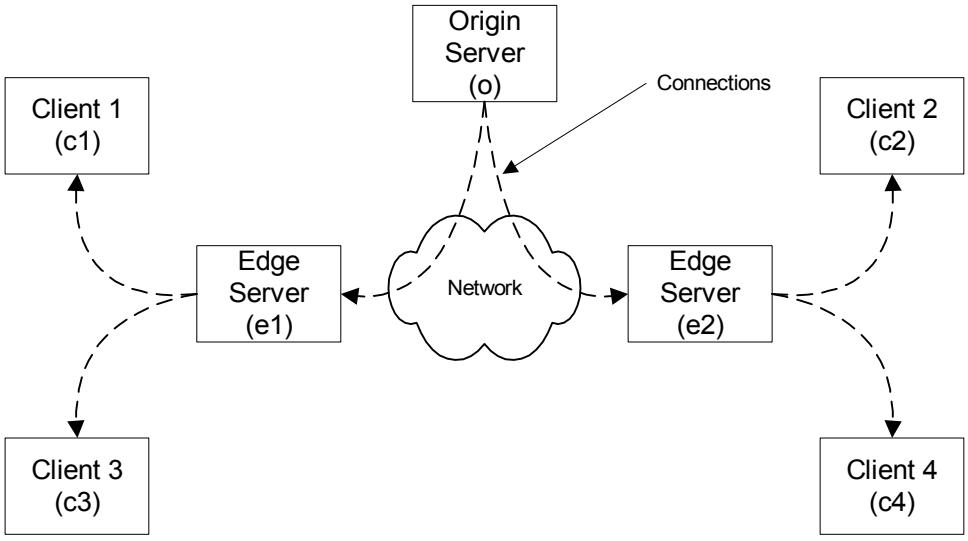

Figure 2

Because clients communicate with the edge servers rather than the origin server, the communication paths are shorter. This causes those paths to be generally less expensive, faster and more reliable. Furthermore, the load on the origin server is reduced, as it handles fewer requests directly.

While CDNs have proven to be effective at reducing costs and increasing performance and reliability, they still don't offer the same scalability as broadcast or cable television. This is due to the fact that the deployment of edge servers is still a substantial expense, and each additional edge server—on behalf of the clients it supports—increases the load on the origin server.

Furthermore, the communications paths between clients and edge servers are still not as short, reliable and efficient as possible.

Although conventional CDNs address some of the problems of traditional content delivery, they are only a partial solution.

2.2 Peer-to-Peer Networks

Peer-to-Peer (P2P) networks were first created to distribute computational tasks to a large number of small computers (e.g., http://www.seti.org/), followed by systems for the person-to-person exchange of files such as digitized music (e.g., http://www.napster.com). More recently P2P architectures have been combined with content delivery networks (e.g., http://www.centerspan.com).

Figure 3 illustrates the operation of a typical P2P network. In this example, client 2 (c2) requires a copy of an object as indicated by the empty square within the c2 rectangle. c2 asks the server (s), "Who has the object?" The server replies, "You can find the object at c1 and c3." Either with user intervention (as in the case of Napster) or based on some algorithm (as in the case of a P2P CDN) c2 selects either c1 or c3, and downloads the object from that source. The dotted line in Figure 3 illustrates that c2 has selected c1 as the source of the download.

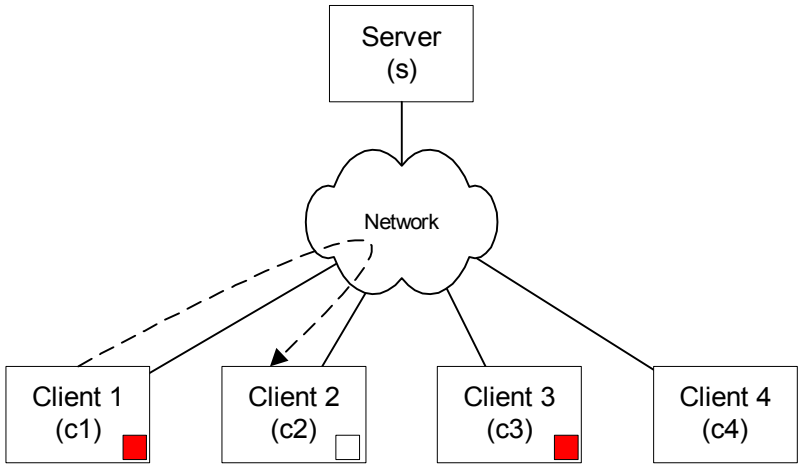

Figure 3

There are many variations on the above scheme including some that are entirely server-less. In such cases the clients (peers of one another) communicate among themselves to locate objects.

All current P2P content networks have one problem, however. If a client (perhaps with the assistance of a server) selects a single peer (another client) from which to request an object, there is no guarantee that the selected client will remain online throughout the download. Likewise, it is difficult to predict the quality of the network path between the peers. Since clients are often not as robust as servers—both in terms of configuration as well as infrastructure and connectivity—they are notoriously unreliable.

To compensate for this, some P2P networks allow a client to download an object from more than one peer simultaneously, or if the download from one peer fails, to then request the object from a subsequent peer. While this increases reliability, it does so at the cost of reduced performance.

Simultaneous downloads are typically not additive. That is, the identical data is redundantly downloaded from multiple sources. Since the multiple downloads must share a communications channel and other client resources, the effective download rate is reduced.

Similarly, if the client must fallback to a subsequent peer as the source for an object, significant time will be wasted while a partially downloaded object is retrieved in its entirety from the subsequent peer. In any case, it takes time to determine that the first peer is incapable of completing the download.

Some techniques have been developed to reduce the impact of such problems. For example, some P2P networks allow two or more peers to deliver separate portions of a requested object. For instance, c2 could request the first third of the object from c1 and the second third of the object from c3. c2 would then request the final third of the object from the peer that was able to deliver its portion most quickly. Such solutions offer improvements, but they are still vulnerable to sudden changes in peer availability or connectivity.

2.3 Information Additive Codecs

Normally, a digitally-represented object such as a file containing text, audio, or a still or moving image is be transmitted electronically by breaking the object into smaller packets—perhaps using compression—and transmitting those packets in sequence from a sender to a receiver.

The protocol for such transmissions may be either reliable (such as the Transmission Control Protocol, TCP) or unreliable (such as the User Datagram Protocol, UDP). Assuming the need for accurate reproduction of the original object at the receiving end, however, a reliable protocol (e.g., TCP) is typically used, in which case the sequence of packets is deterministic and all lost packets must be retransmitted. Only upon receipt of exact copies of all of the original packets can an object transmitted in this manner be accurately reproduced.

An Information Additive Codec (IAC), on the other hand, treats digital content much like a holographic image, and encodes the content using mathematically derived redundant abstractions of the content. Using an IAC, one transmits the abstractions (rather than the original content) from sender to receiver, and decodes the abstractions at the receiver to recover the original content.

The abstractions are not segments or replicas of specific portions of the original content. Rather, each abstraction is a partial description of the content. When a sufficient number of such abstract descriptions are encoded, transmitted, received and decoded, the effective resolution of the total description will be sufficient to fully reconstruct the original object.

As a useful analogy, some IAC technologies can be described as a series of random linear equations. For instance, consider a trivially simple digital object that consists of three integers: X, Y and Z. The following are linear equations (abstractions), each of which describe the content accurately but incompletely.

a)	$X + Y = 5$	d)	$X - Y = 1$
b)	$X + Z = 4$	e)	$X - Z = 2$

c) $Y + Z = 3$ f) $Y - Z = 1$

Rather than transmit the values of X, Y and Z (the content), an IAC-based scheme transmits the equations as abstractions of the content.

Because they are overlapping abstractions, not all six of the equations are required to reconstruct the content. *Any* three of the above six linear equations are sufficient to solve for X, Y and Z.

For instance, adding (b) and (e) yields:

$2X = 6$ or $X = 3$

Solving (b) for Z:

$3 + Z = 4$ or $Z = 1$

Solving (d) for Y:

$3 - Y = 1$ or $Y = 2$

Continuing with the linear equation analogy for the moment, consider the following attributes of a communications system that transmits such abstractions in lieu of the underlying data:

- For data containing n values (n=3 in the above example), precisely n unique equations must be received in order to reconstruct (solve for) the original data.
- Any n unique equations are sufficient to reconstruct the original.
- The ordering of the equations is not relevant. It is okay if the equations are received in an order different from the order in which they were transmitted. (The transmission channel does not need to preserve order.)
- If the equations are transmitted as a repeating stream, a receiver may "join" the stream (i.e., begin receiving) at any point in the sequence without penalty. (There is no beginning or end.)
- If equations are lost during transmission, they do need to be retransmitted, so long as the receiver eventually receives the required number of equations. It is receipt of an adequate quantity of unique equations—not the reception of any specific equation—that is significant. (The system has high immunity to packet loss.)

2.3.1 IACs for Content Delivery

IAC techniques can be used to deliver content over a connection-oriented protocol such as TCP, but doing so offers no advantage as compared to placing the content directly into TCP packets. But because of their tolerance for packet loss, IAC techniques can be used with a lower-overhead unreliable datagram transport such as UDP. When used over links with high rates of packet loss (5% or more) IAC over UDP is far more efficient than TCP. However, this technique still does not address scalability, as a unique stream of UDP packets must be transmitted from the server to each receiving client.

Where IAC techniques have been shown to be particularly scalable, is when used in conjunction with a multicast datagram protocol. In such cases, a small server may transmit a single stream of datagrams containing IAC-encoded data. The datagrams are

then fanned-out by multicast routers. Using this technique, many clients (potentially millions) can receive content originated by a single low-power server.

Two applications for the IAC/multicast combination are particularly appropriate: live streaming data and high-volume file distribution. In fact, even without IAC, multicast networks are appropriate for the transmission of live streaming data. However, because of the dropouts caused by even the slightest packet loss, traditional datagram-based streaming over multicast can be problematic. By breaking the streamed data into smaller pieces (perhaps 10 Kbytes to 100 Kbytes each), then encoding each such piece using an IAC algorithm, and finally transmitting somewhat more abstract data than would be required if no packet loss was anticipated, such a system can effectively overcome the problems caused by packet loss.

With traditional reliable transmission protocols such as TCP, recovery from packet loss requires that the system be able to determine which packets are lost. Using IAC techniques, only the number of packets lost is significant, and even then fairly high packet-loss rates can be overcome.

High-volume file distribution using IAC/multicast can be achieved by transmitting a continuous stream of content abstractions into the multicast network. When a client wants to download the file, it merely joins the multicast group and then begins to receive the stream of content abstraction packets. It continues to receive the packets until it has received enough to reconstruct the original file. This requires the reception of only slightly more data than if the file were transmitted without IAC encoding.

Using this technique there is no beginning or end of the data stream. It is continuous. A receiver may join at any time without penalty. A receiver may even join, leave and re-join at a later time, and all content abstractions received will be additive.

2.3.2 Simultaneous Reception from Multiple IAC Sources

Another significant feature of IAC technologies is that they allow data to be transmitted from more than one location or via multiple paths simultaneously without the typical expense of redundancy. Using IAC, one can use n paths without incurring n times the cost of single-path transmission.

Figure 4, illustrates this principle. Three senders are each transmitting a stream of randomly selected abstractions, labeled a through f as were the linear equations in the previous example.

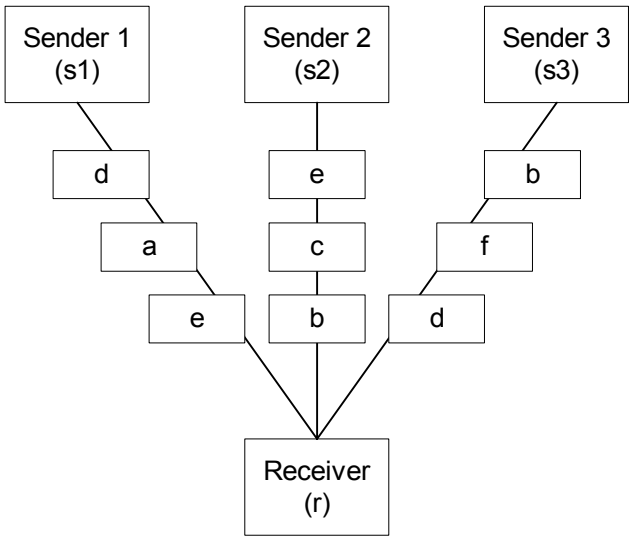

Figure 4

If the receiver (r) is capable of receiving from all three streams, the following are true:

- The data received by r is additive. Because they are randomly generated, the abstractions received from one sender tend not to duplicate the abstractions received from the others. (The example uses a very small set of abstractions, so duplication appears more frequently than with real-world algorithms.)
- If one or two senders or paths fail, the transmission of the data will still be completed.
- So long as the receiver's local communications capability permits it, the net throughput will be as high as three times the throughput experienced when receiving from a single sender.
- The system is quite insensitive to packet loss. Assuming the communications bottleneck is indeed at the receiver's end, if two-thirds of the packets are lost enroute (whether from one source or from multiple sources), r will still receive a sufficient number of abstractions to reconstruct the original data within a period of time not much longer than if no packets had been lost.

2.3.3 The Problem with IAC/Multicast Content Distribution

The problem with IAC/multicast content distribution is that multicast is not available via the public Internet. IAC/multicast can be used effectively within a privately controlled network such as a corporate-owned WAN or LAN or a network of caches or edge servers operated by single vendor. The owners of such networks have the ability to enable multicast features within their privately controlled routers and switches. This is not currently the case on the public Internet.

While IAC techniques can still be used without multicast over the public Internet to overcome the problems of high packet loss, this use of IAC offers little in the way of a solution to the general problem of content distribution and the particular challenges of scalability.

The proposed system, Peer-to-Peer Content Delivery Using Information Additive Codecs, is a solution to the general problem of content delivery. Using a peer-to-peer network (which can be deployed on the public Internet) instead of multicast technology, the system brings both scalability and resilience to packet loss to content delivery on the Internet.

3 P2P-IAC Content Delivery

The proposed system is a method of delivering content via a combination of P2P communications (either a P2P CDN or a network used for content sharing or collaboration) and Information Additive Codecs (IACs). This combination yields a new and unique solution to all of the problems described previously.

3.1 Components

There are, in fact, multiple existing technologies within each domain (P2P, CDNs and IACs) that may be used to manifest the proposed system, and those existing technologies are generally interchangeable. Specifically, there are many IAC algorithms that may be combined with one of many P2P technologies to build a working system based on the system.

What is unique about this system is the combination of these existing technologies.

The examples that follow are based on a single, non-specific component for each of these technologies. This not meant to imply either than these are the preferred choices or that they are in any other way unique. They are used for illustrative purposes only and to demonstrate the plausibility and practicality of the system.

3.2 Examples of P2P Content Delivery Using IACs

Initially, the requested object (represented by a small filled-in square in the figures that follow) exists only on the origin server (o, Figure 5). The first client (c1) requires a copy of the object, as indicated by the small empty square. c1 asks o, “Who has the object?” o replies, “You can find the object at o.” c1 then downloads the object from o.

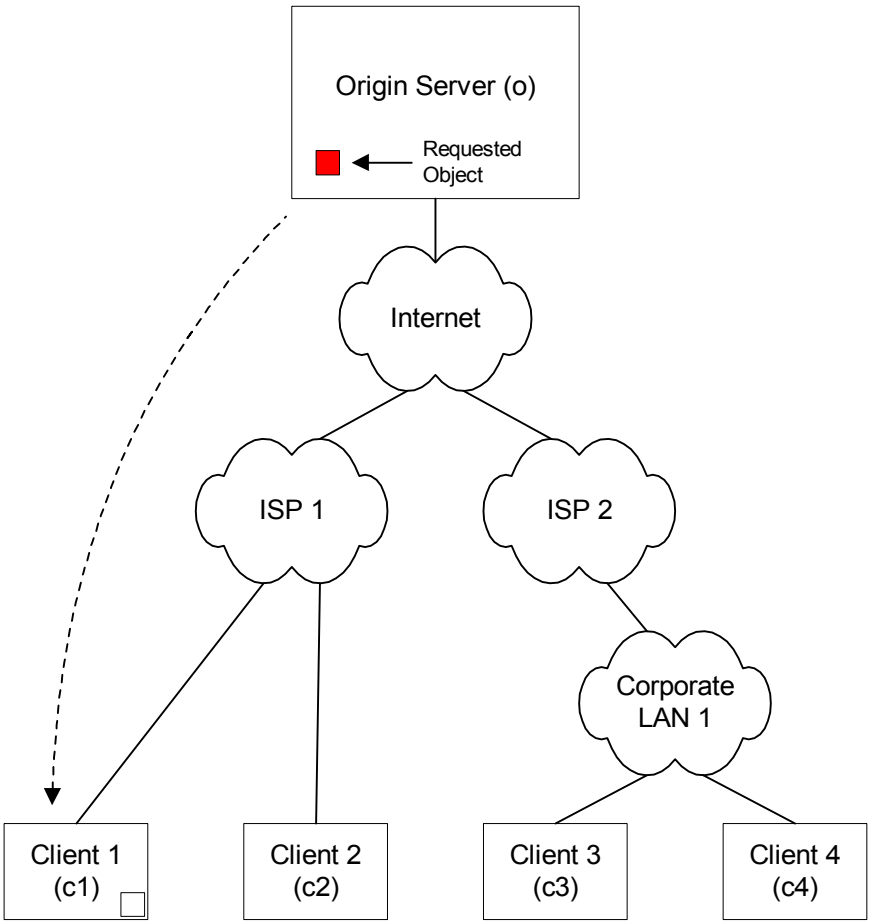
Origin Server (o)
Requested Object
Internet
ISP 1
ISP 2
Corporate LAN 1
Client 1 (c1)
Client 2 (c2)
Client 3 (c3)
Client 4 (c4)

Figure 5

In Figure 6, client 2 (c2) requires the same object. c2 asks o, “Who has the object?” o replies, “You can find the object at o and c1.” c2 then initiates downloads of the object from both o and c1 simultaneously. However, because the c1/c2 path may be more efficient (lower latency and packet loss) than the o/c2 path, more data may be sent per unit time over the c1/c2 path than over the o/c2 path. Once c2 has received enough total data to reconstruct the requested object—accumulated from both sources—c2 terminates the downloads from o and c1.

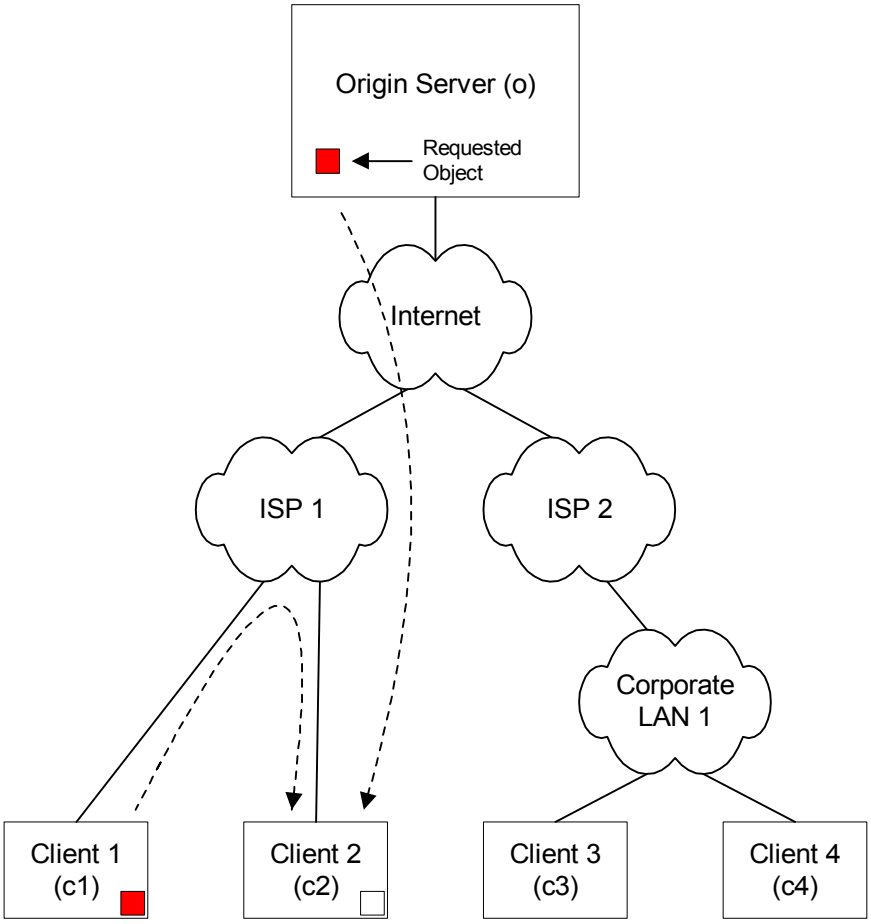

Figure 6

If the performance of the c1/c2 and o/c2 paths was identical, the system will have achieved a roughly 50% reduction of the load placed on the origin server, o. If the c1/c2 link was more efficient than the o/c2 link, the reduction of server load will be correspondingly greater than 50%.

In Figure 7, client 3 (c3) requires the same object. c3 asks o, “Who has the object?” o replies, “You can find the object at o, c1 and c2.” C3 then initiates three downloads of the object: from o, c1 and c2 simultaneously.

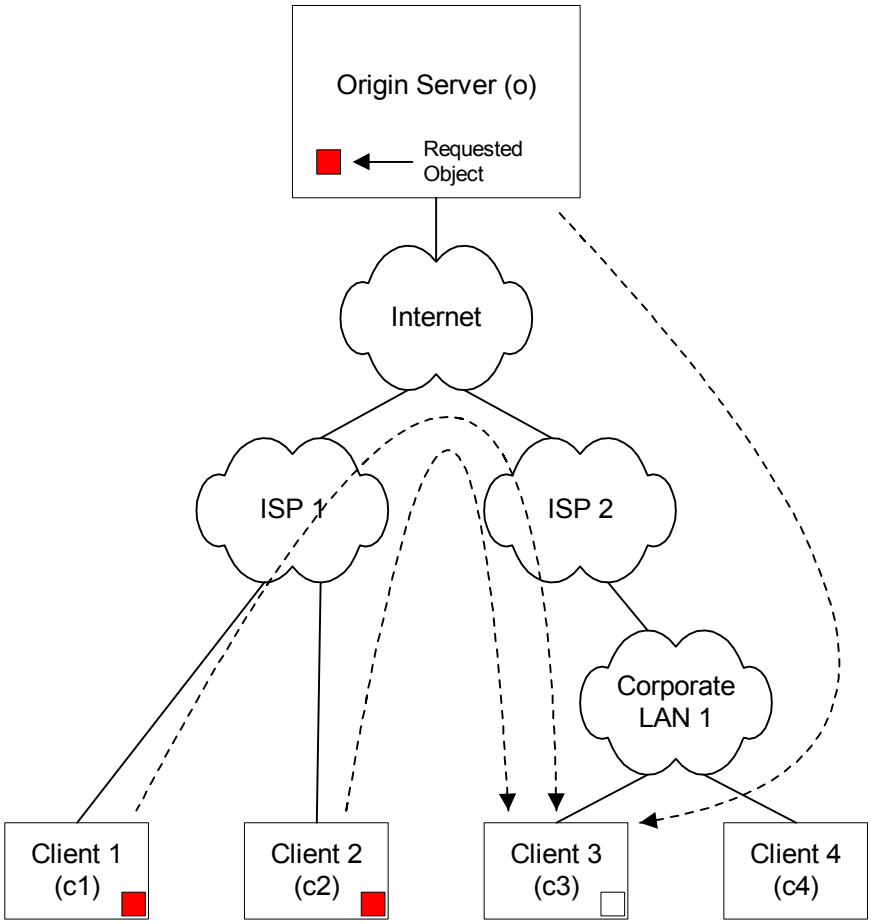

Figure 7

As with the previous example, all three downloads will be terminated as soon as c3 has received sufficient data from the combined downloads to reconstruct the requested object. In this instance, the overhead to the server is reduced even further. Assuming all three paths are equally efficient, the server will incur only one-third of the overhead it would experience if it were to process the entire request on its own.

Figure 8 illustrates a somewhat different situation. In this instance a new client (c4) is located on the same high-speed local area network (LAN) as another client that already has a copy of the desired object. As before, c4 asks o, "Who has the object?" o replies, "You can find the object at o, c1 and c3." c4 then initiates three downloads of the object: from o, c1 and c3 simultaneously.

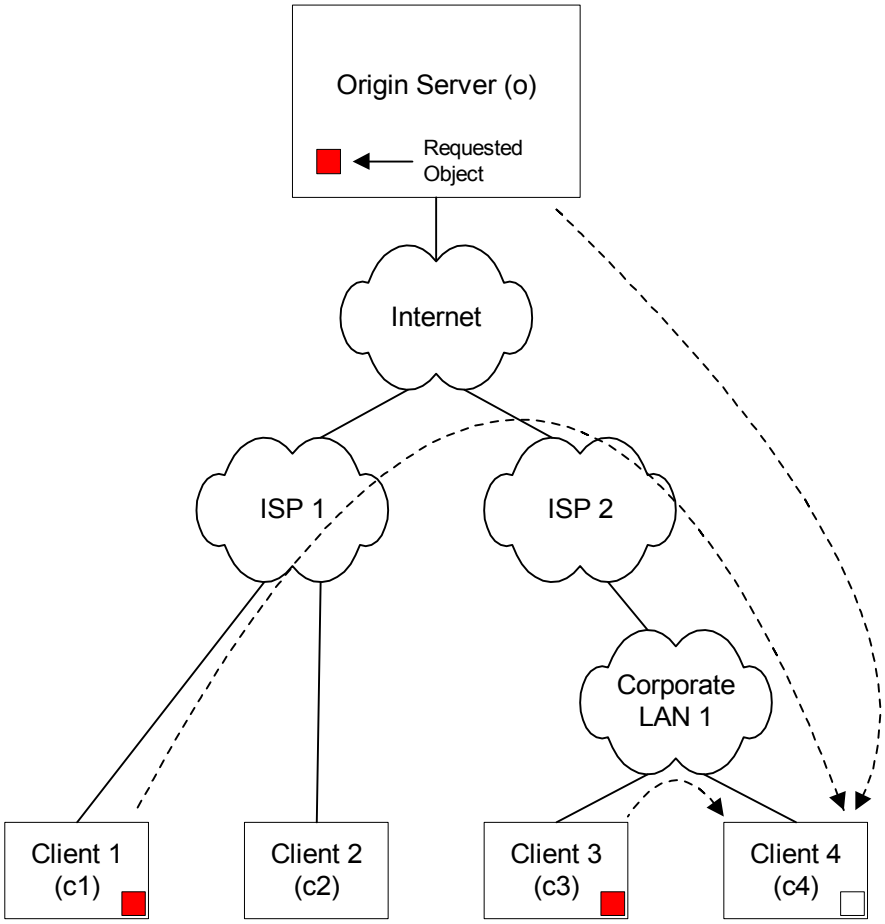

Figure 8

In this case, however, the performance of the c3/c4 link is vastly superior to either the c1/c4 or o/c4 links. Therefore, although three downloads are initiated, virtually all of the data needed to reconstruct the requested object is received from c3. Again, once sufficient data has been received, the downloads from c1 and o are terminated. The c1/c4 and o/c4 paths will barely have been used.

Note that the receiving client not only receives data via the best available path, it accumulates data from all sources, even those that are relatively inefficient.

Figure 9 illustrates yet another variation. In this instance, c4 asks o, "Who has the object?" This time, however, the origin server (o) decides based on its knowledge of which clients have copies of the requested object that the server (o) does not need to include itself in the list of sources. (There are many algorithms for making such a decision, and none is specified or unique to this system.) o replies, "You can find the object at c1, c2 and c3." c4 then initiates three downloads of the object: from c1, c2 and c3 simultaneously.

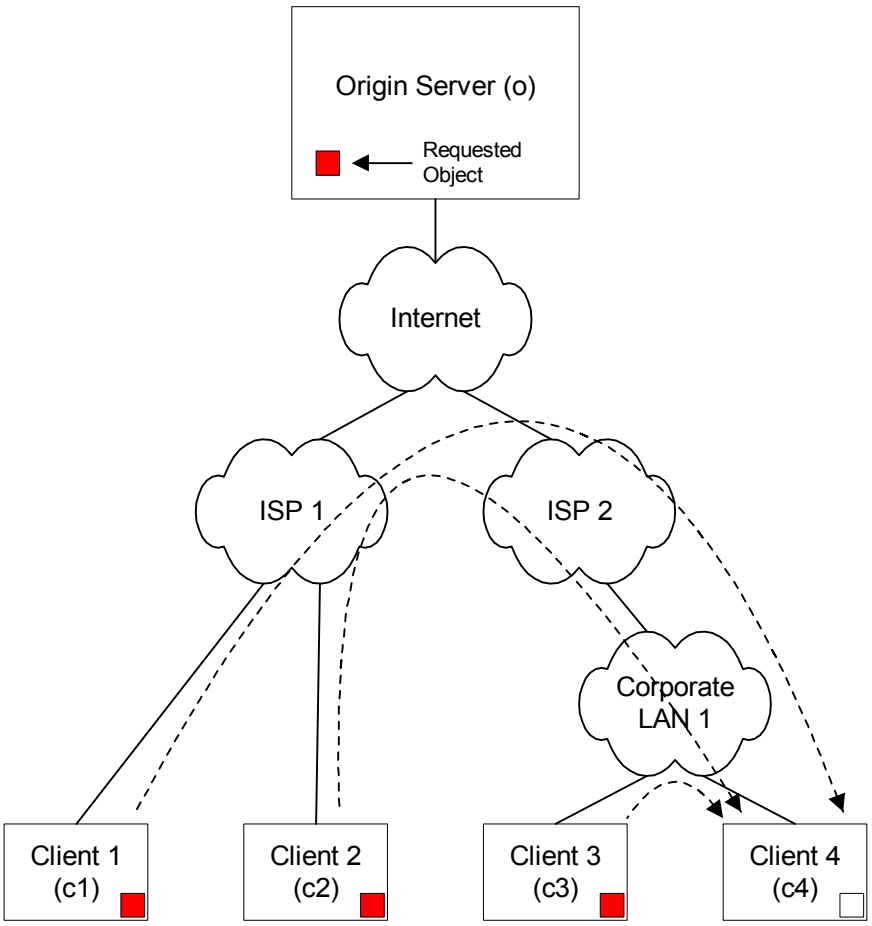

Figure 9

In this instance the overhead on the server has been reduced by nearly 100%, as all three of the simultaneous downloads requested by c4 are from other clients.

The more clients that request, retain and therefore can re-deliver an object, the lower the percentage of downloads that must be requested from the origin server. The system scales far better than linearly.

Once the system is servicing the needs of moderate number of clients, the origin server will primarily take on the role of traffic cop rather than content server. The traffic-cop role requires far fewer computer resources than serving the content.

The traffic cop must direct clients requesting objects to other clients from which the objects can be downloaded. There are two issues associated with providing such information. First, the traffic copy must have some way of knowing which clients are capable of supplying the object. (Is the client on line? Does it have a copy of the desired object?) Second, the traffic cop must be able to recommend clients that are in close topological proximity to the requesting client. (e.g., It is important to recommend other clients on the same LAN if possible, rather than those that are far away.)

3.3 Locating Objects in a P2P Network

There are many schemes for determining the availability of peers to deliver objects, and the system does not depend on the use of any one of the schemes in particular. However, to show plausibility, I shall present one such scheme that may, in fact, have certain unique and valuable characteristics.

The scheme creates *preference vectors* for peers that are candidates from which a client can request a given object. In practice, a client would issue simultaneous requests to multiple (at least more than one) peer.

A preference vector is calculated by combining two other vectors. The *availability vector* approximates the likelihood that a specific peer is capable of supplying and transmitting the specific object. The *topological proximity vector* approximates the performance-based distance between the peer and the requesting client.

These vectors may be stored, manipulated and queried on the server, clients or both.

3.3.1 Availability Vector

Availability consists of two elements: (1) the ability of the peer to transmit objects in general, and (2) the presence of the specifically requested object on that peer. A peer may not be able to transmit for a variety of reasons such as the presence of a prohibiting firewall or proxy server, a disabled communications connection (e.g., the modem disconnects), the sudden halting or de-installation of the appropriate software, or the user merely turning off the computer. Presence of the object on the peer is straightforward.

In lieu of far more complex algorithms, consider one that is quite simple. It makes use of the probability that the clients that recently requested an object tend to be the most likely to be able to re-deliver that same object to peers. This is essentially a least-recently used (LRU) algorithm. One way to implement such a system is to maintain a table that contains the following columns:

Client IP Address Object Requested URL Timestamp

Only the most recent requests are retained in the table. When the server receives a new request, the server assumes that the requesting client will be able to serve as a peer server for that object and enters the request into the table. If the table is full, the oldest entry in the table is first deleted to make room.

The availability vector for client/object pair is the number of seconds that have elapsed since that client requested the object. For instance, if the client at IP address 205.167.241.193 requested http://www.rds.com/images/logo.gif at 11:04:32 and it is now 11:12:42, the availability vector is 490. Below are sample availability vectors for requests of an object from four different clients:

a) 205.167.241.193	490
b) 205.167.253.85	18
c) 207.85.32.6	1,015
d) 217.165.120.66	7

3.3.2 Topological Proximity Vector

As the availability vector represents the inverse of the likelihood that a particular client will be able to deliver a specific object, we also need a way to rank possible clients according to their topological proximity to the requesting client. Topological proximity is an attempt to measure the performance of the path between two systems according to a variety of factors such as distance, speed of intervening connection, packet loss and latency on that connection, etc.

There are many algorithms for computing or estimating topological proximity, and many of those algorithms may be substituted for the following, which is simple and efficient to implement.

The proposed algorithm uses arithmetic closeness of IP addresses as an approximation of topological proximity. This works for a variety of reasons:

- Devices on the same LAN tend to have IP addresses that are arithmetically close to their own.
- Devices connected to the same ISP or corporate WAN also tend to have IP addresses that are arithmetically close to their own, but not as close as those on the same LAN.

At its highest levels, the IP address space is assigned to countries and other entities. Therefore, practical implementation of this algorithm suggests that a table of the most significant IP address octet values be used to adjust the topological proximity calculation for devices whose IP addresses differ in the most significant octet.

The calculation of topological proximity between two devices is done according to the following. Assume the two devices have IP addresses: a.b.c.d and A.B.C.D, respectively.

$$proximity = w*lookup(a,A) + x*(abs(b-B)) + y*(abs(c-C)) + z*(abs(d-D))$$

Assume, for instance that the lookup() function returns values in the range of 0-10, where 0 is returned if a=A, and higher values are returned as the distance between the geopolitical entities assigned to manage the address space likewise increases. Assume, also, the following constants:

w=16,777,216 x=65,536 y=256 z=1

Then consider the following examples of pairs of IP addresses and the resulting topological proximity vector:

a) 205.167.241.21 and 205.167.241.193 (on the same LAN) = 172
b) 205.167.241.21 and 205.167.253.85 (using the same ISP) = 3,136
c) 205.167.241.21 and 207.85.32.6 (on different U.S. ISPs = 5,427,471
d) 205.167.241.21 and 217.165.120.66 (in different countries) = 67,270,957

Note that these calculations are not intended to provide an approximation of physical distance. The last two devices are not necessarily one hundred thousand times farther apart than the first two, for instance. However, the values can be used to rank potential peers according to an approximation of their topological proximity to the requesting client.

3.3.3 Preference Vector

With two vectors, one for availability and the other for topological proximity, we can now combine them to produce a third vector that can be used to determine which peers should be recommended as the source of an object for a particular client. The resulting preference vector can be computed as follows:

$$\text{preference} = j*\text{availability}^2 + k*\text{proximity}$$

Assume, for instance, the following values:

j=1 k=1

Now let's compute the four preference vectors for our for prospective peers:

IP Address	Availability	Proximity	Preference
a) 205.167.241.193	490	72	240,172
b) 205.167.253.85	18	3,136	3,460
c) 207.85.32.6	1,015	5,427,471	6,457,696
d) 217.165.120.66	7	67,270,957	67,271,006

By the above calculations, device (b) would be given the highest preference and device (a) would be ranked next highest. Device (d) requested the object very recently, but because that device is likely to be far away, it is ranked last.

A system using the above rankings that attempted two simultaneous downloads would request those downloads from hosts (a) and (b).

Although the preference vectors should be as accurate as possible, accuracy at the expense of performance (calculation, storage, etc.) is not critical. For instance, suppose that host (a) in the above example is, in fact, able to deliver content faster than host (b). Since both hosts were selected, the actual ordering of the two is of no consequence. The

system could be tuned heuristically, by tracking the performance of downloads from each pair of peers. However, due to the constantly changing nature of peer-to-peer performance, such optimizations should be pursued conservatively.

Note again, that this mechanism of vectors is but one possible method of determining from which peers a given client should request a given object.

4 Peer and Server Engines

There are two primary software modules, the server engine (SE) and the peer engine (PE). Each is explained in detail later in this presentation.

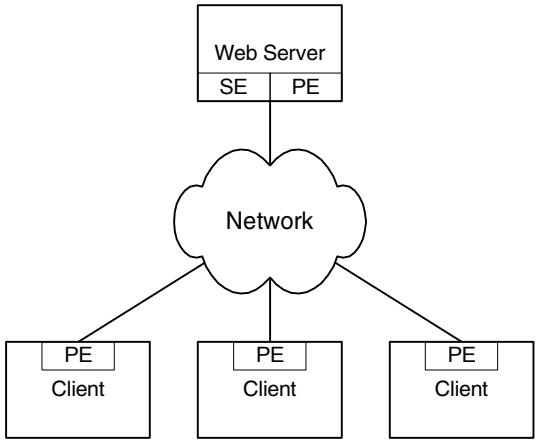

Figure 10

As illustrated in Figure 10, the PE is installed on any device that needs to retrieve objects and/or is willing and able to deliver objects to other PEs. It consists of two primary asynchronous tasks or threads. The first is the retrieval mechanism that requests and receives objects from peers. The second is the sharing mechanism that makes such objects available to peers. In some cases, as we shall discuss later, a device may retrieve without also sharing (i.e., a read-only peer), and therefore need not implement the sharing mechanism. Note that devices such as web servers may include an implementation of the PE sharing mechanism so that they can deliver objects using P2P-IAC.

The SE is installed on devices that serve as moderators for a set of PEs. The SE may be installed on a web server or other device (e.g., a dedicated server or a reverse-proxy cache) that is specific to one or more web sites. In this case, the SE will offer P2P-IAC moderation services to any PE that requests content from the particular web site.

An SE may also be installed on a proxy cache (e.g., within a corporate firewall for corporate users), within an access cache (owned and operated by an ISP for its customers), or within a content delivery network edge server. In these cases the SE moderates peering between a limited set of PEs (the corporate users, ISP customers, etc.) but for any web site that supports P2P-IAC. We will explore these configurations later.

4.1 Caches, Tables and Databases

The system includes the following data structures:

4.1.1 Peer Object Cache

Objects downloaded by a PE are stored in a least-recently-used (LRU) cache for sharing with peers or for later use by applications (e.g., web browsers) on the local host. Objects may be stored encoded (optimized for transmission), decoded (optimized for use) or both.

This is an implementation-specific decision and does not affect the ability of an implementation to interoperate with others.

The Peer Object Cache honors HTTP and other caching directives of content servers.

4.1.2 Server Table

A PE maintains a cached list of servers—not SEs—for the purpose of remembering which servers support P2P-IAC and which do not. Once the PE learns that a given server (identified by host name) does not support P2P-IAC, the PE will not bother to ask for such services from that server in the future, thereby ensuring that traditional non-P2P-IAC operations are not impeded by the presence of the PE. Some PEs may also track servers that do support P2P-IAC, but that is not strictly necessary, as the protocol will always attempt to communicate with an unknown server.

The server table must be flushed from time to time in order to allow hosts that add P2P-IAC capabilities to be recognized as such. The specific mechanism for this is not critical to the system, but may, for instance, be associated with the aging/flushing of a local web browser's object cache.

4.1.3 Peer Performance Cache

A PEs maintains an LRU table of peers from whom the PE has recently requested downloads. Each row of the table contains a peer's IP address and the most-recent download performance of that peer in bytes-per-second. This is used for subsequently optimizing peer selection.

4.1.4 Availability Table

This is a database of Availability Vectors (see page 20) maintained by each SE to keep track of the availability requests it receives from specific PEs for specific objects.

4.2 Availability Lists

An Availability List (AL) is an XML object sent from an SE to a PE that answers the PE's question, "From where can I download a specific object, and how large is it?" The AL includes the size of the object (in bytes), the IP address of other PEs that have recently requested an AL for the same object, the IP port on which each is willing to accept a download request, and the time (in seconds) since each of those requests was received.

The following is an example of an AL response. It uses version 1.0 of the P2P-IAC protocol and reports on peers that have recently requested the object http://www.rds.com/images/logo.gif. Three hosts are listed along with the time in seconds since each requested an AL for the object. The order of hosts is not significant.

```
<?xml version="1.0" ?>
<!-- P2P-IAC Availability List -->
<p2p-iac version="1.0">
  <al url=http://www.rds.com/images/logo.gif size="3126">
    <host ip="167.160.32.6" port="43" seconds="42"/>
    <host ip="152.143.64.5" port ="3001" seconds="16"/>
    <host ip="122.12.192.133" port="3023" seconds="1436"/>
```

```
  </al>
</p2p-iac>
```

In the case of streaming object types (whether live or on-demand), the <al/> tag specifies the segment size rather than the absolute object size. For example:

```
  <al url=http://www.rds.com/video/live_audio.mp3 segmentsize="10240"/>
```

4.3 AL Request/Response Protocol

This is a protocol between PEs and SEs. It is based on HTTP. The semantics of the request allow a PE to ask a specific SE, "From where can I download the requested object using P2P-IAC?" Optionally—if the requesting PE is able and willing to provide the requested object to peers—the request includes the IP port number to which download requests should be sent.

The response contains an Availability List (AL) in XML. (See page 26.)

(Note: In the case of an SE implemented within a cache server or similar device, the response to an AL request may come from the cache server's SE rather than from the SE to whom the request was addressed.)

The request is an HTTP GET request with a query string, sent by the client to the server. If standard (i.e., non-P2P-IAC) URL of the object is of the form:

```
  http://<host>[:<port>]<path>
```

the AL request URL is of the form:

```
  http://<host>[:<port>]/P2P-IAC-AL?path=<path>[&port=<download_port#>]
```

(Note: This is a preliminary format of the request to be used during development. Other potential formats including HTTP POSTs may ultimately be used.)

The <download_port#> specifies the IP port number on the PE which to which download requests are to be sent. Omitting the <download_port#> implies that the requesting PE is not capable of accepting (or is unwilling to accept) download requests. (i.e., It is a read-only peer.)

Any standard HTTP response code may be returned. If the object does not exist on the server, the server will respond with HTTP code 404—File Note Found. Note that this is also the error code that will be returned if P2P-IAC is not supported by the server.

If the object does exist and can be delivered via P2P-IAC, the server responds with HTTP code 200 and an Availability List (AL) in XML format. (See page 26.)

(Note: Because any HTTP client may request an AL, anyone (or any system) can learn which hosts have recently requested any given object. Some may consider this to be a

security/privacy risk. Potential schemes for authentication and/or encryption of this data should be considered.)

4.4 Download Protocol

This is the protocol by which one PE (the requester) may request and receive a copy of an object from another PE (the responder). The protocol is based on the User Datagram Protocol (UDP). It is an unreliable connectionless protocol. Packets may be lost, received out of sequence or received in duplicate. There are three packet types:

- Download Begin Request
- Download Negative Response
- Data Transfer
- Download End Request

4.4.1 Download Begin Request (DBR)

The requester sends a single UDP datagram to the responder at the IP address and IP port learned from a recently-received Availability List (AL) for the desired object. The datagram contains an XML object in the following form.

```
<?xml version="1.0" ?>
<!-- P2P-IAC Download Begin Request (DBR) -->
<p2p-iac version="1.0">
  <dbr url="http://www.rds.com/images/logo.gif" id="123" maxspeed="100000"/>
</p2p-iac>
```

The <url> attribute is required and specifies the object to be downloaded. The URL is that used to reference the original object. (i.e., The host name, directory structure, etc., are not those of the requester or the responder unless the responder happens to be running on the origin server.)

The <id> attribute is required. It is an arbitrary positive integer less than 2^{31} that is used to identify response packets as being associated with the requested object.

The <maxspeed> attribute is optional. If supplied, it specifies the maximum rate (in bytes per second) at which the responder should transmit data, when averaged over a relatively long period such as a second. This is not a flow-control mechanism, for such is not possible when using UDP. The intent, however, is to minimize waste of bandwidth. For instance, if a requestor is connected to a network at 56,000 bits per second (approximately 7,000 bytes per second), a majority of packets sent by a responder at one million bits per second (125,000 bytes per second) may be lost and therefore waste responder and network resources. The algorithm for specifying the <maxspeed> attribute is not specified, but there is no reason for it to ever be greater than the speed of the requestor's connection to the network. If connected to a responder via a 10baseT (10 megabit) LAN, a requestor might specify maxspeed=1250000. When downloading from another responder via a T1 Internet connection, the same requestor might specify maxspeed=193000.

4.4.2 Download Negative Response (DNR)

[*Note: I originally thought it would be best for a PE to simply ignore Download Begin Requests (DBRs) that would cause the limit of downloads to be exceeded, that were for objects that were either not found in the Peer Object Cache, or for which the copy of the requested object had expired. Given the redundancy of multiple simultaneous downloads and the potential for abuse via denial-of-service (DoS) techniques, it seemed that simply not responding would be sufficient and even desirable. The requesting PE would still receive data from other downloads. Furthermore, the unresponsive peer would be flagged by the requesting PE as such, and subsequent requests from the same PE would, therefore, be less likely.*

However, one scenario convinced me of the need for a somewhat more positive NAK (negative acknowledgment). If the requesting PE selects only one peer from which to download an object, and that peer doesn't respond for the reasons stated above, the only fallback is for the requesting PE to timeout the download, and request the object from the origin server. Given the relatively unstable nature of peers, this seems too risky. …drk 9/20/01]

If a responder is unable to supply the requested object, it transmits a Download Negative Response (DNR) to the requester. Typical reasons for such a rejection include too many current simultaneous downloads on the responder, or the requested object either not found in the local Peer Object Cache or expired.

A DNR is a single UDP packet containing the following:

```
<id>
<#FFFF>
```

The <id> is a 32-bit unsigned integer transmitted in network-standard byte order. It is the value of the <id> attribute received in the Download Begin Request (DBR). The requester uses it to determine which download request is being rejected.

<#FFFF> is a 32-bit string of "1" bits (signed –1 value). This is a semantic convention implying a failed or rejected download, and is used to distinguish the DBR packet from a valid Data Transfer (DT) packet. (See page 29.)

[Note that the use of #FFFF in the same position as a valid Data Transfer (DT) packet's <sequence_number> means that #FFFF may never be used as a <sequence_number> in a DT Packet. …drk 9/21/01]

4.4.3 Data Transfer (DT)

The responder sends a continuous stream of UDP datagram packets to the responder, addressed to the same IP address and port number from which the DBR was sent. Each DT packet contains the following:

```
<id>
<sequence_number>
<data>
```

The <id> is a 32-bit unsigned integer transmitted in network-standard byte order. It is the value of the <id> attribute received in the Download Begin Request (DBR). It is used by the requestor to segregate data received from multiple simultaneous downloads.

(Note: Some implementations of the requestor may use a single IP port to receive download protocol packets from all peers and downloads. In such cases, the <id> field is crucial for segregating objects. Other implementations may use a separate IP port for each download, and therefore be able to ignore the <id>.)

The <sequence_number> is a 32-bit unsigned integer (initially zero) for the download of each object, incremented for each DT packet, and transmitted in network-standard byte order. It allows the receiver (requester) to analyze packet loss and perform other diagnostics. It has no other purpose. (e.g., There is no concept of retransmission of lost packets.) The <sequence_number> has no semantic relationship to the data. For streams or extremely large objects (many gigabytes), the number will wrap to zero once the value 2^{31} is exceeded.

The <data> portion of the packet contains a randomly selected portion of the IAC-encoded object. The number of bytes of data within each packet is determined according to the IAC codec selected and MTU (Maximum Transmission Unit) considerations of the network and interface.

The responder should check regularly—perhaps after the transmission of each DT packet—for a receipt of a Download End Request (DER) packet.

The responder should also attempt to honor the <maxspeed> attribute of the Dowload Begin Request (DBR) by counting bytes transmitted and possibly delaying between packets.

4.4.4 Download End Request (DER)

To terminate a download, the requester sends a single UDP datagram to the responder at same the IP address and IP port to which the DBR request was sent. The datagram contains an XML object in the following form.

```
<?xml version="1.0" ?>
<!-- P2P-IAC Download End Request (DER) -->
<p2p-iac version="1.0">
  <der id="123"/>
</p2p-iac>
```

The <id> attribute is required. It is the same value used to initiate the specific download.

4.4.5 Download Protocol Notes

1. If no response is received to a Download Begin Request (DBR), the requester should not issue a second request. There is no way to determine whether the request was lost, delayed or ignored by the responder. In any case, the responder should be considered (at least temporarily) unsuitable. Sending a second DBR

could initiate a second download, and in any case is an improper request of the responder's resources.

2. Likewise, there is no guarantee that a Download End Request (DER) will be received, but unlike the DBR there is relatively little harm in issuing a follow-up. If a requester detects that it is receiving Data Transfer (DT) packets for far longer than it should (taking into consideration buffering in the network and protocol stacks), subsequent DERs may be issued sparingly.
3. Note that both DBR and DER packets are formatted in XML. This is so that they can be distinguished from one another when arriving at the same IP port. However, since DT packets are the only UDP packets that should arrive at the requester's IP port, there is no need to identify them further.

4.5 Peer Engine Operation (Retrieval Mechanism)

Figure 11 illustrates the operation of the retrieval mechanism of the PE. A description follows.

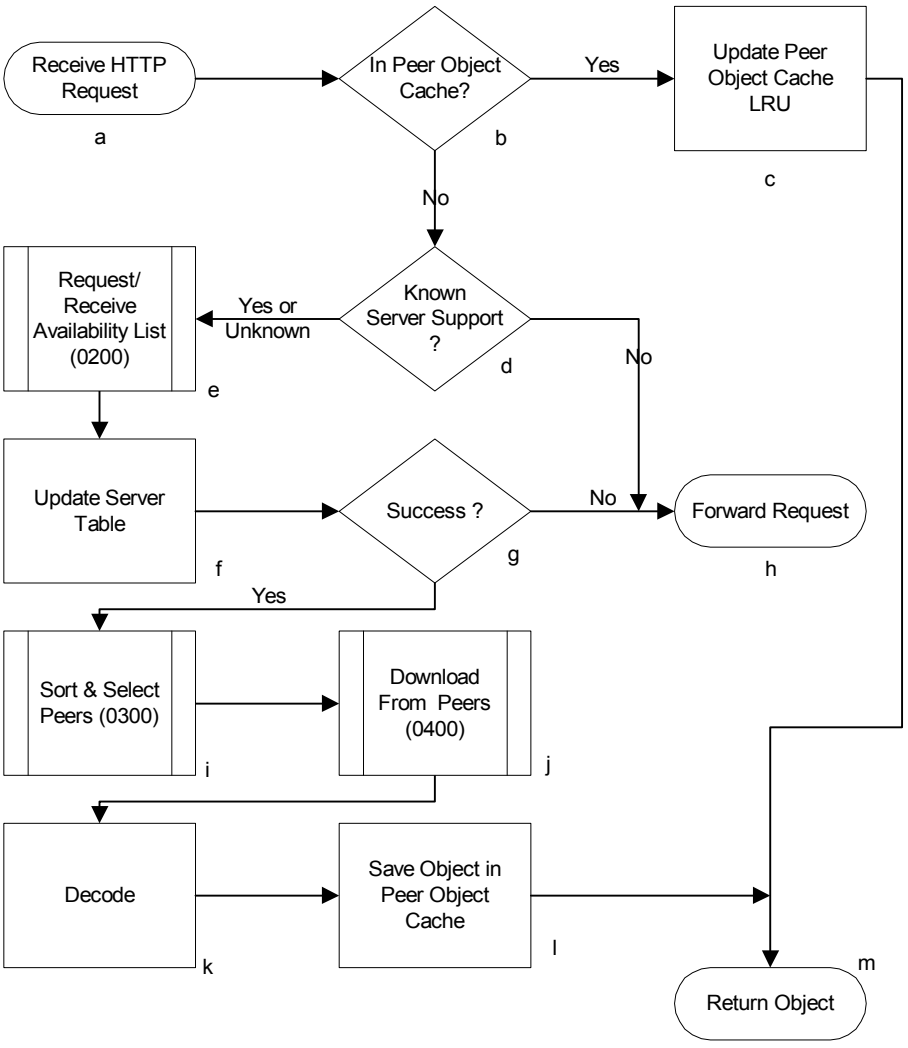

Figure 11: Peer engine—Retrieval Mechanism

Referring to Figure 11: Peer engine—Retrieval Mechanism:

(a) Processing begins when the peer engine (PE) receives a request to retrieve a remote object. For instance, this may be a request from the local web browser (e.g., running on the same device) for an HTML page or an image.
(b) The PE checks to see whether the requested object is stored within the PE's local peer object cache and whether such a stored copy is current (i.e., not expired).
(c) If a current copy of the object is found in the local peer object cache, the cache's least-recently-used (LRU) database is updated to indicate that the object was requested and the PE continues at (m).
(d) Otherwise (i.e., no current copy of the object was found in the cache) the Server Table is checked to see whether the remote server is known to support the P2P-IAC protocols.
(e) If the server is known to support P2P-IAC or its ability is unknown, the PE uses the AL Request/Response Protocol to attempt to retrieve an Availability List (AL) for the object from the server. (See page 27.)
(f) The Server Table is updated to reflect the ability of the server to support P2P-IAC.
(g) If an Availability List (AL) was successfully retrieved, processing continues at (i). Otherwise processing continues at (h).
(h) If the server was already known not to support P2P-IAC or if the request to retrieve an AL failed, the PE forwards the original HTTP request directly to the remote server. The PE bypasses its remaining logic and causes the request for the object to be processed as though the PE were not present. For example, if installed in conjunction with a web browser, the web browser's request is now forward to the web server and the server's response is sent directly to the web browser (not to the PE) using traditional protocols.
(i) In the case where an AL was successfully retrieved, the PE combines the AL with locally-maintained data and selects the peer(s) from which it will request the object. (See page 35.)
(j) The PE downloads the object from the selected peer(s). (See page 36.)
(k) The downloaded object is decoded using the selected IAC codec.
(l) A decoded copy of the object is saved in local cache. (Note: An encoded copy may also be maintained in order to improve the performance of replies to requests for the object from other peers.)
(m) The decoded object is returned to the requesting entity (e.g., web browser).

4.5.1 Availability List Request (0200)

Figure 12 illustrates the mechanism by which a PE requests an Availability List (AL) from an SE.

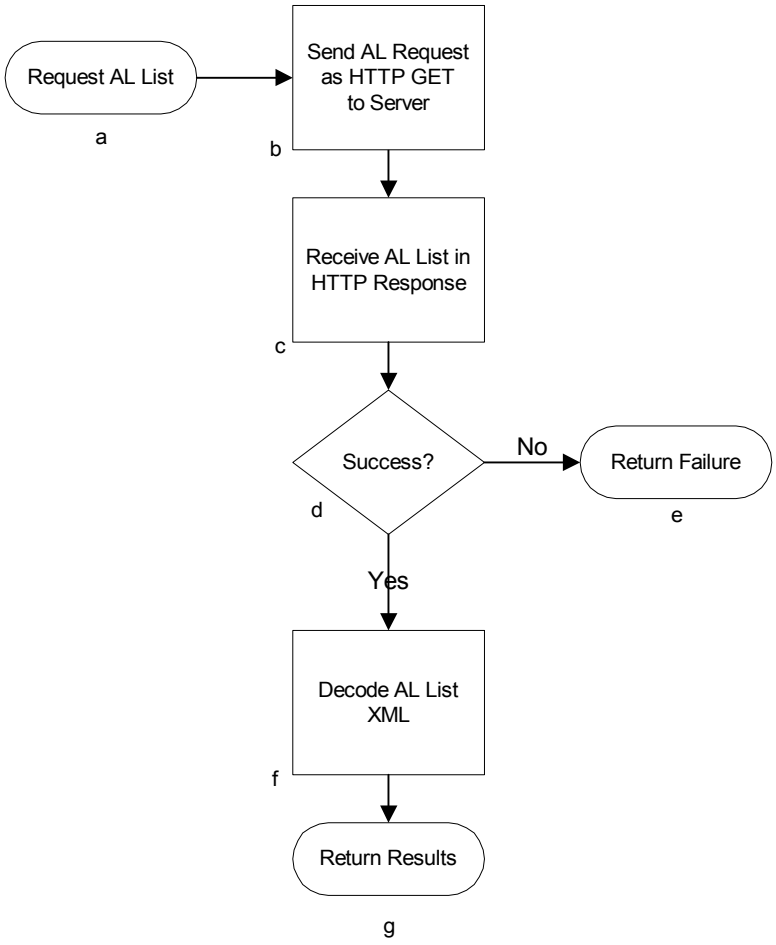

Figure 12: Availability List Request (0200)

Referring to Figure 12: Availability List Request (0200):

a) This sub-function is invoked when a peer engine (PE) needs to retrieve an Availability List (AL) for a specific object on a specific server for which the PE already has a conventional URL.
b) The PE creates an AL Request (see pages 26-27) and transmits it to the server.
c) The PE receives an AL Response (again see pages 26-27).
d) An HTTP error 404—File Not Found or a timeout of the attempted reception indicates an error.
e) If an error occurred, this sub-function returns a failure indication, implying that for one reason or another the requested server is unable to provide an Availability List for the requested object.

f) If no error occurred, the XML in the AL Response is decoded.
g) The sub-function returns the Availability List with an indication of success.

4.5.2 Sort and Select Peers (0300)

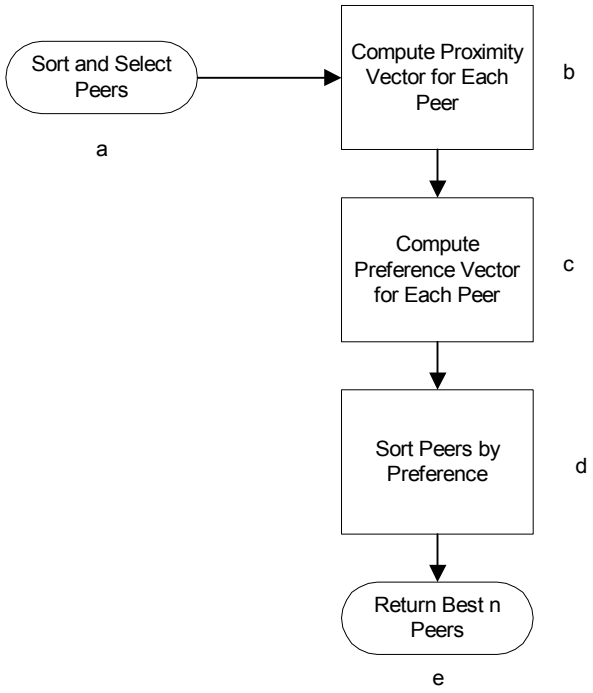

Figure 13: Sort and Select Peers (0300)

Referring to Figure 13: Sort and Select Peers (0300):

a) The sub-function is invoked by the peer engine (PE) once it has received an Availability List (AL) from a P2P-IAC-compatible server in response to an AL request for a particular object.
b) A Proximity Vector (V_{prox}) is computed for each potential peer host returned in the AL response. The V_{prox} is a single numeric value derived by comparing the PE host's IP address to that of the potential peer host. The larger the V_{prox} value, the topologically farther the system assumes the associated potential peer to be from the client. (See page 21 for an example of possible algorithms for computing V_{prox} vectors.)
c) The V_{avail} vector (from the AL response) and the V_{prox} vector for each potential peer host are combined to generate a Preference Vector (V_{pref}) for each host. (See page 23 for an example of possible algorithms for computing V_{pref} vectors.)
d) The resulting V_{pref} vectors are sorted in most-preferred-first order. (As an optimization, only the n-most preferred hosts need to be sorted. See below.)
e) IP addresses of the n-most preferred hosts are returned. n is a configurable value of the number of peers from which this PE will download objects simultaneously.

4.5.3 Download from Peers (0400)

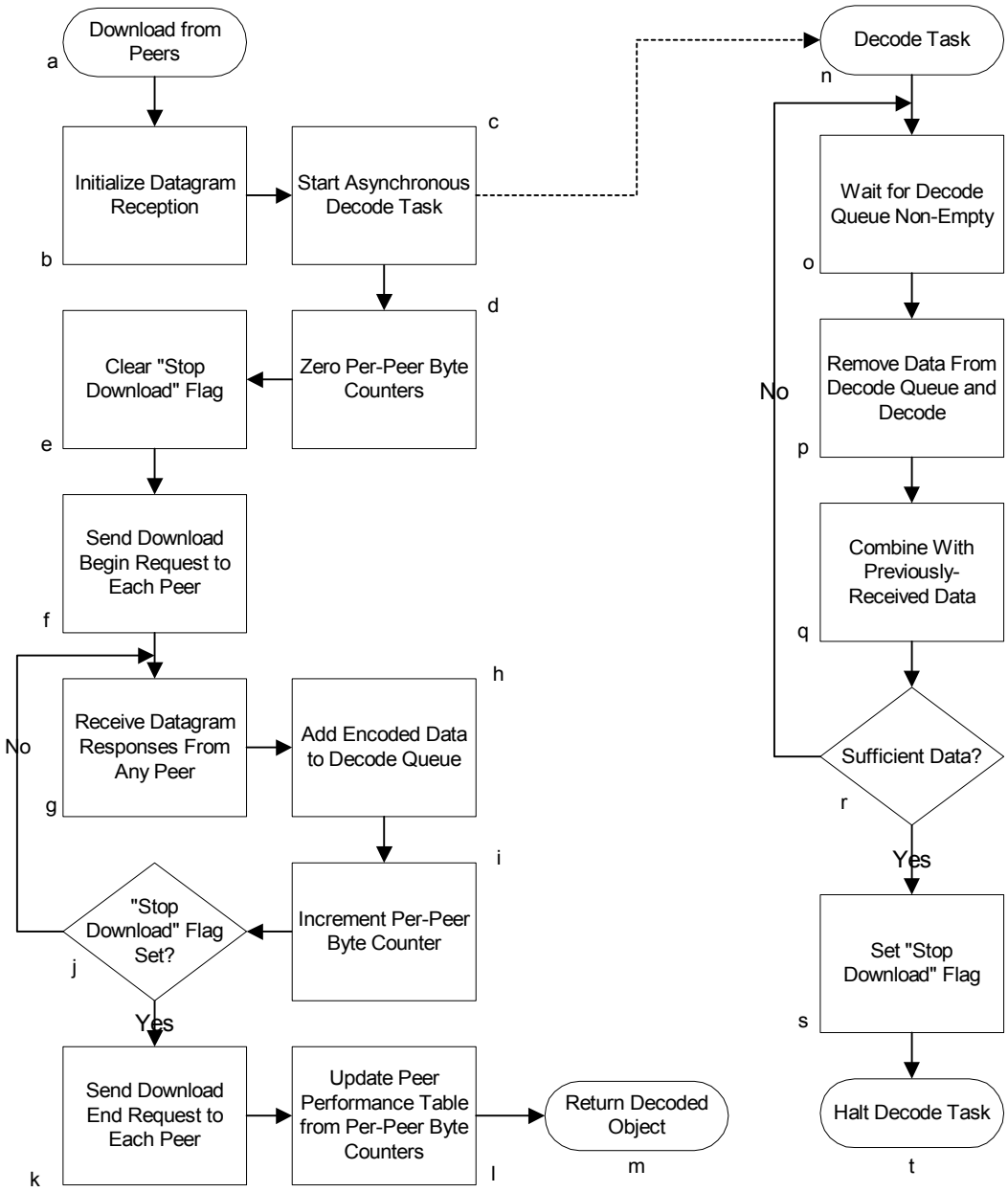

Figure 14: Download from Peers (0400)

Referring to Figure 14: Download from Peers (0400):

Main Task:

a) The sub-function is called by a peer engine (PE) with a list of one or more peer(s) from which the PE wants to download an object.
b) One or more communications ports (or sockets) on the local host are opened or otherwise made ready to receive UDP datagram packets.

c) The main PE task creates or activates a separate, asynchronous Decode Task at (n).
d) Temporary byte counters—one per peer—are initialized to zero. Once the object download is complete, these will be used to analyze the effectiveness of each peer's ability to communicate with the PE.
e) A Stop Download Flag—used for inter-task communications—is cleared.
f) A Download Begin Request (DBR) is sent to each selected peer. (See page 28.)
g) Data Transfer (DT) datagrams (see page 29) are now received from any peer in any sequence.
h) Data received from valid DT datagrams are added to a Decode Queue, shared by the two tasks.
i) The count of data bytes in each valid DT datagram are added to the temporary byte counters maintained for the associated peer.
j) If the Stop Download Flag has been set by the Decode Task, processing for the main task continues at (k). Otherwise processing repeats at (g).
k) Since sufficient data has been received (as indicated by the Decode Task setting the Stop Download Flag) a Download End Request (DER) is sent to each peer. (See page 30.)
l) The per-peer number of bytes-per-second are computed according to the data received from each peer. These values are stored in the Peer Performance Cache (see page 25), possibly overwriting existing entries.
m) The decoded object is returned to the caller.

Decode Task:

n) The decode task begins or is re-activated.
o) The task awaits a non-empty state for the Decode Queue.
p) Data is removed from the Decode Queue and decoded using the selected IAC codec.
q) Decoded data is combined with previously-decoded data (from all peers) for the same object.
r) If sufficient data has been received and decoded to reconstruct the object, processing continues at (s). Otherwise, processing repeats at (o).
s) The Decode Task sets the Stop Download Flag, thereby signaling the main task.
t) The Decode Task terminates or otherwise deactivates, awaiting re-creation or activation by the main task.

4.5.4 Peer Engine Operation (Sharing Mechanism)

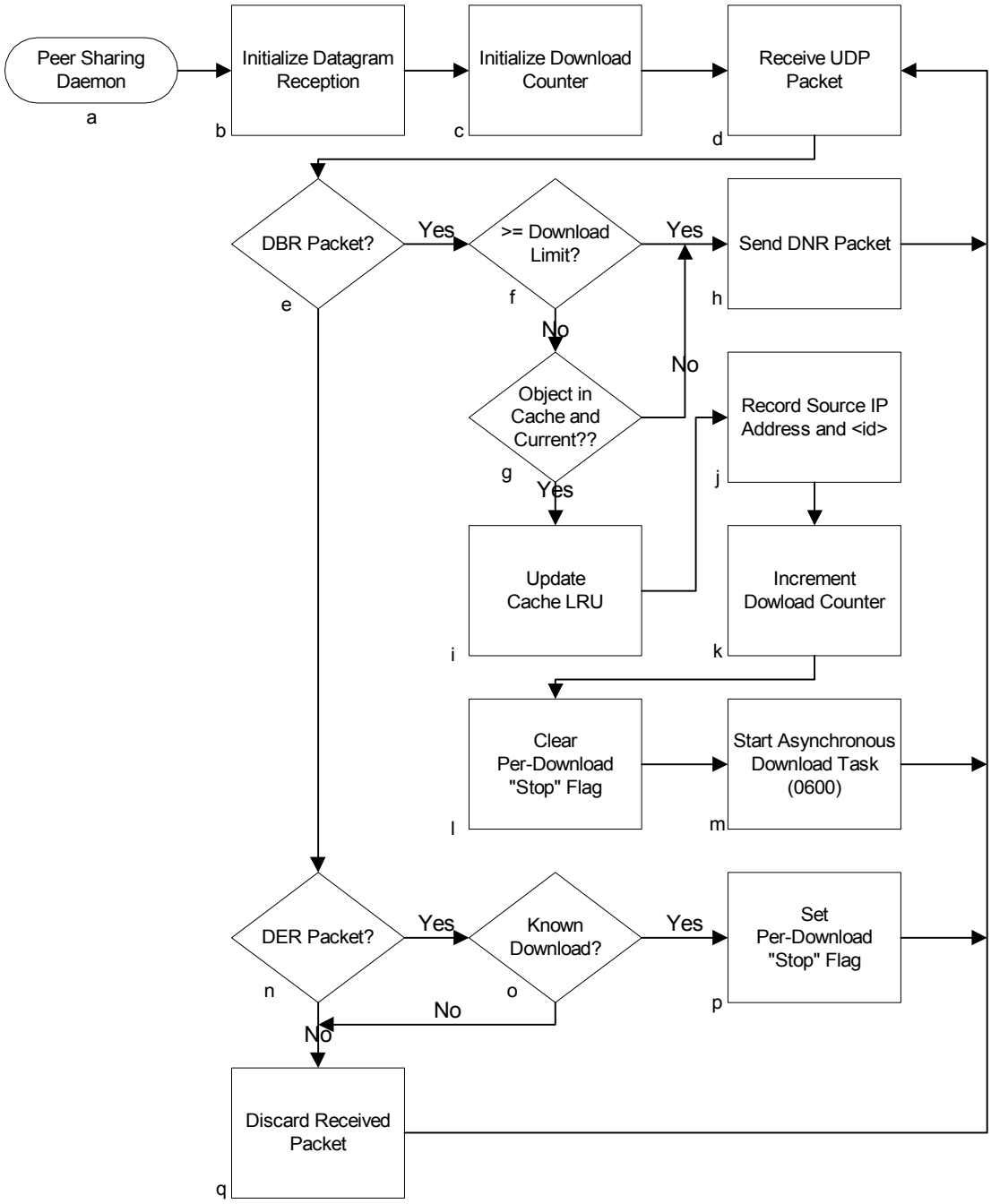

Figure 15: Peer Engine Operation (Sharing Mechanism)

Referring to Figure 15: Peer Engine Operation (Sharing Mechanism):

a) The peer sharing mechanism main task runs as a daemon, awaiting download requests from other peers.

b) A port, socket or other system-dependent resource is allocated in order that the daemon may receive UDP datagrams.
c) A counter of simultaneous downloads is initialized to zero.
d) The daemon awaits the reception of a UDP packet on the assigned port.
e) If the packet is a valid Download Begin Request (DBR, see page 28), processing continues at (f). Otherwise processing continues at (n).
f) (Valid DBR) Are the maximum number of simultaneous downloads already active? If so, processing continues at (h). Otherwise (another download is allowed) processing continues at (g). (The maximum number of sessions is a configurable value.)
g) Is there a copy of the requested object in the local Peer Object Cache (see page 25) and is it a current (non-expired) copy? If so, processing continues at (i). Otherwise, processing continues at (h).
h) Either the maximum number of permissible simultaneous downloads are already active, no copy of the requested object was found in the local Peer Object Cache, or the copy has expired. A Download Negative Response (DNR, see page 38) is sent and processing continues at (d).
i) (The download limit has not been reached, and a non-expired copy of the requested object has been found in the cache.) The cache LRU is updated to reflect this now most-recent request for the object.
j) The requester's IP address and the <id> field from the DBR packet associated with this request are saved. These will later be used to validate the end-download request.
k) The download counter is incremented. (A new download is about to be initiated.)
l) The per-download "Stop" flag is cleared. One instance of this flag is associated with each allowable simultaneous download.
m) An asynchronous download service task is started, initiated or otherwise begun. (See page 41.) The current task continues processing at (d).
n) (Not a valid DBR.) If the packet is a valid Download End Request (DER, page 30) the processing continues at (o). Otherwise processing continues at (q).
o) (Valid DER) Does the DER refer to a valid and currently-active download task? (This is determined be examining the <id> field in the DER packet and the requester's source IP address, and comparing these to those received at the time the downloads were initiated. See (j) above. If so, processing continues at (p). Otherwise it's an unknown download, and processing continues at (q).
p) (Known download) The "Stop" flag associated with the proper download task is set, and processing continues at (d).
q) (Unknown packet type or download) The received packet is discarded and processing continues at (d).

4.5.5 Download Service Task (0600)

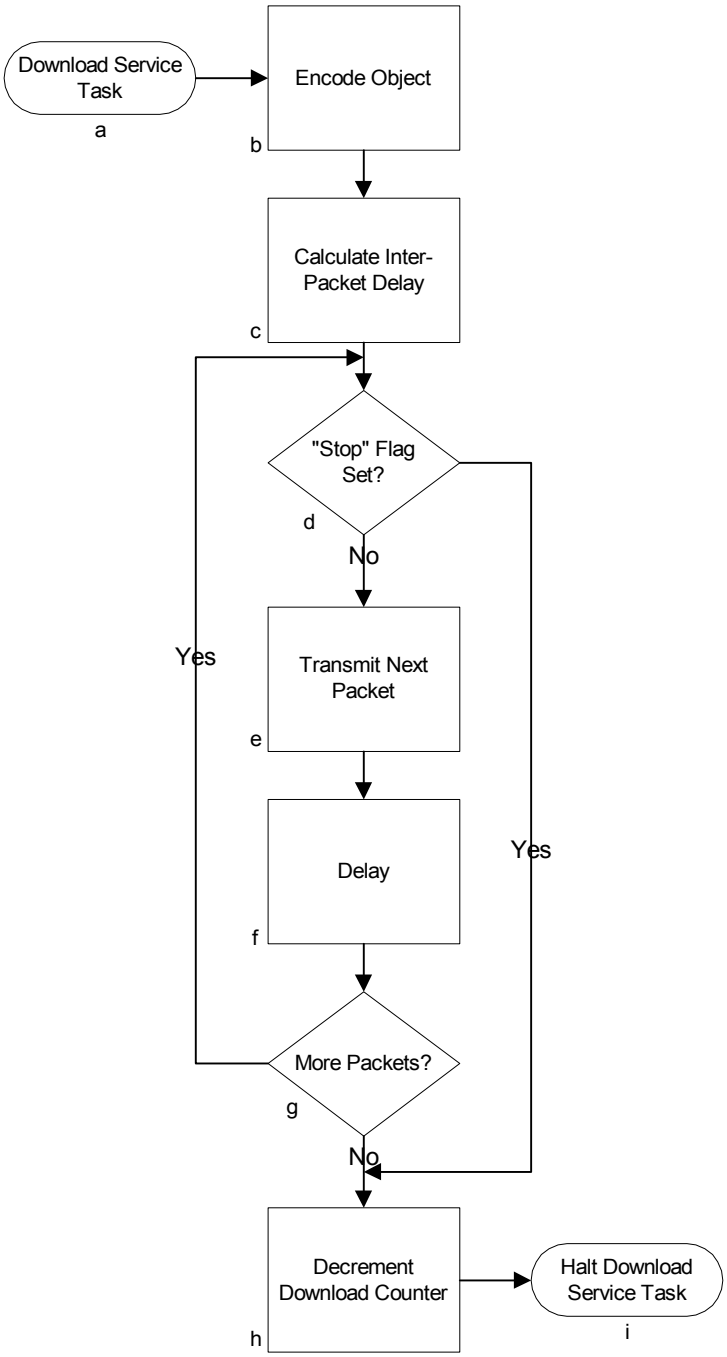

Figure 16: Download Service Task

Referring to Figure 16: Download Service Task:

a) This is a service task initiated by the Peer Engine Sharing Mechanism task. (See page 39.) There may be as many instances of this task as the maximum allowable downloads, a configurable value.
b) The requested object is retrieved from the local Peer Object Cache, and encoded using the selected IAC codec. (Note: PEs may store objects in encoded format, decoded format, or both.)
c) Based on the value of the <maxspeed> attribute in the associated Download Begin Request (DBR) packet, the inter-packet delay time is calculated. If no <maxspeed> was specified, a delay of zero may be used. Alternatively, if the implementer of the PE wishes to protect local resources from being monopolized by peering activities, an implementation-specific and potentially configurable minimum may be used.
d) If the download-specific "Stop" flag has been set by the main task, processing continues at (h). Otherwise processing continues at (e).
e) The next (or potentially first) packet is transmitted.
f) The task pauses for the previously-computed inter-packet delay, if any.
g) If more packets are required to complete transmission of the requested object, processing continues at (d). Otherwise processing continues at (h).
h) The global counter of simultaneous downloads is decremented.
i) This service task is terminated or halted, as appropriate for the implementation's operating system environment.

[Note: There is a possible race condition associated with the local Peer Object Cache. There is a period of time between the determination that an object is available (in the Sharing Mechanism main task) and the retrieval of that object from cache by this task. It is possible that the object may have expired or been deleted by the LRU logic. This should be solved via a semaphore that locks the object between the query and the use by this task. Alternatively, the decode operation (and subsequent unique copy) could be performed by the main task. …drk 9/20/01]

4.6 Server Engine Operation

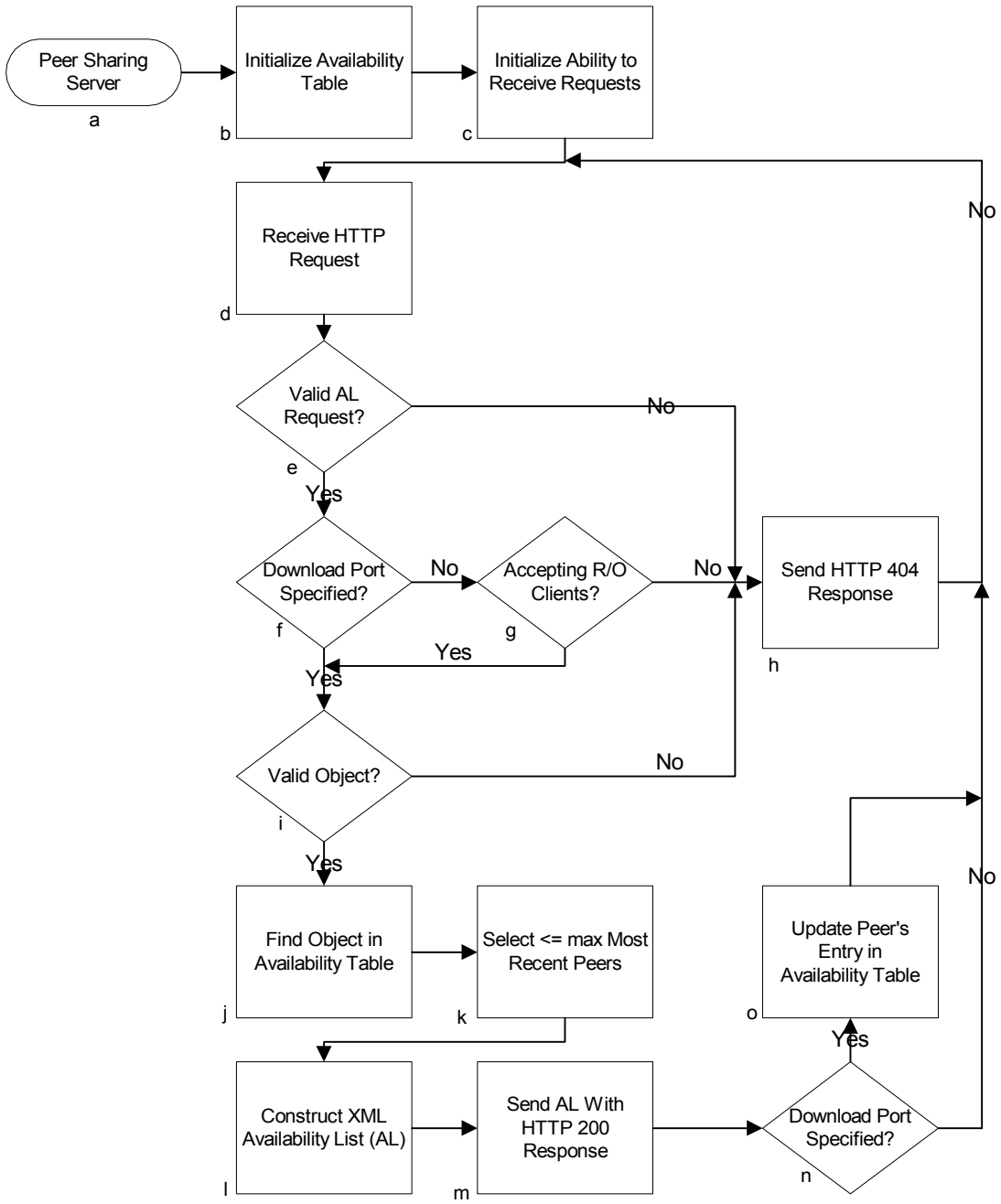

Figure 17: Server Engine Operation

Referring to Figure 17: Server Engine Operation:

a) The server engine (SE) software is far simpler than the peer engine (PE). The SE consists only of a daemon that awaits HTTP requests from peers, typically on the standard port (80).

b) An availability table (see page 25) is initialized to empty.
c) The daemon initializes the interface to the host system that will allow the daemon to receive AL Requests. Using the preliminary AL Request/Response Protocol (see page 27) this implies receiving control whenever the path /P2P-IAC-AL is referenced in an HTTP request URL.
d) The daemon awaits the receipt of a request.
e) If the HTTP request contains a valid AL Request (see page 27), processing continues at (f). Otherwise it continues at (h).
f) If a <download_port#> (see page 27) was specified in the request (implying that the client is able and willing to accept download requests from peers) processing continues at (i). Otherwise processing continues at (g).
g) (No <download_port#>) If the SE accepts read-only clients (a configuration option), processing continues at (i). Otherwise processing continues at (h).
h) (If the request is invalid, or it's an unacceptable request from read-only client, or a request for a non-existent object or one that cannot be delivered via P2P-IAC.) An HTTP response containing code 404—File Does Not Exist—is sent to the requesting client. Processing continues at (d).
i) If the request is for a valid object (one that exists and can be delivered via P2P-IAC), processing continues at (j). Otherwise processing continues at (h).
j) The daemon searches the Availability Table for peers that have also requested the currently-requested object.
k) The 0 through <max> number of peers that most-recently requested the object are selected. <max> is a configurable value.
l) An Availability List (AL, see page 26) containing the selected peers (if any) is constructed.
m) The AL is sent as an HTTP response with code 200—Success.
n) If a <download_port#> was specified in the request (see (f) above), processing continues at (o). Otherwise processing continues at (d).
o) (A <download_port#> was specified. The client is not a read-only peer.) The Availability Table is updated to reflect that the client requested the current object at the current time. Processing continues at (d).

Doug Kaye
doug@rds.com
29 January 2002