
Local DCOM value objects that are not primary DCOM value objects are not required to support this interface.

20.6.2 *ISynchronize and DISynchronize*

These interfaces are implemented on local DCOM value objects (**ISynchronize** is found on COM proxies, **DISynchronize** is found on Automation proxies). If the interface is available, it means that this is a local DCOM value object, not a regular object or a primary DCOM value object.

20.6.2.1 *Mode Property*

The property **Mode** is used to control when synchronization is done. A value of **kNeverSync** means that the local and the primary value objects are never synchronized.

A value of **kSyncOnSend** means that, if the local value object has been changed, the primary value object will be synchronized with the local value object when the local value object is sent to another client/server, which cannot be reliably determined to support the required DCOM value object. Implementations can choose to synchronize using either batch synchronization through a call to **IValueObject**, or through calls for each changed member through the regular remote interface.

A value of **kSyncOnChange** means that, as a member is changed, the update of the member should be propagated to the primary value object as a regular remote call.

20.6.2.2 *SyncNow Method*

The **SyncNow** method can be called by application code to force the changes to the local value object to be propagated to the primary value object. Implementations can choose to synchronize using either batch synchronization through a call to **IValueObject**, or through calls for each changed member through the regular remote interface.

20.6.2.3 *ReCopy Method*

The **ReCopy** method can be called by application code to retrieve the current value of the primary value object and update the local value object.

20.7 *DCOM Value Objects*

20.7.1 *Passing Automation Compound Types as DCOM Value Objects*

Compound types such as structures and unions are encapsulated in Automation classes so they may be used by Automation applications. These are DCOM value objects. When a DCOM value object representing a compound type is passed to a remote

client, its interface pointer is marshaled using standard marshaling (as with any DCOM value object), and its value data is forwarded simultaneously using the extent described in Section 20.4, “Extent Definition,” on page 20-5.

20.7.2 *Passing CORBA-Defined Pseudo-Objects as DCOM Value Objects*

To handle the DCOM views of CORBA pseudo objects as DCOM value objects, the memory representation of these data types must be defined. The following sections detail the value data that will be passed in the extent.

20.7.3 *IForeignObject*

Supporting **IForeignObject**’s as a DCOM value object is required to avoid proxy explosion. The marshaled data for value objects of type **IForeignObject** is described in Section 20.8.2, “COM Chain Avoidance,” on page 20-17.

20.7.4 *DIForeignComplexType*

The value data for DCOM value objects of type **DIForeignComplexType** can be represented by the following structure (note that this also includes the state for **DIOBJECTINFO**):

```
struct FOREIGN_COMPLEX
{
    LPSTR      name;           // Name of type
    LPSTR      scopedName;    // Scoped name (if available)
    LPSTR      repositoryId;  // Repository ID of type
};
```

20.7.5 *DIForeignException*

The value data for DCOM value objects of type **DIForeignException** can be represented by the following structure:

```
struct FOREIGN_EXCEPTION
{
    FOREIGN_COMPLEX base;
    long            majorCode;
};
```

20.7.6 *DISystemException*

The value data for DCOM value objects of type **DISystemException** can be represented by the following structure:

```
struct CORBA_SYSTEM_EXCEPTION
{
    FOREIGN_EXCEPTION base;
};
```

```

        long          minorCode;
        long          completionStatus;
    };

```

20.7.7 *DICORBAUserException*

The value data for **DICORBAUserException** is identical to that of **DIForeignException**. Value objects deriving from **DICORBAUserException** are passed as DCOM value objects according to the previously defined format. The value data of exception members must be preceded by the value data of **DIForeignException**.

20.7.8 *DICORBAStruct*

The value data for **DICORBAStruct** is identical to that of **DIForeignComplexType**. Value objects deriving from **DICORBAStruct** are passed as DCOM value objects according to the previously defined format. The value data of struct members must be preceded by the value data of **DIForeignComplexType**.

20.7.9 *DICORBAUnion*

The value data for **DICORBAUnion** is identical to that of **DIForeignComplexType**. Value objects deriving from **DICORBAUnion** are passed as DCOM value objects according to the previously defined format. The value data of a union must be preceded by the value data of **DIForeignComplexType**. The value data for the union itself is the discriminant followed by the selected member, if any.

20.7.10 *DICORBATypeCode and ICORBATypeCode*

The value data for type code DCOM value objects can be represented by the following struct:

```

struct CORBA_TYPECODE
{
    FOREIGN_COMPLEX    base;
    TCKind             kind; // TypeCode kind

    union TypeSpecific switch(kind)
    {
        case tk_objref:
            LPSTR      id;
            LPSTR      name;

        case tk_struct:
        case tk_except:
            LPSTR      id;
    }
}

```

```

        LPSTR        name;
        long         member_count;
        [size_is(member_count,)] LPSTR    *member_names;
        [size_is(member_count,)] IUnknown**member_types;

    case tk_union:
        LPSTR        id;
        LPSTR        name;
        long         member_count;
        LPSTR        member_names[];
        [size_is(member_count,)] IUnknown**member_types;
        [size_is(member_count)] VARIANT *member_labels;
        IUnknown     *discriminator_type;
        long         default_index;

    case tk_enum:
        long         member_count;
        [size_is(member_count,)] LPSTR    *member_names;
        [size_is(member_count,)] IUnknown**member_types;

    case tk_string:
        long         length;

    case tk_array:
    case tk_sequence:
        long         length;
        IUnknown     *content_type;

    case tk_alias:
        LPSTR        id;
        LPSTR        name;
        long         length;
        IUnknown     *content_type;
    }
};

```

Note that members of type `IUnknown` will actually be `ICORBATypeCode` instances for COM and `DICORBATypeCode` instances for Automation.

20.7.11 DICORBAAny

The value data for DCOM value objects of type `DICORBAAny` can be represented by the following structure:

```

struct CORBA_ANY_AUTO
{
    FOREIGN_COMPLEXbase;
    VARIANT        value;
    DICORBATypeCode*typeCode;
};

```

20.7.12 ICORBAAny

The value data for DCOM value objects of type **ICORBAAny** can be represented by a **CORBAAnyDataUnion** as defined in COM/CORBA Part A.

20.7.13 User Exceptions In COM

In COM, all CORBA user exceptions used in an interface are represented by another interface, which contains one method per user exception. The value data for one of these exception interfaces is an encapsulated DCOM union where each member of the union is one of the exception definition structures. The discriminant values are the indices of the corresponding structure retrieval method from the user exception interface.

```

module Bank
{
    ...
    exception InsufFunds { float balance; };
    exception InvalidAmount { float amount; };

    interface Account
    {
        exception NotAuthorized {};

        float Deposit(in float amount)
            raises(InvalidAmount);

        float Withdraw(in float amount)
            raises(InvalidAmount, NotAuthorized);
    };
};

```

Per the COM/CORBA Part A specification, the above IDL results in the following interface used for user exceptions:

```

struct Bank_InsufFunds { float balance; };
struct Bank_InvalidAmount { float amount; };
struct Bank_Account_NotAuthorized {};

interface IBank_AccountUserExceptions : IUnknown
{
    HRESULT get_InsufFunds([out] Bank_InsufFunds *);
    HRESULT get_InvalidAmount([out] Bank_InvalidAmount *);
    HRESULT get_NotAuthorized([out] Bank_Account_NotAuthorized *);
};

```

When this DCOM value object is passed, the value data is marshaled as the following data structure:

```

union Bank_AccountUserExceptionsData switch(unsigned short)
{
    case 0: Bank_InsuffFunds           m0;
    case 1: Bank_InvalidAmount        m1;
    case 2: Bank_Account_NotAuthorized m2;
};

```

20.8 Chain Avoidance

To avoid view chaining (and thus proxy explosion), we define a general mechanism to carry chain information along with object references. This mechanism is defined in both COM and in CORBA to allow for bidirectional chain avoidance. Views in either system carry this information along with their object references. For example, the information carried in the object reference to a CORBA view of an Automation object would describe the object referred to by the view; that is, information about the Automation object.

20.8.1 CORBA Chain Avoidance

In CORBA, the chain avoidance information is carried as an IOP profile in an object reference that is part of a chain.

```

module CosBridging
{
    typedef sequence<octet> OpaqueRef;
    typedef sequence<octet> OpaqueData;
    typedef unsigned long  ObjectSystemID;

    interface Resolver
    {
        OpaqueRef Resolve(in ObjectSystemID objSysID,
                        in unsigned long  chainDataFormat,
                        in octet         chainDataVersion,
                        in OpaqueData     chainData);
    };

    struct ResolvableRef
    {
        Resolver      resolver;
        ObjectSystemID objSysID;
        unsigned long  chainDataFormat;
        octet         chainDataVersion;
        OpaqueData     chainData;
    };

    typedef sequence<ResolvableRef> ResolvableChain;

    struct BridgingProfile
    {

```

```

        ResolvableChainchain;
    };
};

```

The content of the profile is defined as a single **BridgingProfile** structure. The ID for this profile will be allocated by the OMG. The profile structure contains a sequence of **ResolvableRef** structures, potentially one for each object system in the chain.

The **ResolvableRef** structure contains a **resolver** CORBA object reference that can be called at runtime through its **Resolve** method to return an opaque (because it is not CORBA) object reference for the specified link in the chain. The link in the chain is identified by the object system ID, **objSysID**.

Currently defined object system IDs are: 1 for CORBA, 2 for Automation, and 3 for COM. IDs in the range from 0 through 100000 are reserved for use by the OMG for future standardization.

The **ResolvableRef** structure also contains information that can be used by the **resolver** as context to find the appropriate information to return. While this chain data is opaque, it is also tagged with a format identifier so that bridges that understand the format can directly interpret the contents of **chainData** instead of making a remote call to **Resolve**. The only currently defined format tag is 0, which is currently defined as "private." That is, **chainData** tagged as private cannot be directly interpreted and must be passed to the resolver for interpretation. All other format tags are specific to each object system. Format tags in the range of 1 to 100000 are reserved for allocation by the OMG.

The result of calling the **Resolve** method on a COM or Automation **ResolvableRef** is an NDR marshaled DCOM object reference with at least one strong reference.

20.8.2 COM Chain Avoidance

A similar approach is adopted to resolve the same chain avoidance issues in COM; however, since DCOM does not support profiles, the implementation is different. The information for chain avoidance (also used by **IForeignObject** and **IForeignObject2**) is provided as DCOM value data associated with each passed view object. This information is represented by a **ResolvableRefChain**.

```

struct OpaqueRef
{
    unsigned long          len;
    unsigned long          maxlen;
    BYTE [size_is(len)]   *data;
};

struct OpaqueData
{
    unsigned long          len;
    unsigned long          maxlen;
};

```

```

        BYTE [size_is(len)]      *data;
};

typedef unsigned long ObjectSystemID;

struct ResolvableRef
{
    IResolver                resolver;
    ObjectSystemID           objSysID;
    unsigned long            chainDataFormat;
    BYTE                     chainDataVersion;
    OpaqueData               chainData;
};

struct ResolvableRefChain
{
    unsigned long            len;
    unsigned long            maxlen;
    ResolvableRef [size_is(len,)]**data;
};

[
    object,
    pointer_default(unique),
    uuid(5473e440-20ac-11d1-8a22-006097cc044d)
]
interface IResolver : IUnknown
{
    OpaqueRef Resolve([in] ObjectSystemID objSysID,
                     [in] unsigned long chainDataFormat,
                     [in] BYTE chainDataVersion,
                     [in] OpaqueData chainData);
};

[
    object,
    pointer_default(unique),
    uuid(60674760-20ac-11d1-8a22-006097cc044d)
]
interface IForeignObject2 : IForeignObject
{
    ResolvedRefChain ChainInfo();
};

```

The use semantics of the resolver is identical to the use semantics described in Section 20.8.1, "CORBA Chain Avoidance," on page 20-16. One format tag with value 1 is defined for a **ResolvableRef** with **objSysID** 1 (CORBA). If the format tag is 1, the **chainDataVersion** must be 0 and the **chainData** contains a (CDR marshaled) byte defining the byte ordering for the rest of the **chainData** (the byte value is identical to that used to encode GIOP messages) followed by a CDR

marshaled object reference. If **Resolve** were called for this **ResolvableRef**, the same value as contained in the **chainData** would be returned by **Resolve** (i.e., a CDR-marshaled object reference).

In addition to this mechanism, the interface **IForeignObject2** is defined on COM or Automation views to return the **ResolvableRefChain** in cases where this information has been lost.

20.9 Chain Bypass

Using the chain avoidance technique defined in this specification, the formation of view chains can be avoided. However, there are cases where the chain avoidance information carried with the object references may have been discarded (for instance, as the object reference is passed through a firewall). In this case, chaining of views cannot be avoided without an explicit performance hit, which was deemed unacceptable. However, at the point when the first invocation is performed, information about the current chain can be returned as out-of-band data. This information can then be used on subsequent invocations to bypass as many views as possible in order to avoid the performance hit of multiple view translations.

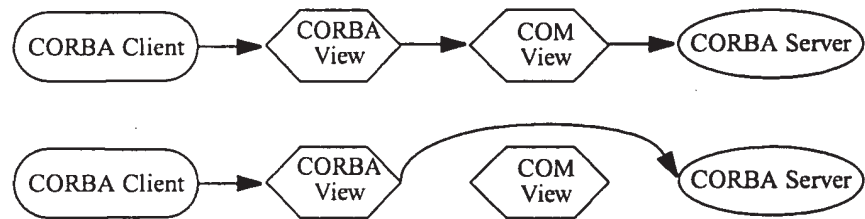


Figure 20-3 Invocation With and Without Chain Bypass

Figure 20-3 shows an example of a call that does not perform chain bypass followed by one that does. Note that chain bypass cannot eliminate all unnecessary calls since the client already has a reference to the view (not to the original object) and thus invokes an operation on the view. It is the responsibility of the view to perform the chain bypass if it so chooses -- in this case the view essentially becomes a rebindable stub.

20.9.1 CORBA Chain Bypass

For views to discover the chain information, two service contexts are defined as follows:

```

ChainBypassCheck = 2
ChainBypassInfo = 3

module CosBridging
{
    struct ResolvedRef
  
```

```

    {
        Resolver      resolver;
        ObjectSystemID objSysID;
        unsigned long chainDataFormat;
        octet         chainDataVersion;
        OpaqueData    chainData;
        OpaqueRef     reference;
    };

    typedef sequence<ResolvedRef> ResolvedRefChain;

    struct ChainBypassCheck // Outgoing service context
    {
        Object    objectToCheck;
    };

    struct ChainBypassInfo // Reply service context
    {
        ResolvedRefChain chain;
    };
};

```

The **ChainBypassCheck** service context is sent out with the first outgoing (non-oneway) request. Since the service context is propagated automatically to subsequent calls, an object is provided in the service context to avoid returning chain information for an incorrect object. For a reply service context to be generated, the object in the service context must match the object (a view) being invoked.

If a reply service context, **ChainBypassInfo**, is received with the reply message, then a view has been detected. The information in the **ResolvedRefChain** can be used to bypass intermediate views. Each **ResolvedRef** is identical to a **ResolvableRef** except that it also contains the result of the resolution -- the **reference** member contains the data that would be returned if **Resolve** were called on the included resolver. If the reference field of **ResolvedRef** is an empty sequence, then the marshaled object reference is assumed to be identical to the **chainData**.

20.9.2 COM Chain Bypass

The technique used for COM chain bypass is very similar to the technique used in CORBA. The only difference is the result of the fact that DCOM extents are not propagated into subsequent calls unlike CORBA service contexts.

```

struct ResolvedRef
{
    IResolver      resolver;
    ObjectSystemID objSysID;
    unsigned long  chainDataFormat;
    BYTE          chainDataVersion;
    OpaqueData    chainData;
};

```

```

        OpaqueRef          reference;
};

struct ResolvableRefChain
{
    unsigned long          len;
    unsigned long          maxlen;
    ResolvableRef [size_is(len,)]**data;
};

struct ChainBypassCheck    // Outgoing extent body
{
};

struct ChainBypassInfo    // Reply extent body
{
    ResolvableRefChain      chain;
};

```

The **ChainBypassCheck** extent is sent out with the first outgoing request. If a reply extent, **ChainBypassInfo**, is received with the reply message, then a view has been detected. The information in the **ResolvedRefChain** can be used to bypass intermediate views. Each **ResolvedRef** is identical to a **ResolvableRef** except that it also contains the result of the resolution -- the **reference** member contains the data that would be returned if **Resolve** were called on the included resolver. If the reference field of **ResolvedRef** is an empty sequence, then the marshaled object reference is assumed to be identical to the **chainData**.

The UUID for the request and reply extents are both:

```
1eba96a0-20b1-11d1-8a22-006097cc044d
```

20.10 Thread Identification

To correlate incoming requests with previous outgoing requests, DCOM requires a *causality ID*. The identifier is essentially a logical thread ID used to determine whether an incoming request is from an existing logical thread or is a different logical thread of execution. CORBA, on the other hand, does not strictly require a logical thread ID. To maintain the logical thread ID as requests pass through both DCOM and CORBA, we define a general purpose service context, which can be used to maintain logical thread identifiers for any system a thread of execution passes through.

```

module CosBridging
{
    struct OneThreadID
    {
        ObjectSystemID objSysID;
        OpaqueData      threadID;
    };
};

```

```

};

typedef sequence<OneThreadID> ThreadIDs;

struct LogicalThreadID // Service context
{
    ThreadIDs    IDs;
};
};

```

The logical thread ID information is propagated through a CORBA call chain using a service context (IDs to be assigned by the OMG) containing the **LogicalThreadID** structure.

For future use, a DCOM extent is defined to allow the same logical thread identification information to be passed through a DCOM call chain. If the OMG chooses to standardize a logical thread ID format for CORBA, this can be passed through a DCOM call chain using this extent.

```

struct OneThreadID
{
    ObjectSystemID objSysID;
    OpaqueData     threadID;
};

struct ThreadIDs
{
    unsigned long    len;
    unsigned long    maxlen;
    OneThreadID [size_is(len)] *data;
};

struct LogicalThreadID // DCOM extent
{
    ThreadIDs    IDs;
};

```

This extent, used for passing logical thread IDs, is identified by the following UUID:

```
f81f4e20-2234-11d1-8a22-006097cc044d
```

The Random Walk For Dummies

RICHARD A. MONTE

Abstract. We look at the principles governing the one-dimensional discrete random walk. First we review five basic concepts of probability theory. Then we consider the Bernoulli process and the Catalan numbers in greater depth. Finally we determine the probability that, if a drunk is found again at the bar, then this is his first return visit.

1. Introduction. The random walk has been a topic of interest in many disciplines, but it has been of particular interest in probability theory. Indeed, every student of probability theory has heard of the random walk, especially in the form of a drunk leaving a bar and wandering aimlessly up and down the street.

Among other issues, the following four have been investigated in one, two, and three dimensions:

- The expected position x of the drunk after n steps.
- The maximum position x that the drunk has reached after n steps.
- The expected time of the drunk's last visit to 0.
- The probability that the drunk hasn't stumbled upon his own path after n steps.

These issues are discussed, for example, in Rota's book on probability [3].

The one-dimensional discrete case is most widely known because it is simple, yet illustrates many interesting and important features. In this paper, we treat this case of the first issue above. Thus we answer a basic question: What is the probability that the drunk is at a certain distance x , from the bar, after n steps?

In Section 2, we review five basic topics: the sample space, the binomial coefficient, random variables, the Bernoulli process, and Catalan numbers. In Section 3, we find the probability distribution for the position of the drunk after n steps. In Section 4, we proceed to calculate the probability that the drunk arrives at the bar for the first time after n steps. Finally, in Section 5, we determine the conditional probability that the drunk's first return occurs on the n th step given that he is indeed at the bar then; the formula is surprisingly simple.

2. The Basics. In this section, we review five concepts of probability theory, which we will use to study the random walk of the drunk.

First, a *sample space* Ω is defined to be the set of all possible outcomes of an experiment. Consider the coin toss as an example. A single toss has two possible outcomes: heads H , or tails T ; thus $\Omega = \{H, T\}$. If we toss the coin twice, then there are 2^2 different possibilities; now $\Omega = \{TT, TH, HT, HH\}$.

Second, the *binomial coefficient* $\binom{n}{k}$ is defined to be the number of k -combinations of an n -element set. In other words, it is the number of different ways to pick k elements out of n . The binomial coefficient is given by the formula,

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

This formula is proved in [1, p. 61].

Third, a *random variable* is defined to be a function X that assigns to each element c , in the sample space of an experiment, one and only one real number $X(c)$. The sample space Ω of X is the set of real numbers x such that $x = X(c)$ for some c in the sample space of the experiment. This definition is found in [2, p. 28].

Fourth, a *Bernoulli process* is defined to be a sequence of random variables X_1, X_2, X_3, \dots . Each X_i records the outcome of an experiment modeled by the toss of a coin. Let p equal the probability of getting a head, and q the probability of getting a tail. Since these are the only possible outcomes, $p + q = 1$. Let X_i be 1 if the i th trial yields a head, and be -1 if a tail. Thus, the sample space Ω of a Bernoulli process is the set of all possible sequences of 1 and -1 .

Fifth and finally, the n th *Catalan number* c_n counts certain arrangements of $2n$ parentheses. A *well-formed arrangement* is a list of $2n$ parentheses where each open parenthesis can be paired with a corresponding closed parenthesis to its right. The n th Catalan number is the number of such well-formed arrangements. For example, when $n = 3$, the possible well-formed arrangements are

$$()()(), ()(()), (())(), ((())), ((())).$$

According to [1, p. 253], the n th Catalan number c_n is given by the formula,

$$c_n = \frac{1}{n+1} \binom{2n}{n}.$$

Thus, for example, the first six Catalan numbers are 1, 1, 2, 5, 14, and 42.

3. How It All Ties In. Suppose a drunk leaves a bar and walks aimlessly up and down the street, totally disoriented. We model the street as a line with the bar at the origin, and assume that the drunk takes unit steps, so we may record his position with an integer. Thus, for example, if he takes 5 steps to the left, he will be at position -5 .

We now calculate the probability that the drunk is at position x after n steps. Assume that the drunk's walk can be modeled by a Bernoulli process. Say that the i th step is represented by the random variable X_i . A value of -1 indicates a step to the left; a value of 1 , a step to the right, so that

$$G_n = X_1 + X_2 + \dots + X_n$$

gives the position of the drunk after n steps. We want to know the distribution of the random variable G_n .

Denote the value of G_n by x . Denote the number of steps taken to the right by r , the number to the left by l . Then

$$x = r - l \text{ and } n = r + l.$$

It follows that

$$r = \frac{1}{2}(x + n) \text{ and } l = \frac{1}{2}(n - x).$$

Now, there are $\binom{n}{l}$ ways that l given steps can occur among n total steps. This is also the number of ways of arriving at the point x , and each way has probability $p^r q^l$. Note that n and x must have the same parity because $n - x = 2l$. We can therefore conclude that the probability distribution at point x is given by the formula,

$$P(G_n = x) = \begin{cases} \binom{n}{l} p^r q^l, & \text{if } x = n \bmod 2; \\ 0, & \text{otherwise.} \end{cases}$$

Consider for example the case $p = q = \frac{1}{2}$; this is the case of the *symmetric random walk*. The probability $P(G_n = x)$ of being at position x after n steps is given in Table 3-1. This table is simply a pascal triangle interspersed with 0s.

Table 3-1
The Symmetric Random Walk

$n \setminus x$	-5	-4	-3	-2	-1	0	1	2	3	4	5
0						1					
1					$\frac{1}{2}$	0	$\frac{1}{2}$				
2				$\frac{1}{4}$	0	$\frac{2}{4}$	0	$\frac{1}{4}$			
3			$\frac{1}{8}$	0	$\frac{3}{8}$	0	$\frac{3}{8}$	0	$\frac{1}{8}$		
4		$\frac{1}{16}$	0	$\frac{4}{16}$	0	$\frac{6}{16}$	0	$\frac{4}{16}$	0	$\frac{1}{16}$	
5	$\frac{1}{32}$	0	$\frac{5}{32}$	0	$\frac{10}{32}$	0	$\frac{10}{32}$	0	$\frac{5}{32}$	0	$\frac{1}{32}$

The probability distribution in this particular case is given by

$$P(G_n = x) = \binom{n}{l} \left(\frac{1}{2}\right)^n = \binom{n}{l} / 2^n.$$

Since every path is equally likely in the symmetric random walk, the probability can be interpreted purely combinatorially. The probability is the number of different ways of arriving at x divided by the size of the sample space. The sample space is the set of all the possible paths of length n . Since the drunk has two choices at each point, and he takes a total of n steps, the total number of possibilities is 2^n .

4. Taking a Step Further. From now on, we assume that the random walk is symmetric; that is, $p = \frac{1}{2}$. What is the probability that the drunk's first return is at the $2n$ th step? Here is where Catalan numbers enter. Observe that there is a one-to-one correspondence between paths and arrangements of parentheses. First, let an open parenthesis represent a step to the left, and a closed parenthesis a step to the right. For now, we consider only the case where the drunk's first step is towards the left, since the case to the right is clearly symmetric to it.

From Section 2, recall the definition of the Catalan number c_n : it is the number of $2n$ well-formed arrangements of parentheses. Note that, for well-formed arrangements, the number of open parentheses is always at least that of closed parentheses, regardless of what first k parenthesis we pick. Because of the correspondence between paths and arrangements, the Catalan numbers count the number of paths that the drunk can take, which start and end at the bar, without ever crossing to the right side of the bar.

We now adapt our correspondence to the problem of the drunk's first return. The drunk takes his first step to the left. On the next $2n - 2$ steps, we insist that the drunk's path corresponds to a well-formed arrangement so that at step $2n - 1$ he is at position

-1 without ever crossing to the right of -1 . Then the final step brings the drunk back into the bar. This counts the total number of different paths that the drunk can take given the condition that the first return be at the $2n$ th step.

Summarizing, we have the following sequence of steps:

- **Step 1.** The drunk takes one step to the left.
- **The next $2n-2$ steps.** The drunk follows a path, which corresponds to a $2n-2$ well-formed arrangement, that restricts the drunk to be either at point -1 or to the left of it, and that forces him to be at -1 after the $(2n-1)$ st step.
- **Step $2n$.** The drunk's first return visit occurs when he takes one step to the right and into the bar.

Hence, the number of paths the drunk can take is simply the Catalan number c_{n-1} , since the well-ordered restriction applies only to the middle $2n-2$ steps.

We know that the n th Catalan number is equal to

$$\frac{1}{n+1} \binom{2n}{n}.$$

So the $(n-1)$ st Catalan number is simply

$$\frac{1}{n} \binom{2n-2}{n-1}.$$

Allowing the drunk to take his first step to the right will double the total number of paths that the drunk can take. So the total is

$$2 \frac{1}{n} \binom{2n-2}{n-1}.$$

To compute the probability of first return, we need only divide this quantity by the size of the total sample space. The latter was seen at the end of Section 3 to be 2^{2n} since there are $2n$ steps. Therefore, the probability that the drunk reaches the bar for the first time after $2n$ steps is

$$2 \frac{1}{n} \binom{2n-2}{n-1} / 2^{2n}.$$

So, for example, consider the probability that the first return is at Step $2n = 6$; this probability is given by

$$2 \frac{1}{3} \binom{4}{2} / 2^6 = \frac{1}{16}.$$

5. Conditional Probability and the First Return. We now consider a variant of the problem studied in the previous section: Given that the drunk is at the bar at Step $2n$, what is the probability that this is his first return visit? This problem is like the previous one. The only difference is that the sample space has been reduced. Instead of considering all of the possible paths, we now consider the total number of paths given that, in the end, the drunk will be at the bar, which he may or may not have passed by earlier.

To find the probability that the drunk will be at the bar after $2n$ steps, we use the conclusions from Section 3 to obtain

$$P(G_{2n} = 0) = \binom{2n}{n} p^{\frac{1}{2}(2n)} q^{\frac{1}{2}(2n)}.$$

Since $p = q = \frac{1}{2}$, this expression becomes

$$\binom{2n}{n} / 2^{2n}.$$

The numerator in this expression represents the number of paths the drunk can take provided he is at the bar at the $2n$ th step. This is sample space we need for the conditional probability.

From Section 4, recall that, if the drunk's first return visit is on the $2n$ th step, then the number of paths that the drunk can take is

$$2 \frac{1}{n} \binom{2n-2}{n-1}.$$

On the other hand, in Section 3, we found the number of paths that the drunk can take that put him back at the bar on Step $2n$; this number is simply

$$\binom{2n}{n}.$$

Finally we divide these two numbers, getting

$$2 \frac{1}{n} \binom{2n-2}{n-1} / \binom{2n}{n} = \frac{2}{n} \frac{(2n-2)!}{(n-1)!(n-1)!} / \frac{(2n)!}{n!n!},$$

which simplifies to

$$\frac{2}{n} / \frac{2n(2n-1)}{n^2} = \frac{2n}{2n(2n-1)} = \frac{1}{2n-1}.$$

The final simplicity is amazing! The formula is just

$$\frac{1}{\#(\text{steps}) - 1},$$

and is unexpectedly simple!

REFERENCES

- [1] Brualdi, R., "Introductory Combinatorics," Prentice-Hall, Third Edition, 1999.
- [2] Hogg, Robert V., "Introduction to Mathematical Statistics," Fifth Edition, 1995.
- [3] Rota, G. C., "Probability Theory," Preliminary Edition, 1998.

This page will be blank.



Published on **The O'Reilly Network** (<http://www.oreillynet.com/>)
<http://www.oreillynet.com/pub/a/linux/2000/09/22/p2psummit.html>
See this if you're having trouble printing code examples

Peer-to-Peer Makes the Internet Interesting Again

by Andy Oram
09/22/2000

By the summer of 2000, it looked like the Internet had fallen into predictable patterns. Retail outlets had turned the Web into the newest mail order channel, while entertainment firms used it to rally fans. Portals and search engines presented a small slice of Internet offerings in the desperate struggle to win eyes for banner ads. The average user, stuck behind a firewall at work or burdened with usage restrictions on a home connection, settled down to sending e-mail and passive viewing.

In a word, boredom. Nothing much to write about, or for creative souls to look forward to. An Olympic sports ceremony that would go on forever.

At that moment a number of shocks exploded upon the public. The technologies were not precisely new, but a number of people realized for the first time that they were having a wide social impact. First, Napster (preceded by Scour) caused a ruckus over its demands on campus bandwidth, as well as its famous legal problems. A new generation of distributed file servers like Freenet and Gnutella were widely reported. The founders of a new chat service called Jabber declared that XML was more than a tool for B2B transaction processing, and in fact could structure information chosen by ordinary users. Shortly after that, Microsoft announced they were betting the house on the .NET initiative, in which Web clients and servers divide jobs among themselves.


Analysts trying to find the source of inspiration among these developments also noted that the computing model of the SETI@Home project, which exploited processing power that had gone to waste for years, was being commercialized by a number of ventures. And that a new world of sporadically connected Internet nodes was emerging in laptops, handhelds, and cell phones, with more such nodes promised for the future in the form of household devices.

What thread winds itself around all these developments? In various ways they return content, choice, and control to ordinary users. Tiny end points on the Internet, sometimes without even knowing each other, exchange information and form communities. There are no more clients and servers -- or at least, the servers retract themselves discreetly. Instead, the significant communication takes place between cooperating peers. And thus, starting around early July 2000, the new Internet model was dubbed peer-to-peer.

Mapping the terrain at the summit

In August, Tim O'Reilly surmised that peer-to-peer technology could evolve faster if key leaders, each of whom "had a hand on a piece of the elephant," started talking intensively to each other. Furthermore, he hoped they could emerge from a summit with a message to help the public

Related Articles:

[Open Source Roundtable: O'Reilly's Peer-to-Peer Summit](#) 

[Peer-to-Peer Summit Attendees](#)

[P2P news from Meerkat](#)

understand the potential of peer-to-peer. In particular, he wanted to counteract the negative image often attached to the movement and its most visible expressions, such as Napster and Gnutella.

Organized by numerous departments across O'Reilly & Associates, the summit on September 18 in San Francisco was attended by some 20 people whose expertise ranged across the computer field. (See list in accompanying Web page.) They included technologists from major companies, visionary heads of startups, journalists, leaders of academic and experimental projects, and people who have just done a variety of interesting things in the information processing field and are always looking for the next interesting thing.

The meeting took place in a modest conference room in the Lone Mountain Center of the University of San Francisco, located atop a hill that offered gorgeous views in every direction. The day went fast, even though most participants looked tired at the end. Several expressed pleasure at being in the same room with such an aggressively innovative bunch of people. Our agenda broke down pretty much as follows:

- Define peer-to-peer. Decide what we want from it and why we're in it.
- Describe the opportunities for the use of peer-to-peer. For what problems is it a good candidate? What key memes characterize it?
- Come up with a message to offer the public.

The energy in the day rose and fell in a kind of bell curve. At first we were getting to know each other and setting the groundwork for discussion. It was notable how often someone would state what he felt to be a defining characteristic of peer-to-peer (such as "it can't do searches for poorly understood items") only to be vigorously contradicted by someone from another project. Being geeks to some extent, the group could become highly animated over minor technical points, such as whether instant messaging passes text through a central server (the answer is sometimes) where a snoop could theoretically monitor content. We also got incensed over the threat of software patents.

But in the descending rays of afternoon sun, when we tried to extract a simple set of principles from our experience, we found out how new and unformed the field is. Simple generalities couldn't hold up to dispassionate observation. At the end of the day, literally, we had to be content with listing the early successes of peer-to-peer and suggesting a vision that many of us are fashioning.

Still, some themes emerge from a number of peer-to-peer projects.

Making galaxies out of dark matter

Clay Shirky, a partner at The Accelerator Group and a writer for Business 2.0, who came up with many of the catchy epigrams of the day, pointed out that peer-to-peer exploits the "dark matter of the Internet." Just as interstellar space contains huge amounts of inert dust that we can't track or

"With peer-to-peer the computer is no longer just a life-support system for your browser."

measure -- but that may be forming galaxies all the time -- personal computers offer enormous amounts of cycles consumed by busy waits, along with disk space waiting to be filled and communication lines that lie idle in between keystrokes. Peer-to-peer creates constellations of cooperating users out of these wasted resources. As explained by Dave Stutz, a software architect at Microsoft, these three key resources -- CPU power, storage, and communications capacity -- are the prerequisites for each new leap in computer applications. Or,

-- Clay Shirky

to quote again from Shirky, "With peer-to-peer the computer is no longer just a life-support system for your browser."

Performance concerns many observers. Does peer-to-peer cause unnecessary processing and Internet traffic? Napster is not valid evidence for these concerns, because exchanging large files would stress networks regardless of the technology used to administer the transaction. Many expect peer-to-peer to actually conserve bandwidth when used appropriately.

Intel hopes that they can avoid the crippling effect of users downloading training videos by letting files relocate near groups of systems that request them.

One such observer is Bob Knighten, peer-to-peer evangelist for Intel. His company hopes that they can avoid the crippling effect of users downloading training videos by letting files relocate near groups of systems that request them. Comparable to caching by Internet hubs or multicasting, this technique is distinguished by its use of regular, underutilized computers instead of specialized, expensive file servers. The model of shifting content to where it's wanted is also the basis of Freenet.

But there's no doubt that peer-to-peer will challenge the architecture of current Internet services. Nelson Minar, cofounder of a distributed computing startup named Popular Power, says that peer-to-peer redefines the assumptions behind asymmetric service (like ADSL and cable modems). Michael Tiemann, CTO of Red Hat, adopts a positive attitude and goes so far as to say, "Peer-to-peer may be the critical enabling technology that makes broadband possible."

Flexibility offers a key opportunity

Traditional search services may offer a variety of advanced options, but you can't tell the service how to organize its data. Napster is child's play for finding pop songs, but its rigid classification service doesn't accommodate classical music well.

Peer-to-peer raises the possibility for people interested in a topic to create their own language for talking about it. While different communities may all share an underlying infrastructure, like Jabber's chat service or Gnutella file sharing, the structure of the users' data can emerge directly from the users.

Metadata, which describes each file and the elements within it, holds the key to self-organization. XML is a good foundation -- but only a foundation, because it just offers a syntax. Building on the XML foundation, schemas hold some promise for structuring both content and users' reactions to the content. One slogan we considered was, "Publish my taste, not just my music files."

Schemas can be handed down by standards committees, but they can also be thrown out on the Net by individuals or communities for widespread consideration. Each community will settle on the ones it likes. We called this principle "schema agnosticism."

While we all paid homage to the glories of metadata, we remained skeptical that people would understand the need for it and take the trouble to contribute information. Rael Dornfest, creator of the RSS-based Meerkat service for the O'Reilly Network, said that a little metadata goes a long way, adding logarithmically to the value and uses of the data it describes.

Any service or tool that provides either automated or manual access to reading and writing metadata offers an opening for an intrepid explorer to map the data and thus create an innovative service for other users. (Invisible Worlds is promoting a generalized approach to such APIs through the Simple Exchange Protocol, currently an Internet Draft.) This is a good reason to ask services to expose their data schemas and publish open APIs to their services. It's also a reason to

deplorable attempts to shut off innovation through bans on deep linking, restrictions on the amount of data one user can retrieve, or forcing developers to resort to "screen-scraping" and other fragile data-retrieval tricks.

The message

Peer-to-peer is so new that we have trouble attaching categorical statements to it. Yet it is also the oldest model in the world of communications. Telephones are peer-to-peer, as are Fidonet and the old-style UUCP implementation of Usenet. IP routing, the basis of the Internet, is peer-to-peer, even now when the largest access points raise themselves above the rest.

Up until the past decade, every Internet-connected system hosted both servers and clients. Aside from dial-up users, the second-class status of today's PC browser set didn't exist. One of our messages, therefore, is:

Peer-to-peer is fundamental to the architecture of the Internet.

Another element of our message is that peer-to-peer brings people together; it builds communities in a way that eludes all the portals that so earnestly try to build them from the top down. Jabber, by providing chatters with an easy way to categorize their content, hopes to help them organize themselves. Weblogs, Wiki, and Meerkat let people follow and comment on the goings-on of other people who interest them. Even a passive sharing of resources, like Popular Power or SETI@Home, makes people feel they're part of something big -- and they like the feeling, according to Minar.

When people collaborate, each has the opportunity -- though not the requirement -- of being a producer as well as a consumer. Those who produce may be relatively few: For instance, a recent study found that only 2% of Gnutella users contribute content, and even on Usenet News the ratio of posters to total readers is only about 7%. But that is enough to keep the system afloat. The important thing is to give everybody the opportunity. Thus our slogans:

Peer-to-peer is the end of the read-only Web.

Peer-to-peer allows you to participate in the Internet again.

Peer-to-peer: steering the Internet away from TV.

Since each peer represents a person (and in fact, a person can appear in many different guises on one or more peer-to-peer systems), these systems can lead to emerging, self-organizing communities.

Some peer-to-peer systems deal in fungible resources, such as the use of idle computing power by SETI@Home or Popular Power. In these systems, resources are valuable because they're interchangeable. But in many systems, where the goal is to share files or metadata, the *diversity* of the peers is what makes the system valuable. Resources like disk space may start out fungible but then develop differences as users add unique content.

Limitations

Ray Ozzie, the developer of Lotus Notes and more recently the founder of Groove Networks, asked us to come up with a clear marketing slogan that would let a busy system designer decide whether peer-to-peer was right for his system. We didn't succeed at finding a pithy summary, but we did make a list of applications that raise a warning flag. There are situations where peer-to-peer is counterindicated.

Peer-to-peer is probably not appropriate when it's important for everybody to know all the current data at the same time. You probably can't run a stock market with peer-to-peer, for instance, because fair pricing fundamentally depends on everybody knowing the asking price and seeing the impact of all bids. An airline reservation system wouldn't be appropriate either, unless each buyer is willing to sit on somebody else's lap. We did not achieve consensus on whether peer-to-peer applications are good for rapidly changing data.

Peer-to-peer systems seem to contain an inherent fuzziness. Gnutella, for instance, doesn't promise you'll find a file that's hosted on its system; it has a "horizon" beyond which you can't see what's there. Knighten said, "There are a lot of things in the peer-to-peer world that we can take advantage of only if we give up certitude." Most computers come and go on the Internet, so that the host for a particular resource may be there one minute and gone the next.

Furthermore, peer-to-peer seems to reward large communities. If very few people ask for a file on Freenet, it might drop off the caches of the various participating systems. Steve Burbeck, a Senior Technical Staff Member at IBM, pointed out that self-organizing file sharing systems like Napster, Gnutella and Freenet are affected by the popularity of files, and hence may be susceptible to the tyranny of the majority.

Some participants suggested that peer-to-peer is good if you have a strong idea what you're looking for (an author, for instance) but have no idea where it is. A centralized system like Yahoo! is better if you have only a vague idea what you're looking for, but possess some confidence that you can find it at a particular site.

Darren New of Invisible Worlds explained that the value of knowing what you want in advance gives the advantage to peer-to-peer systems where a lot of information is available to potential users out-of-band. This rather abstract statement becomes meaningful when one considers Napster. No one would be able to find much on Napster if they couldn't obtain the name of a band and its songs beforehand. Ironically, many people depend on traditional sales channels like Amazon.com to obtain names of songs before they troop over to Napster and download them. Thus, Napster requires a pre-existing, outside infrastructure.

Furthermore, data is easy to find if it's easy to categorize. Tim O'Reilly pointed out that people would have an easier time searching a peer-to-peer system for a pop song than for a particular combination of performers singing an opera number -- and they'd have even more trouble searching for a chapter in an O'Reilly book. Metadata, once again, proves key to the future of peer-to-peer.

Even if you don't know what you want, you might be able to find it by contacting a knowledgeable member of your online cohort.

Peer-to-peer does offer an alternative strength, though: connecting people. Even if you don't know what you want, you might be able to find it by contacting a knowledgeable member of your online cohort. You might even find your request met by an infobot (which records frequently asked questions along with their answers, and spits back answers to newsgroup users), which is a functioning application developed by Kevin Lenzo of Carnegie Mellon University.

Historically, the balance between centralized systems and peer-to-peer shifts back and forth. Many systems, like Usenet and IP routing, started off as pure peer-to-peer but developed some degree of hierarchy as they got bigger. Other systems started out centralized but got too big for it, like the Domain Name System (which had to give up single hosts files). Because of peer-to-peer's scalability, some systems are clearly going to require it. To quote IBM's Burbeck, "When you get up to a billion cellular phones and other devices, you have to be decentralized."

Barriers to peer-to-peer

As with any new technology, peer-to-peer has to grapple with old assumptions, which have been embodied in policies ranging in size from a site administrator's firewall configuration to acts of Congress.

Bandwidth limitations

These have already been mentioned. Low-cost technologies can deliver high bandwidth downstream, but cheat the user who wants to publish as well as read. License restrictions such as bans on running a service out of the home, while justifiable from the viewpoint of maintaining adequate throughput on multiple systems, also amputate the possibilities of peer-to-peer.

Firewalls and address translation

When it comes to IP addresses, IPv6 may eliminate the scarcity but not the scares. Most firewall administrators keep their users from getting fixed IP addresses out of fear that they make intrusion easier. The sense of security may be illusory (because people monitoring their own ports have reported intrusion attempts within a few minutes of bringing up their network software), but the practice makes it much harder for an end user's system to host a service.

Even if each user gets a fixed IP address, the Domain Name System is a barrier to end-to-end addressing. Domain names have become a significant expense over the years, and ICANN is not improving the situation. (For instance, they require anyone who wants a new top level domain to pay \$50,000 just for the privilege of having the request considered.) For that reason, new services like AOL Instant Messenger and Napster bypass DNS and use their own addressing.

Unfortunately, such ad-hoc addressing solutions lead to problems of their own. AOL tries to shut out competitors by denying them its addresses. Systems that use proprietary systems of addressing can't interoperate. Most ad-hoc namespaces are flat and crude.

Finally, packet filtering, while it plays an important role in securing a LAN, also leads to "port 80 pollution" as everybody tries to use the Web for services that would be more appropriate with a different protocol. The solution to this problem, unfortunately, will not be in sight until operating systems become more secure and IP security can be set up with a few mouse clicks (or vocal commands).

Regulation

Government restrictions have not yet affected peer-to-peer technologies, but could have a chilling effect in the future. Scott Miller, lead U.S. developer for Freenet, pointed out that early peer-to-peer systems happen to be associated with violations of intellectual property (at least in the minds of the copyright holders) but that such activities were neither the purpose of peer-to-peer nor the most likely types of mature applications to emerge. Nevertheless, a couple of participants reported that peer-to-peer has been branded with a scarlet I for Infringement in the minds of investors, courts, and the general public.

Software patents could also stifle technology. We spent some time discussing ways to use prior art searches of patents defensively.

Barriers to entry

Finally, things that currently make it hard for users to start a peer-to-peer service need to be attacked by the peer-to-peer developers themselves. If every new peer-to-peer service requires the user to download, install, and configure a new program, it will remain a craft shop for tinkerers. Shirky pointed out that Java was supposed to provide the universal client that allowed everything else to run seamlessly, but Java on the client didn't pan out. We need something as

easy as the update service offered by Windows and some other products.

Even after users install a client, they may abort their involvement if they are forced to learn a system of metadata and apply a lot of tags. Jon Udell, programming consultant and Byte Magazine columnist, said, "When everybody's a publisher, responsibility goes along with that." Dornfest called for local vocabularies to provide just the right depth of metadata, combined with clients that allow its application with just a mouse click, a keystroke, or automagically.

Napster made the registration of users' MP3 files nearly automatic. Janelle Brown, senior writer at Salon, reminded us of the sad history of Hotline, a service similar to Napster that appeared much earlier but never took off. Among the various diagnoses the group offered in post mortem, it was pointed out that prospective users had to upload material before doing any downloads. That restriction may have turned them off. Nevertheless, another quid-pro-quo system -- where you have to provide a resource in order to use a resource -- is being tried by the new venture MojoNation.

Trust

The ability to trust the system and other users is critical. Balancing the privacy of the individual with the need to authenticate users and kick malicious disrupters off the system is a difficult feat.

Future developments

In this article I've tried to summarize the most interesting themes from the September 2000 peer-to-peer summit. Leaders in the field are exploring more such themes in the book *Peer-to-Peer*, to be released by O'Reilly & Associates in February, and at the O'Reilly Peer-To-Peer Conference in San Francisco, which takes place February 14-16. I hope you'll stick around -- and don't be afraid to participate.

Andy Oram is an editor at O'Reilly & Associates specializing in books on Linux and programming. Most recently, he edited *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*.

Related Articles:

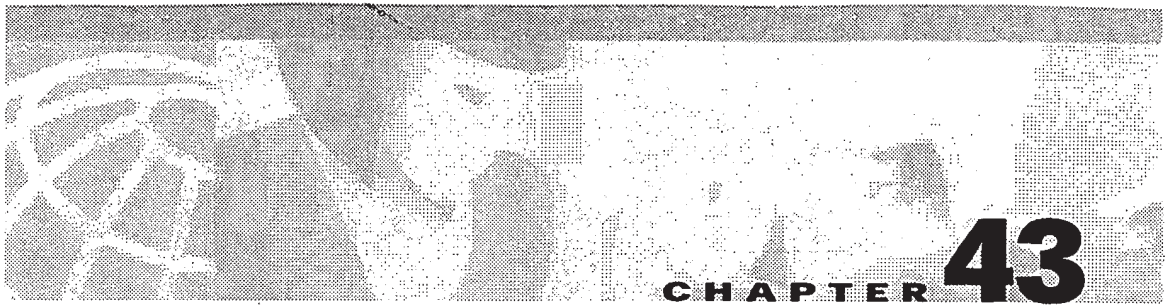
[Open Source Roundtable: O'Reilly's Peer-to-Peer Summit](#) 

[Peer-to-Peer Summit Attendees](#)

Discuss this article in the O'Reilly Network [Linux Forum](#).

Return to the [Linux DevCenter](#).

oreillynet.com Copyright © 2000 O'Reilly & Associates, Inc.



Chapter Goals

- Explain IP multicast addressing.
- Learn the basics of Internet Group Management Protocol (IGMP).
- Explain how multicast in Layer 2 switching works.
- Define multicast distribution trees.
- Learn how multicast forwarding works.
- Explain the basics of protocol-independent multicast (PIM).
- Define multiprotocol BGP.
- Learn how Multicast Source Discovery Protocol (MSDP) works.
- Explain reliable multicast: PGM.

Internet Protocol Multicast

Background

Internet Protocol (IP) multicast is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to thousands of corporate recipients and homes. Applications that take advantage of multicast include videoconferencing, corporate communications, distance learning, and distribution of software, stock quotes, and news.

IP Multicast delivers source traffic to multiple receivers without adding any additional burden on the source or the receivers while using the least network bandwidth of any competing technology. Multicast packets are replicated in the network by Cisco routers enabled with Protocol Independent Multicast (PIM) and other supporting multicast protocols resulting in the most efficient delivery of data to multiple receivers possible. All alternatives require the source to send more than one copy of the data. Some even require the source to send an individual copy to each receiver. If there are thousands of receivers, even low-bandwidth applications benefit from using Cisco IP Multicast. High-bandwidth applications, such as MPEG video, may require a large portion of the available network bandwidth for a single stream. In these applications, the only way to send to more than one receiver simultaneously is by using IP Multicast. Figure 43-1 demonstrates how data from one source is delivered to several interested recipients using IP multicast.

Figure 43-1 Multicast Transmission Sends a Single Multicast Packet Addressed to All Intended Recipients

Multicast Group Concept

Multicast is based on the concept of a group. An arbitrary group of receivers expresses an interest in receiving a particular data stream. This group does not have any physical or geographical boundaries—the hosts can be located anywhere on the Internet. Hosts that are interested in receiving data flowing to a particular group must join the group using IGMP. Hosts must be a member of the group to receive the data stream.

IP Multicast Addresses

Multicast addresses specify an arbitrary group of IP hosts that have joined the group and want to receive traffic sent to this group.

IP Class D Addresses

The *Internet Assigned Numbers Authority (IANA)* controls the assignment of IP multicast addresses. It has assigned the old Class D address space to be used for IP multicast. This means that all IP multicast group addresses will fall in the range of 224.0.0.0 to 239.255.255.255.



Note

This address range is only for the group address or destination address of IP multicast traffic. The source address for multicast datagrams is always the unicast source address.

Reserved Link Local Addresses

The IANA has reserved addresses in the 224.0.0.0 through 224.0.0.255 to be used by network protocols on a local network segment. Packets with these addresses should never be forwarded by a router; they remain local on a particular LAN segment. They are always transmitted with a time-to-live (TTL) of 1. Network protocols use these addresses for automatic router discovery and to communicate important routing information. For example, OSPF uses 224.0.0.5 and 224.0.0.6 to exchange link state information. Table 43-1 lists some of the well-known addresses.

Table 43-1 Link Local Addresses

Address	Usage
224.0.0.1	All systems on this subnet
224.0.0.2	All routers on this subnet
224.0.0.5	OSPF routers
224.0.0.6	OSPF designated routers
224.0.0.12	DHCP server/relay agent

Globally Scoped Address

The range of addresses from 224.0.1.0 through 238.255.255.255 are called globally scoped addresses. They can be used to multicast data between organizations and across the Internet.

Some of these addresses have been reserved for use by multicast applications through IANA. For example, 224.0.1.1 has been reserved for Network Time Protocol (NTP).

More information about reserved multicast addresses can be found at <http://www.isi.edu/in-notes/iana/assignments/multicast-addresses>.

Limited Scope Addresses

The range of addresses from 239.0.0.0 through 239.255.255.255 contains limited scope addresses or administratively scoped addresses. These are defined by RFC 2365 to be constrained to a local group or organization. Routers are typically configured with filters to prevent multicast traffic in this address range from flowing outside an autonomous system (AS) or any user-defined domain. Within an autonomous system or domain, the limited scope address range can be further subdivided so those local multicast boundaries can be defined. This also allows for address reuse among these smaller domains.

Glop Addressing

RFC 2770 proposes that the 233.0.0.0/8 address range be reserved for statically defined addresses by organizations that already have an AS number reserved. The AS number of the domain is embedded into the second and third octets of the 233.0.0.0/8 range.

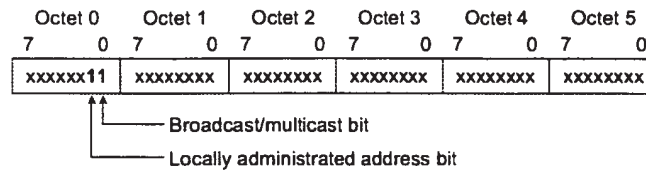
For example, the AS 62010 is written in hex as F23A. Separating out the two octets F2 and 3A, we get 242 and 58 in decimal. This would give us a subnet of 233.242.58.0 that would be globally reserved for AS 62010 to use.

Layer 2 Multicast Addresses

Normally, network interface cards (NICs) on a LAN segment will receive only packets destined for their burned-in MAC address or the broadcast MAC address. Some means had to be devised so that multiple hosts could receive the same packet and still be capable of differentiating among multicast groups.

Fortunately, the IEEE LAN specifications made provisions for the transmission of broadcast and/or multicast packets. In the 802.3 standard, bit 0 of the first octet is used to indicate a broadcast and/or multicast frame. Figure 43-2 shows the location of the broadcast/multicast bit in an Ethernet frame.

Figure 43-2 IEEE 802.3 MAC Address Format



This bit indicates that the frame is destined for an arbitrary group of hosts or all hosts on the network (in the case of the broadcast address, 0xFFFF.FFFF.FFFF).

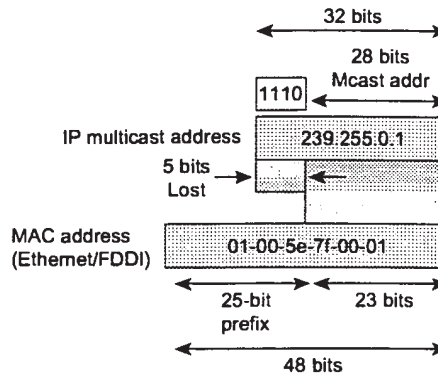
IP multicast makes use of this capability to transmit IP packets to a group of hosts on a LAN segment.

Ethernet MAC Address Mapping

The IANA owns a block of Ethernet MAC addresses that start with 01:00:5E in hexadecimal. Half of this block is allocated for multicast addresses. This creates the range of available Ethernet MAC addresses to be 0100.5e00.0000 through 0100.5e7f.ffff.

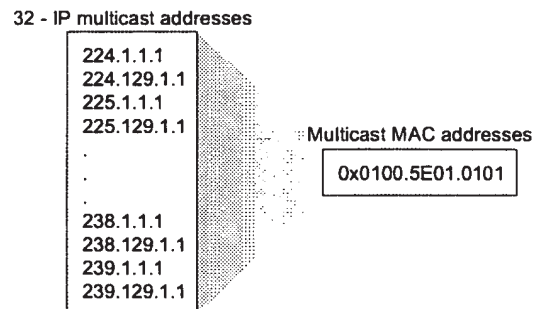
This allocation allows for 23 bits in the Ethernet address to correspond to the IP multicast group address. The mapping places the lower 23 bits of the IP multicast group address into these available 23 bits in the Ethernet address (shown in Figure 43-3).

Figure 43-3 Mapping of IP Multicast to Ethernet/FDDI MAC Address



Because the upper 5 bits of the IP multicast address are dropped in this mapping, the resulting address is not unique. In fact, 32 different multicast group IDs all map to the same Ethernet address (see Figure 43-4).

Figure 43-4 MAC Address Ambiguities



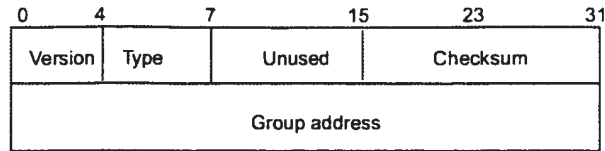
Internet Group Management Protocol

IGMP is used to dynamically register individual hosts in a multicast group on a particular LAN. Hosts identify group memberships by sending IGMP messages to their local multicast router. Under IGMP, routers listen to IGMP messages and periodically send out queries to discover which groups are active or inactive on a particular subnet.

IGMP Version 1

RFC 1112 defines the specification for IGMP Version 1. A diagram of the packet format is found in Figure 43-5.

Figure 43-5 IGMP Version 1 Packet Format



In Version 1, there are just two different types of IGMP messages:

- Membership query
- Membership report

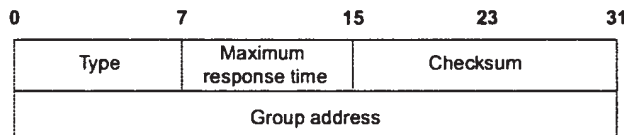
Hosts send out IGMP membership reports corresponding to a particular multicast group to indicate that they are interested in joining that group. The router periodically sends out an IGMP membership query to verify that at least one host on the subnet is still interested in receiving traffic directed to that group. When there is no reply to three consecutive IGMP membership queries, the router times out the group and stops forwarding traffic directed toward that group.

IGMP Version 2

RFC 2236 defines the specification for IGMP Version 2.

A diagram of the packet format follows in Figure 43-6.

Figure 43-6 IGMPv2 Message Format



In Version 2, there are four types of IGMP messages:

- Membership query
- Version 1 membership report
- Version 2 membership report
- Leave group

IGMP Version 2 works basically the same as Version 1. The main difference is that there is a leave group message. The hosts now can actively communicate to the local multicast router their intention to leave the group. The router then sends out a group-specific query and determines whether there are any remaining hosts interested in receiving the traffic. If there are no replies, the router times out the group and stops forwarding the traffic. This can greatly reduce the leave latency compared to IGMP Version 1. Unwanted and unnecessary traffic can be stopped much sooner.

Multicast in the Layer 2 Switching Environment

The default behavior for a Layer 2 switch is to forward all multicast traffic to every port that belongs to the destination LAN on the switch. This would defeat the purpose of the switch, which is to limit traffic to the ports that need to receive the data.

Two methods exist by which to deal with multicast in a Layer 2 switching environment efficiently—Cisco Group Management Protocol (CGMP) and IGMP snooping.

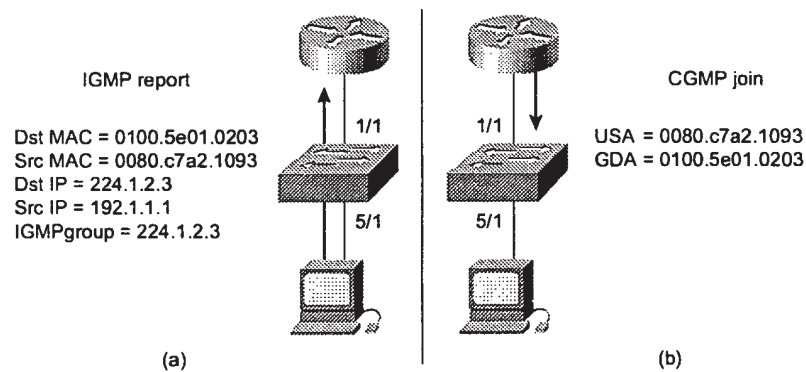
Cisco Group Management Protocol

CGMP is a Cisco-developed protocol that allows Catalyst switches to leverage IGMP information on Cisco routers to make Layer 2 forwarding decisions. *CGMP* must be configured both on the multicast routers and on the Layer 2 switches. The net result is that with *CGMP*, IP multicast traffic is delivered only to those Catalyst switch ports that are interested in the traffic. All other ports that have not explicitly requested the traffic will not receive it.

The basic concept of *CGMP* is shown in Figure 43-7. When a host joins a multicast group (part A), it multicasts an unsolicited IGMP membership report message to the target group (224.1.2.3, in this example). The IGMP report is passed through the switch to the router for the normal IGMP processing. The router (which must have *CGMP* enabled on this interface) receives this IGMP report and processes it as it normally would, but in addition it creates a *CGMP* join message and sends it to the switch.

The switch receives this *CGMP* join message and then adds the port to its content addressable memory (CAM) table for that multicast group. Subsequent traffic directed to this multicast group will be forwarded out the port for that host. The router port is also added to the entry for the multicast group. Multicast routers must listen to all multicast traffic for every group because the IGMP control messages are also sent as multicast traffic. With *CGMP*, the switch must listen only to *CGMP* join and *CGMP* leave messages from the router. The rest of the multicast traffic is forwarded using its CAM table exactly the way the switch was designed.

Figure 43-7 Basic *CGMP* Operation



IGMP Snooping

IGMP snooping requires the LAN switch to examine, or snoop, some Layer 3 information in the IGMP packets sent between the hosts and the router. When the switch hears the IGMP host report from a host for a particular multicast group, the switch adds the host's port number to the associated multicast table entry. When the switch hears the IGMP leave group message from a host, it removes the host's port from the table entry.

Because IGMP control messages are transmitted as multicast packets, they are indistinguishable from multicast data at Layer 2. A switch running IGMP snooping examines every multicast data packet to check whether it contains any pertinent IGMP control information. If IGMP snooping has been implemented on a low-end switch with a slow CPU, this could have a severe performance impact when

data is transmitted at high rates. The solution is to implement IGMP snooping on high-end switches with special ASICs that can perform the IGMP checks in hardware. CGMP is ideal for low-end switches without special hardware.

Multicast Distribution Trees

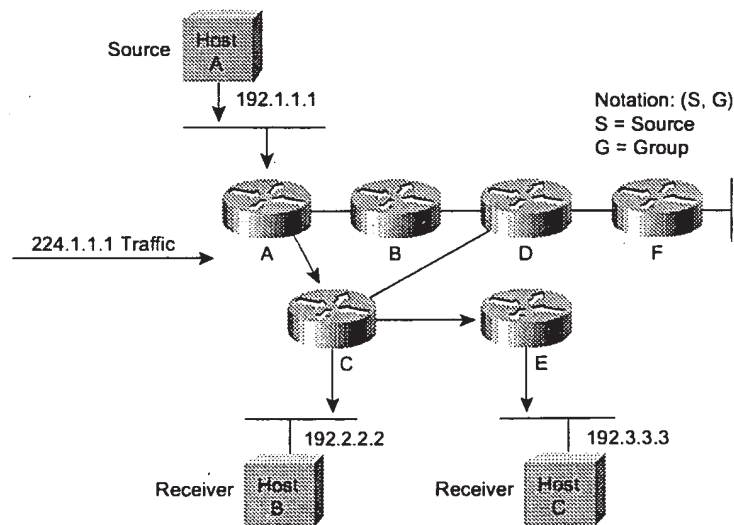
Multicast-capable routers create distribution trees that control the path that IP multicast traffic takes through the network to deliver traffic to all receivers. The two basic types of multicast distribution trees are source trees and shared trees.

Source Trees

The simplest form of a multicast distribution tree is a *source tree* whose root is the source of the multicast tree and whose branches form a spanning tree through the network to the receivers. Because this tree uses the shortest path through the network, it is also referred to as a shortest path tree (SPT).

Figure 43-8 shows an example of an SPT for group 224.1.1.1 rooted at the source, Host A, and connecting two receivers, hosts B and C.

Figure 43-8 Host A Shortest Path Tree



The special notation of (S,G), pronounced “S comma G,” enumerates an SPT in which S is the IP address of the source and G is the multicast group address. Using this notation, the SPT for the example in Figure 43-7 would be (192.1.1.1, 224.1.1.1).

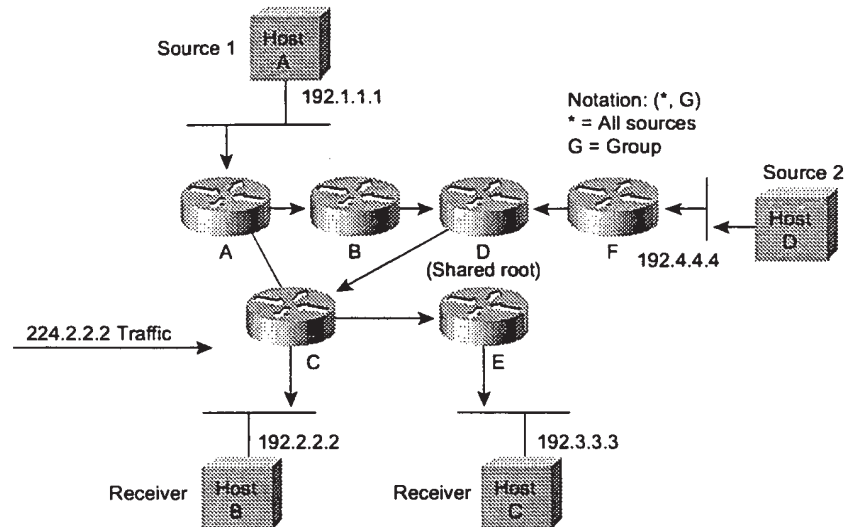
The (S,G) notation implies that a separate SPT exists for each individual source sending to each group, which is correct. For example, if Host B is also sending traffic to group 224.1.1.1 and hosts A and C are receivers, then a separate (S,G) SPT would exist with a notation of (192.2.2.2, 224.1.1.1).

Shared Trees

Unlike source trees that have their root at the source, *shared trees* use a single common root placed at some chosen point in the network. This shared root is called the *rendezvous point (RP)*.

Figure 43-9 shows a shared tree for the group 224.2.2.2 with the root located at Router D. When using a shared tree, sources must send their traffic to the root, and then the traffic is forwarded down the shared tree to reach all receivers.

Figure 43-9 Shared Distribution Tree



In this example, multicast traffic from the source hosts A and D travels to the root (Router D) and then down the shared tree to the two receivers, hosts B and C. Because all sources in the multicast group use a common shared tree, a wildcard notation written as (*, G), pronounced “star comma G,” represents the tree. In this case, * means all sources, and the G represents the multicast group. Therefore, the shared tree shown in Figure 43-8 would be written as (*, 224.2.2.2).

Both SPT and shared trees are loop-free. Messages are replicated only where the tree branches.

Members of multicast groups can join or leave at any time, so the distribution trees must be dynamically updated. When all the active receivers on a particular branch stop requesting the traffic for a particular multicast group, the routers prune that branch from the distribution tree and stop forwarding traffic down that branch. If one receiver on that branch becomes active and requests the multicast traffic, the router dynamically modifies the distribution tree and starts forwarding traffic again.

Shortest path trees have the advantage of creating the optimal path between the source and the receivers. This guarantees the minimum amount of network latency for forwarding multicast traffic. This optimization does come with a price, though: The routers must maintain path information for each source. In a network that has thousands of sources and thousands of groups, this can quickly become a resource issue on the routers. Memory consumption from the size of the multicast routing table is a factor that network designers must take into consideration.

Shared trees have the advantage of requiring the minimum amount of state in each router. This lowers the overall memory requirements for a network that allows only shared trees. The disadvantage of shared trees is that, under certain circumstances, the paths between the source and receivers might not be the optimal paths—which might introduce some latency in packet delivery. Network designers must carefully consider the placement of the RP when implementing an environment with only shared trees.

Multicast Forwarding

In unicast routing, traffic is routed through the network along a single path from the source to the destination host. A unicast router does not really care about the source address—it only cares about the destination address and how to forward the traffic towards that destination. The router scans through its routing table and then forwards a single copy of the unicast packet out the correct interface in the direction of the destination.

In multicast routing, the source is sending traffic to an arbitrary group of hosts represented by a multicast group address. The multicast router must determine which direction is upstream (toward the source) and which direction (or directions) is downstream. If there are multiple downstream paths, the router replicates the packet and forwards the traffic down the appropriate downstream paths—which is not necessarily all paths. This concept of forwarding multicast traffic away from the source, rather than to the receiver, is called *reverse path forwarding*.

Reverse Path Forwarding

Reverse path forwarding (RPF) is a fundamental concept in multicast routing that enables routers to correctly forward multicast traffic down the distribution tree. RPF makes use of the existing unicast routing table to determine the upstream and downstream neighbors. A router forwards a multicast packet only if it is received on the upstream interface. This RPF check helps to guarantee that the distribution tree will be loop-free.

RPF Check

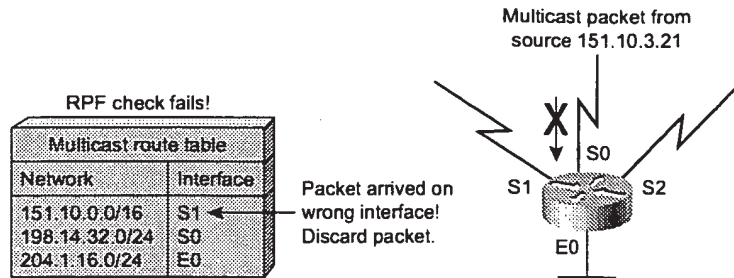
When a multicast packet arrives at a router, the router performs an RPF check on the packet. If the RPF check is successful, the packet is forwarded. Otherwise, it is dropped.

For traffic flowing down a source tree, the RPF check procedure works as follows:

-
- Step 1** Router looks up the source address in the unicast routing table to determine whether it has arrived on the interface that is on the reverse path back to the source.
 - Step 2** If packet has arrived on the interface leading back to the source, the RPF check is successful and the packet is forwarded.
 - Step 3** If the RPF check in Step 2 fails, the packet is dropped.

Figure 43-10 shows an example of an unsuccessful RPF check.

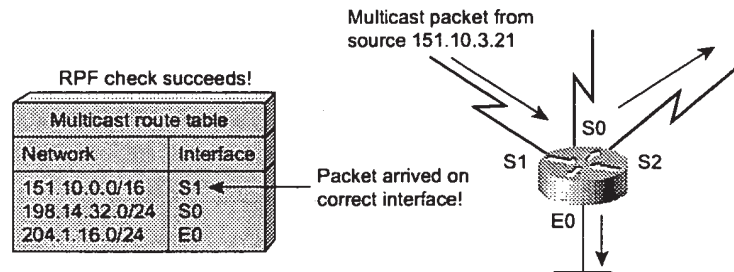
Figure 43-10 RPF Check Fails



A multicast packet from source 151.10.3.21 is received on interface S0. A check of the unicast route table shows that the interface that this router would use to forward unicast data to 151.10.3.21 is S1. Because the packet has arrived on S0, the packet will be discarded.

Figure 43-11 shows an example of a successful RPF check.

Figure 43-11 RPF Check Succeeds



This time the multicast packet has arrived on S1. The router checks the unicast routing table and finds that S1 is the correct interface. The RPF check passes and the packet is forwarded.

Protocol-Independent Multicast

Protocol-independent multicast (PIM) gets its name from the fact that it is IP routing protocol-independent. PIM can leverage whichever unicast routing protocols are used to populate the unicast routing table, including EIGRP, OSPF, BGP, or static routes. PIM uses this unicast routing information to perform the multicast forwarding function, so it is IP protocol-independent. Although PIM is called a multicast routing protocol, it actually uses the unicast routing table to perform the reverse path forwarding (RPF) check function instead of building up a completely independent multicast routing table. PIM does not send and receive multicast routing updates between routers like other routing protocols do.

PIM Dense Mode

PIM Dense Mode (PIM-DM) uses a push model to flood multicast traffic to every corner of the network. This is a brute-force method for delivering data to the receivers, but in certain applications, this might be an efficient mechanism if there are active receivers on every subnet in the network.

PIM-DM initially floods multicast traffic throughout the network. Routers that do not have any downstream neighbors prune back the unwanted traffic. This process repeats every 3 minutes.

The flood and prune mechanism is how the routers accumulate their state information—by receiving the data stream. These data streams contain the source and group information so that downstream routers can build up their multicast forwarding tables. PIM-DM can support only source trees—(S,G) entries. It cannot be used to build a shared distribution tree.

PIM Sparse Mode

PIM Sparse Mode (PIM-SM) uses a pull model to deliver multicast traffic. Only networks that have active receivers that have explicitly requested the data will be forwarded the traffic. PIM-SM is defined in RFC 2362.

PIM-SM uses a shared tree to distribute the information about active sources. Depending on the configuration options, the traffic can remain on the shared tree or switch over to an optimized source distribution tree. The latter is the default behavior for PIM-SM on Cisco routers. The traffic starts to flow down the shared tree, and then routers along the path determine whether there is a better path to the source. If a better, more direct path exists, the designated router (the router closest to the receiver) will send a join message toward the source and then reroute the traffic along this path.

PIM-SM has the concept of an RP, since it uses shared trees—at least initially. The RP must be administratively configured in the network. Sources register with the RP, and then data is forwarded down the shared tree to the receivers. If the shared tree is not an optimal path between the source and the receiver, the routers dynamically create a source tree and stop traffic from flowing down the shared tree. This is the default behavior in IOS. Network administrators can force traffic to stay on the shared tree by using a configuration option (`ip pim spt-threshold infinity`).

PIM-SM scales well to a network of any size, including those with WAN links. The explicit join mechanism prevents unwanted traffic from flooding the WAN links.

Sparse-Dense Mode

Cisco has implemented an alternative to choosing just dense mode or just sparse mode on a router interface new IP. This was necessitated by a change in the paradigm for forwarding multicast traffic via PIM that became apparent during its development. It turned out that it was more efficient to choose sparse or dense on a per group basis rather than a per router interface basis. Sparse-dense mode facilitates this ability.

Network administrators can also configure sparse-dense mode. This configuration option allows individual groups to be run in either sparse or dense mode, depending on whether RP information is available for that group. If the router learns RP information for a particular group, it will be treated as sparse mode; otherwise, that group will be treated as dense mode.

Multiprotocol Border Gateway Protocol

Multiprotocol Border Gateway Protocol (MBGP) gives a method for providers to distinguish which route prefixes they will use for performing multicast RPF checks. The RPF check is the fundamental mechanism that routers use to determine the paths that multicast forwarding trees will follow and successfully deliver multicast content from sources to receivers.

MBGP is described in RFC 2283, Multiprotocol Extensions for BGP-4. Since MBGP is an extension of BGP, it brings along all the administrative machinery that providers and customers like in their interdomain routing environment. Including all the inter-AS tools to filter and control routing (e.g., route maps). Therefore, by using MBGP, any network utilizing internal or external BGP can apply the multiple policy control knobs familiar in BGP to specify routing (and thereby forwarding) policy for multicast.

Two path attributes, MP_REACH_NLRI and MP_UNREACH_NLRI have been introduced in BGP4+. These new attributes create a simple way to carry two sets of routing information—one for unicast routing and one for multicast routing. The routes associated with multicast routing are used to build the multicast distribution trees.

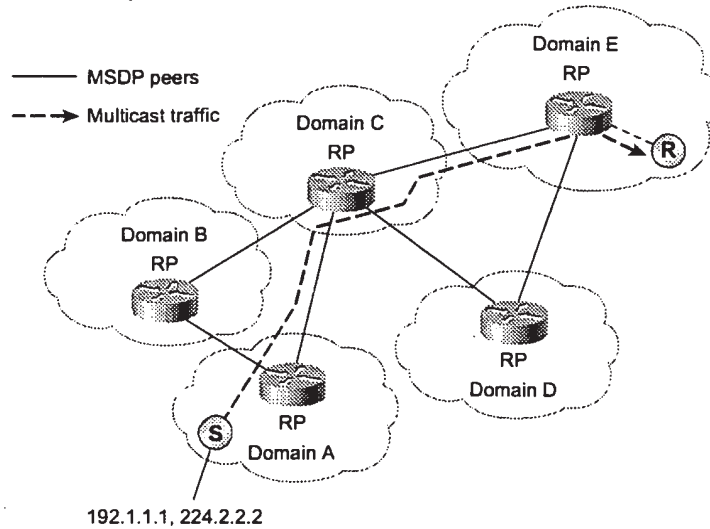
The main advantage of MBGP is that an internet can support noncongruent unicast and multicast topologies. When the unicast and multicast topologies are congruent, MBGP can support different policies for each. MBGP provides a scalable policy based interdomain routing protocol.

Multicast Source Discovery Protocol

In the PIM Sparse mode model, multicast sources and receivers must register with their local Rendezvous Point (RP). Actually, the closest router to the sources or receivers registers with the RP but the point is that the RP knows about all the sources and receivers for any particular group. RPs in other domains have no way of knowing about sources located in other domains. MSDP is an elegant way to solve this problem. MSDP is a mechanism that connects PIM-SM domains and allows RPs to share information about active sources. When RPs in remote domains know about active sources they can pass on that information to their local receivers and multicast data can be forwarded between the domains. A nice feature of MSDP is that it allows each domain to maintain an independent RP which does not rely on other domains, but it does enable RPs to forward traffic between domains.

The RP in each domain establishes an MSDP peering session using a TCP connection with the RPs in other domains or with border routers leading to the other domains. When the RP learns about a new multicast source within its own domain (through the normal PIM register mechanism), the RP encapsulates the first data packet in a Source Active (SA) message and sends the SA to all MSDP peers. The SA is forwarded by each receiving peer using a modified RPF check, until it reaches every MSDP router in the interconnected networks—theoretically the entire multicast internet. If the receiving MSDP peer is an RP, and the RP has a (*,G) entry for the group in the SA (there is an interested receiver), the RP will create (S,G) state for the source and join to the shortest path tree for the state of the source. The encapsulated data is decapsulated and forwarded down that RP's shared tree. When the packet is received by a receiver's last hop router, the last-hop may also join the shortest path tree to the source. The source's RP periodically sends SAs, which include all sources within that RP's own domain. Figure 43-12 shows how data would flow between a source in domain A to a receiver in domain E.

Figure 43-12 MSDP Example



MSDP was developed for peering between Internet Service Providers (ISPs). ISPs did not want to rely on an RP maintained by a competing ISP to service their customers. MSDP allows each ISP to have their own local RP and still forward and receive multicast traffic to the Internet.

Anycast RP-Logical RP

A very useful application of MSDP is called anycast RP. This is a technique for configuring a multicast sparse-mode network to provide for fault tolerance and load sharing within a single multicast domain.

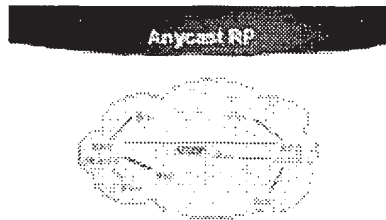
Two or more RPs are configured with the same IP address on loopback interfaces—say, 10.0.0.1, for example (refer to Figure 43-13). The loopback address should be configured as a 32 bit address. All the downstream routers are configured so that they know that their local RP's address is 10.0.0.1. IP routing automatically selects the topologically closest RP for each source and receiver. Because some sources might end up using one RP and some receivers a different RP, there needs to be some way for the RPs to exchange information about active sources. This is done with MSDP. All the RPs are configured to be MSDP peers of each other. Each RP will know about the active sources in the other RP's area. If any of the RPs fail, IP routing will converge and one of the RPs will become the active RP in both areas.



Note

The Anycast RP example above uses IP addresses from RFC 1918. These IP addresses are normally blocked at interdomain borders and therefore are not accessible to other ISPs. You must use valid IP addresses if you want the RPs to be reachable from other domains.

Figure 43-13 Anycast RP



Note

The RPs are used only to set up the initial connection between sources and receivers. After the last-hop routers join the shortest path tree, the RP is no longer necessary.

Multicast Address Dynamic Client Allocation Protocol

The *Multicast Address Dynamic Client Allocation Protocol (MADCAP)* is defined in RFC 2730 as a protocol that allows hosts to request a multicast address allocation dynamically from a MADCAP server. The concept is very similar to the way DHCP works today and is built on a client/server model.

Multicast-Scope Zone Announcement Protocol

Multicast-Scope Zone Announcement Protocol (MZAP) is defined in RFC 2776 as a protocol that allows networks to automatically discover administratively scoped zones relative to a particular location.

Reliable Multicast-Pragmatic General Multicast

Pragmatic General Multicast (PGM) is a reliable multicast transport protocol for applications that require ordered, duplicate-free, multicast data delivery from multiple sources to multiple receivers. PGM guarantees that a receiver in a multicast group either receives all data packets from transmissions and retransmissions, or can detect unrecoverable data packet loss.

The PGM Reliable Transport Protocol itself is implemented on the sources and the receivers. The source maintains a transmit window of outgoing data packets and retransmits individual packets when it receives a negative acknowledgment (NAK). The network elements (routers) assist in suppressing an implosion of NAKs (when a failure does occur) and aids in efficient forwarding of the retransmitted data just to the networks that need it.

PGM is intended as a solution for multicast applications with basic reliability requirements. The specification for PGM is network layer-independent. The Cisco implementation of PGM Router Assist supports PGM over IP.

Today, the specification for PGM is an Internet draft that can be found on the IETF web site (<http://www.ietf.org>) under the name "PGM Reliable Transport Protocol."

Review Questions

Q—What is the range of available IP multicast addresses?

A—224.0.0.0 to 239.255.255.255.

Q—*What is the purpose of IGMP?*

A—IGMP is used between the hosts and their local multicast router to join and leave multicast groups.

Q—*What is an advantage of IGMPv2 over IGMPv1?*

A—IGMPv2 has a leave group message that can greatly reduce the latency of unwanted traffic on a LAN.

Q—*What is a potential disadvantage of IGMP snooping over CGMP on a low-end Layer 2 switch?*

A—IGMP snooping requires the switch to examine every multicast packet for an IGMP control message. On a low-end switch, this might have a severe performance impact.

Q—*What is an advantage of shortest path (or source) trees compared to shared trees?*

A—Source trees guarantee an optimal path between each source and each receiver, which will minimize network latency.

Q—*What is an advantage of using shared trees?*

A—Shared trees require very little state to be kept in the routers, which requires less memory.

Q—*What information does the router use to do an RPF check?*

A—The unicast routing table.

Q—*Why is protocol-independent multicast called "independent"?*

A—PIM works with any underlying IP unicast routing protocol—RIP, EIGRP, OSPF, BGP or static routes.

Q—*What is the main advantage of MBGP?*

A—Providers can have noncongruent unicast and multicast routing topologies.

Q—*How do RPs learn about sources from other RPs with MSDP?*

A—RPs are configured to be MSDP peers with other RPs. Each RP forwards source active (SA) messages to each other.

Q—*What is the purpose of the anycast RP?*

A—Load balancing and fault tolerance.

For More Information

Williamson, Beau. *Developing IP Multicast Networks*. Indianapolis: Cisco Press, 2000.

Multicast Quick Start Configuration Guide (<http://www.cisco.com/warp/customer/105/48.html>)



macromedia
COLDFUSION 5

Get there faster.

Published on **The O'Reilly Network** (<http://www.oreillynet.com/>)
<http://www.oreillynet.com/pub/a/network/2000/05/12/magazine/gnutella.html>
See this if you're having trouble printing code examples



Gnutella and Freenet Represent True Technological Innovation

by Andy Oram
05/12/2000

The computer technologies that have incurred the most condemnation recently -- Napster, Gnutella, and Freenet -- are also the most interesting from a technological standpoint. I'm not saying this to be perverse. I have examined these systems' architecture and protocols, and I find them to be fascinating. Freenet emerged from a bona fide, academically solid research project, and all three sites are worth serious attention from anyone interested in the future of the Internet.

In writing this essay, I want to take the hype and hysteria out of current reports about Gnutella and Freenet so the Internet community can evaluate them on their merits. This is a largely technical article; I address the policy debates directly in a companion article, [The Value of Gnutella and Freenet](#). I will not cover Napster here because its operation has received more press. It's covered in "[Napster: Popular Program Raises Devilish Issues](#)" by Erik Nilsson, and frankly, it is less interesting and far-reaching technically than the other two systems.

In essence, Gnutella and Freenet represent a new step in distributed information systems. Each is a system for searching for information; each returns information without telling you where it came from. They are innovative in the areas of distributed information storage, information retrieval, and network architecture. But they differ significantly in both goals and implementation, so I'll examine them separately from this point on.

Gnutella basics

Each piece of Gnutella software is both a server and a client in one, because it supports bidirectional information transfer. The Gnutella developers call the software a "servent," but since that term looks odd I'll stick to "client." You can be a fully functional Gnutella site by installing any of several available clients; lots of different operating systems are supported. Next you have to find a few sites that are willing to communicate with you: some may be friends, while others may be advertised Gnutella sites. People with large computers and high bandwidth will encourage many others to connect to them.

You will communicate directly only with the handful of sites you've agreed to contact. Any material of interest to other sites will pass along from one site to another in store-and-forward fashion. Does this sound familiar, all you grizzled, old UUCP and Fidonet users out there? The architecture is essentially the same as those unruly, interconnected systems that succeeded in passing Net News and e-mail around the world for decades before the Internet became popular.

Related Articles:

[Napster and MP3: La Revolucion or La Larceny?](#)

[Music Industry Turns Heat on Net Music Pirates](#)

[Why the RIAA is Fighting a Losing Battle](#)

[Napster: Popular Program Raises Devilish Issues](#)

[Why the RIAA Still Stands a Chance](#)

Best Available Copy

Evil or Just Controversial?:

Open Source software such as Gnutella and Freeware are spreading as quickly as a virus. But are they really so unhealthy? Andy Oram points out the advantages--and

But there are some important differences. Because Gnutella runs over the Internet, you can connect directly with someone who's geographically far away just as easily as with your neighbor. This introduces robustness and makes the system virtually failsafe, as we'll see in a minute.

disadvantages—of controversial technologies in this week's edition of Platform Independent on Web Review.

Second, the protocol for obtaining information over Gnutella is a kind of call-and-response that's more complex than simply pushing news or e-mail. Figure 1 shows the operation of the protocol. Suppose site A asks site B for data matching "MP3." After passing back anything that might be of interest, site B passes the request on to its colleague at site C -- but unlike mail or news, site B keeps a record that site A has made the request. If site C has something matching the request, it gives the information to site B, which remembers that it is meant for site A and passes it through to that site.

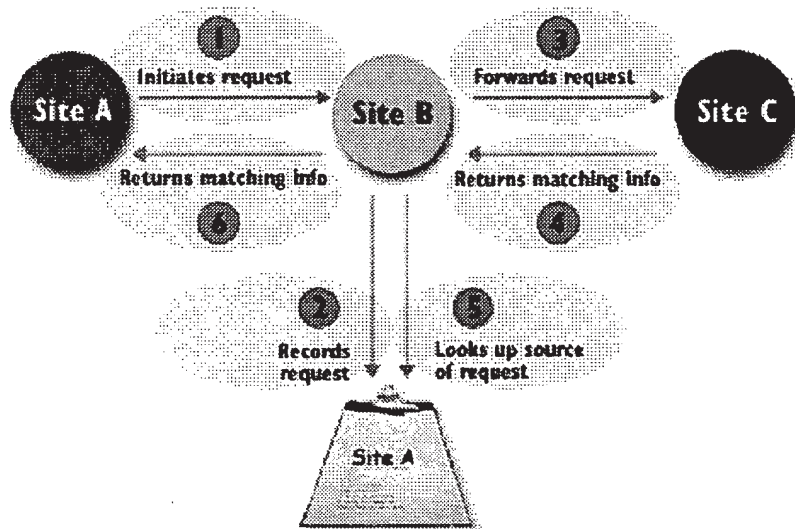


Figure 1. How Gnutella retrieves information

I am tempted to rush on and describe the great significance of this simple system, but I'll pause to answer a few questions for those who are curious.

1. **How are requests kept separate?**

Each request has a unique number, generated from random numbers or semi-randomly from something unique to the originating site like an Ethernet MAC address. If a request goes through site C on to site D and then to site B, site B can recognize from the identifier that it's been seen already and quietly drop the repeat request. On the other hand, different sites can request the same material and have their requests satisfied because each has a unique identifier. Each site lets requests time out, simply by placing them on a queue of a predetermined size and letting old requests drop off the bottom as new ones are added.

2. **What form does the returned data take?**

It could be an entire file of music or other requested material, but Gnutella is not limited to shipping around files. The return could just as well be a URL, or anything else that could be of value. Thus, people are likely to use Gnutella for sophisticated searches, ending up with a URL just as they would with a traditional search engine. (More on this exciting possibility later.)

3. What protocol is used?

Gnutella runs over HTTP (a sign of Gnutella's simplicity). A major advantage of using HTTP is that two sites can communicate even if one is behind a typical organization's firewall, assuming that this firewall allows traffic out to standard Web servers on port 80. There is a slight difficulty if a client behind a firewall is asked to serve up a file, but it can get by the firewall by issuing an output command called GIV to port 80 on its correspondent. The only show-stopper comes when a firewall screens out all Web traffic, or when both correspondents are behind typical firewalls.

4. How does the system stop searching?

Like IP packets, each Gnutella request has a time-to-live, which is normally decremented by each site until it reaches zero. A site can also drastically reduce a time-to-live that it decides is ridiculously high. As we will see in a moment, the time-to-live limits the reach of each site, but that can be a benefit as well as a limitation.

5. How is a search string like "MP3" interpreted?

That is the \$64,000 question, and leads us to Gnutella's greatest contribution.

The Holy Grail: searching for dynamically generated data

Gnutella is a fairly simple protocol. It defines only how a string is passed from one site to another, not how each site interprets the string. One site might handle the string by simply running `fgrep` on a bunch of files, while another might insert it into an SQL query, and yet another might assume that it's a set of Japanese words and return rough English equivalents, which the original requester may then use for further searching. This flexibility allows each site to contribute to a distributed search in the most sophisticated way it can. Would it be pompous to suggest that Gnutella could become the medium through which search engines operate in the 21st century?

Status of Gnutella

Gnutella was started by a division of America Online called Nullsoft. America Online cut off support when it heard about the project, afraid of its potential use for copyright infringement. But a programmer named Brian Mayland reverse engineered the protocol and started a new project to develop clients. None of the developers of current software have looked at code from Nullsoft. Gnutella is an open source project with clients registered under the GNU License.

Limitations and risks of Gnutella

Early experiments with Gnutella suggest it is efficient and useful, but has problems scaling. If you send out a request with a time-to-live of 10, for instance, and each site contacts six other sites, up to 10^6 or 1 million messages could be exchanged.

The exponential spread of requests opens up the most likely source of disruption: denial-of-service attacks caused by flooding the system with requests. The developers have no solution at present, but suggest that clients keep track of the frequency of requests so that they can recognize bursts and refuse further contact with offending nodes.

Furthermore, the time-to-live imposes a horizon on each user. I may repeatedly search a few hundred sites near me, but I will never find files stored a step beyond my horizon. In practice,

information may still get around. After all, Europeans in the Middle Ages enjoyed spices from China even though they knew nothing except the vaguest myths about China. All they had to know was some sites in Asia Minor, who traded with sites in Central Asia, who traded with China.

Spencer Kimball, a developer of the Linux client for Gnutella, says this subnetting can serve to protect Gnutella from attack. Gnutella has already suffered service disruptions, mostly because of bugs in clients, and in the future it is certain to be attacked with vicious and sophisticated attempts to bring it down. While some groups of sites have slowed down temporarily or become severed from other groups, the system has never actually come down.

People may misuse Gnutella for other reasons besides denial of service, of course. One site was recently reported to use it for a sting: The site advertised file names that appeared to offer child pornography, then logged the IP address and domain name of every download request. The reason such information was available is that Gnutella uses HTTP; there is no difference between the user information Gnutella offers and that offered by any Web browser.

A final limitation of Gnutella worth mentioning is the difficulty authenticating the source of the data returned. You really have no idea where the data came from -- but that's true of e-mail and news right now too. Clients don't have to choose anonymity; they can identify themselves as strongly as they want. If a Gnutella client chooses to return a URL, that's just as trustworthy as a URL retrieved in any other manner. If a digital signature infrastructure becomes widespread, clients could use that too. I examine reliability and related policy issues in the article [The Value of Gnutella and Freenet](#).

Freenet basics

The goals of Gnutella and Freenet are very different. Those of Freenet are more explicitly socio-political and, to many people, deliciously subversive:

- To allow people to distribute material anonymously.
- To allow people to retrieve material anonymously.
- To make the removal of material almost insuperably difficult.
- To operate without central control.

The latter feature characterizes both Freenet and Gnutella, and differentiates them from Napster. A court order can shut down Napster (and any mirror site), but shutting down Freenet or Gnutella would be just as hard as prosecuting all those 317,000 Internet users who allegedly exchanged Metallica songs.

Another technical goal of Freenet proves particularly interesting: it spreads data randomly among sites, where the data can appear and disappear unpredictably. In addition to serving the social goals listed above, Freenet offers an intriguing possible solution to the problem of Internet congestion, because popular information automatically propagates to many sites.

Freenet bears no relation to the community networks with similar names of the 1980s and early 1990s. It grew out of a research project launched in 1997 by Ian Clarke at the Division of Informatics at the University of Edinburgh. He has made a [paper](#) from that project available online. (Warning: It's a PDF and I had trouble both viewing and printing it from a couple different PDF viewers.)

The Freenet architecture and protocol is similar to Gnutella in many ways. Each cooperating person downloads a client and sends requests to a few other known clients. Requests are uniquely marked, are handed from one site to another, are temporarily stored on a stack so that data can be returned, and are dropped after each one's time-to-live expires.

The game of find-the-data

The main difference between the two systems is that when a Freenet client satisfies a request, it passes the entire data to the requester. This is an option in Gnutella but is not required. Even more important, as the data passes back along a chain of Freenet clients to the original requester, each client keeps a copy (unless it is a huge amount of data and the client decides that keeping it is not worth the disk space). The client keeps the data so long as other people keep asking for it, but discards the data after some period of time when no one seems to want it.

What is accomplished by this practice, apparently so inefficient compared to the Internet? Ian Clarke tells me it is not all that inefficient -- for large amounts of data its efficiency is comparable to that of the Web -- and that in fact it accomplishes quite a number of things:

- It allows the transience required to meet the goals of anonymity and persistence.
- It lets small sites distribute large, popular documents without suffering bandwidth problems. You don't have run out and get a 16-processor UltraSPARC, or rent space on someone else's, just because you put out an exciting video that lots of people want to download.
- It rewards popular material and allows unpopular material to disappear quietly. In this regard, Freenet is definitely different from the Eternity Service (a model proposed a few years ago but never implemented). Its goal is not to proliferate any kind of garbage people want, but to prevent material from being taken down if a lot of people think it's valuable.
- It tends to bring data close to those who want it. (Like Gnutella, "closeness" in Freenet has no geographical meaning, but refers only to the number of hops between Internet sites.) This is because the first request from node A to node B may have to pass through many other nodes, but the second and subsequent requests can be satisfied by node B directly. Furthermore, nodes A and B are likely to be one or two hops away from many nodes operated by similar people who like the same kinds of content. All those people will be pleased to find the content coming back quickly after the first request is satisfied.

The last item is particularly interesting architecturally, because the popularity of each site's material causes the Freenet system to actually alter its topology. When a site discovers that it is getting a lot of material routinely from one of its partners, it tends to favor that partner for future requests. Bandwidth increases where it benefits the end users. Building on my Europe/China Silk Road analogy, Clarke says, "Freenet is like bringing China closer to Europe as more and more Europeans ask to trade with it."

Other unique features of Freenet

Freenet is more restrained in the traffic generated than Gnutella, perhaps because it expects to transfer a complete file of data for each successful request. When a Freenet client receives a request it cannot satisfy, it sends the request on to a single peer; it does not multicast to all peers as Gnutella does. If the client receives a failure notice because no further systems are known down the line, or if the client fails to get a response because the time-to-live timed out, it tries another one of its peers. In brief, searching is done depth-first and not in parallel. Nevertheless, Clarke says searches are reasonably fast; each takes a couple seconds as with a traditional search

engine. The simple caching system used in Freenet also seems to produce just as good results as the more deliberate caching used by ISPs for Web pages.

Freenet is being developed in Java and requires the Java Runtime Environment to run. It uses its own port and protocol, rather than running over HTTP as Gnutella does.

Limitations and risks of Freenet

Freenet seems more scalable than Gnutella. One would imagine that it could be impaired by flooding with irrelevant material (writing a script that dumped the contents of your 8-gig disk into it once every hour, for instance) but that kind of attack actually has little impact. So long as nobody asks for material, it doesn't go anywhere.

Furthermore, once someone puts up material, no one can overwrite it with a bogus replacement. Each item is identified by a unique identifier. If a malicious censor tries to put up his own material with the same identifier, the system checks for an existing version and says, "We already have the material!" The only effect is to make the original material stay up longer, because a request for it was made by the would-be censor.

The searching problem

The unique identifier is Freenet's current weak point. Although someone posting material can assign any string as an identifier, Freenet chooses for security reasons to hash the string. Two search strings that differ by a single character (like "HumanRights" and "Human-Rights") will hash to very different values, as with any hashing algorithm. This means that a prosecuting agency that is trying to locate offending material will have great difficulty identifying the material from a casual scan of each site.

But the hashing also renders Freenet unusable for random searches. If you know exactly what you're looking for on Freenet -- because someone has used another channel to say, for instance, "Search for HumanRights on Freenet" -- you can achieve success. But you can't do free text searches.

One intriguing use of Freenet is to offer sites with hyperlinks. Take people interested in bird-watching as an example. Just as an avid aviarist can offer a Web page with links to all kinds of Web sites, she can offer links that generate Freenet requests using known strings that retrieve data about birds over Freenet. Already, for people who want to try out Freenet without installing the client, a gateway to the Web exists under the name `fproxy`.

Another area for research is a client that accepts a string and changes it slightly in the hope of producing a more accurate string, then passes it on. The most important task in the Freenet project currently, according to Clarke, is to resolve the search problem.

Letting go

Once again, I refer readers to [The Value of Gnutella and Freenet](#) for a discussion of these systems' policy and social implications. I'll end this technical article by suggesting that the Gnutella and Freenet continue to loosen the virtual from the physical, a theme that characterizes network evolution. DNS decoupled names from physical systems; URNs will allow users to retrieve documents without domain names; virtual hosting and replicated servers change the one-to-one relationship of names to systems. Perhaps it is time for another major conceptual leap, where we let go of the notion of location. Welcome to the Heisenberg Principle, as applied to the Internet. Information just became free.

Gnutella and Freenet, in different ways, make the location of documents irrelevant; the search

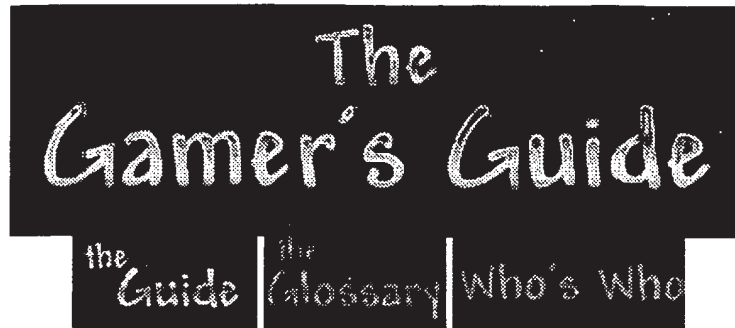
string becomes the location. To achieve this goal, they add a new layer of routing on top of the familiar routing done at the IP level. The new layer may appear at first to introduce numerous problems in efficiency and scaling, but in practice these turn out to be negligible or least tolerable. I think readers should take a close look at these systems; even if Gnutella and Freenet themselves turn out not to be good enough solutions for a new Internet era, they'll teach us some lessons when it's time for yet another leap.

Andy Oram is an editor at O'Reilly & Associates specializing in books on Linux and programming. Most recently, he edited Peer-to-Peer: Harnessing the Power of Disruptive Technologies.

Discuss this article in the O'Reilly Network [General Forum](#).

Return to the [O'Reilly Network Hub](#).

oreillynet.com Copyright © 2000 O'Reilly & Associates, Inc.



Best Available Copy

First-Person Shooters

General

There are many different varieties of computer games on the market today. From flight simulators to real-time strategy, all of these varieties have created niche markets within the huge population of computer gamers. One of the fastest growing game-types is the First-Person Shooter. This subcategory of action games, also called over-the-shoulder shooters and other names, is based upon a relatively simple premise: You see the world through the eyes of your character. The goal of such games also tend to be very simple: You run down darkened hallways and kill anything that gets in your way. However, reducing the genre to simplistic elements fails to appreciate the reasons that these games are so popular. First-person shooters have lead the industry in implementing the latest technology, providing the greatest support for multi-player gaming, and generating the greatest amount of praise and condemnation.

History

The birth of the first-person shooter can be traced back to id Software's Wolfenstein 3-D, released in shareware on May 5, 1992. Using a very strong graphics engine developed in-house, this game allowed players to see an amazing landscape through the eyes of the main character. The graphics may appear dated now after less than a decade, but the ability to move along the x and y axis in a graphically rendered hallway was a revelation. So was the feeling that the perspective gave the game. Players who were used to seeing small, animated characters jump up and down in side-scrolling games were amazed at the immersive feel of Wolfenstein. You could see the weapon that you were carrying and could see it fire. Life-like Nazis and vicious dogs attacked you from every corner.

If Wolfenstein is the father of the first-person shooter, then Doom would be his most popular descendent. Doom, also by id Software, moved the action from the darkened hallways of a German castle during World War II to futuristic, although still darkened, hallways with a score of alien monsters. The game was an immediate success. This can be attributed to both the technical and aesthetic improvements in the game as well as the timing of its release. Doom, released on December 10, 1993, entered the scene just as modem speeds and awareness of the internet were increasing and the corporate and academic worlds were embracing the Local Area Network (LAN). Its ability to support convincing multi-player games catapulted it into the forefront of the gaming community.

Other software companies were quick to recognize the appeal of Doom and the first-person shooters. Many licensed the graphics engine from id and began creating their own games. These "Doom clones" enjoyed great success. Games such as Rise of the Triad and LucasArt's Dark Forces created worlds that rivaled the world of Doom. This competition encouraged companies to find new ways to exploit the Doom engine. One result was beautifully rendered games with innovative ideas, like Raven Software's Heretic. The other result was increasingly graphic violence and a move towards parental ratings. Apogee, and its subsidiary 3D Realms, have ironically been at the forefront of both controversial content and moves to inform parents of the content.

Doom was followed by the highly successful Doom II. This game is viewed by many as the ultimate development of the Doom engine. However, the game still lacked true 3D. id Software then announced the development of Quake, a true 3D game. Development of Quake lasted 18 months, giving competitors an opportunity to develop competing technology. 3D Realms took this opportunity. Using the Build graphics engine, created by the amazing Ken Silverman, 3D Realms developed Duke Nukem 3D. This game wasn't really a 3D game, but the graphics engine was versatile enough to make the illusion of 3D almost seamless. Duke was released several months before Quake and quickly garnered a huge following. Many of the innovations contained in this game are currently being used by other software companies.

The release of Quake marked the true beginning of the 3D age of first-person shooters, at least in the traditional sense that we use the terms 3D and first-person shooter. Descent was the first title to actually use 3D successfully, with impressive and popular results, but Quake was the first title that used 3D in the traditional first-person shooter. With its superb graphics engine, quality levels and monsters, and multi-player support for 16 players, Quake allowed id Software to regain its crown as the king of the first-person shooter. Amazingly, Quake remained the technology leader for a year and a half. This has as much to do with the appeal of the game as with id's decision to release enough tools and information to allow users to modify Quake. Using a language called Quake C, programmers could alter virtually every element of the game. The creation of powerful level editors and a host of utilities, all by the growing "Quake community," paved the way for spectacular new games, new levels, and new ideas.

LucasArts' Jedi Knight was a revolutionary game on many levels. Until this title was released in mid 1997, it was laughable to talk about the "storyline" in a first-shooter. Jedi Knight took advantage of the rich Star Wars history to create a unique and entertaining game. Through the use of cutscenes and a non-linear story, LucasArts provided a game that compelled a player to finish.

The release of Quake II in December 1997 pushed the technology envelop even further. Many of the features that had been pioneered during Quake's ongoing development, such as OpenGL support, transparent water, and the client prediction of QuakeWorld, were included in this new game. Quake II also added nifty visual effects, like 16 bit textures in GL mode and colored lighting, but was also the first id title to have a strong storyline. Granted, the storyline still involved you running around killing everything, but the player was provided with specific tasks for each level.

Just as Duke Nukem 3D and Jedi Knight had done previously, the release of Unreal has had a profound impact on a community that had been entirely focused upon titles from id. Unreal's strength lies not in its storyline, which tended to disappear in

the middle of the game, but in the power of its graphics engine. The engine adds many visual improvements, allowed for incredibly large levels, and is still very easy to modify. A wide range of titles, from hunting games to adventure games, are currently being developed with the Unreal engine. Many have seen this as a real threat to the empire of id.

Fans of the Quake series are looking forward to the release of Quake III Arena. This game has gone through a few profound changes since it was first proposed, but the many new visual effects and gameplay elements that have been proposed for the game are very impressive.

1999 will undoubtedly see a continued evolution of the first-person shooter. Several companies are currently developing games utilizing the graphics engine from id Software's Quake 2. Several other companies are using the popular engine from Unreal to design their games. Still others have created, or are creating, their own graphics engine. It is impossible to determine how successful these efforts will be in an increasingly crowded field. Regardless, it is clear that the first-person shooter will remain a large and powerful genre in computer gaming.

Technology

As a group, first-person shooters tend to be driving forces in accelerating the development of computer software and hardware. Other types of games are now trying to keep up, but FPSs have a clear lead in the technology race. The rise of the 3D graphics accelerator cannot be accredited to business applications; it is rooted firmly in computer games. It could be argued that 3Dfx, the company that created the Voodoo chipset used by several graphics cards, owes much of its success to GL Quake.

First-person shooters have also pioneered the mutli-player gaming experience. Artificial intelligence with games continue to increase, but many players find monsters a poor substitute for playing against real people.

Controversy

Just as FPSs lead the way in technology, they also have cornered the market in controversy. Very few games contain as much graphic violence as a first-person shooter. Many first-person shooters have also received criticism for their sexist portrayal of women, satanic overtones, and racial stereotypes. Many voices have been raised in defense, and in condemnation, of these controversial games. The course of first-person shooters in this area remains uncertain. id Software removed all Satanic references from Quake 2, but the program was banned for sale in Germany for its violent content. 3D Realms created a "parental lock" for Duke Nukem 3D, an effort to allow parents to monitor a child's use of the game, but their is little doubt that the strippers and prostitutes will return for Duke Nukem Forever. Some software companies appear to be concerned about reducing controversy. Others appear to go out of their way to generate controversy. This division will likely remain as long segments of the gaming community are drawn towards each approach. In the end, the actual playability of a game will generate more interest, and more sales, than controversy alone.

[Basics] [Servers] [Beyond basics] [Other versions of Quake] [Sources of information] [Citizenship] [Quake C] [Level editing] [Mods] [Quake derivatives] [Operating Systems] [Guide News]

Last update: October 20, 1998
This page is maintained by Darren L. Tabor,
aka Dakota



Computer Science Open Days

Demonstration on the Problems of Distributed Systems

Introduction

This demonstration is intended to be representative of the material studied by our students in modules on programming and software engineering. As such we hope to stimulate your curiosity in some of the many fascinating areas and challenging problems facing today's software engineers. Of particular interest at the moment is the subject of *distributed systems* where we want one computer to pass information to another. The internet is a topical example of a global distributed system where, for example, I might want to send an e-mail I've just written on my computer to your computer where you can read it. Or perhaps, by using your world wide web browser, you might want to download from America a document to your own computer. The methods used to transfer data from one computer to another are usually hidden from users, however, there are many fascinating challenges facing the software engineer who has to ensure that information is sent in the fastest and most efficient way possible.

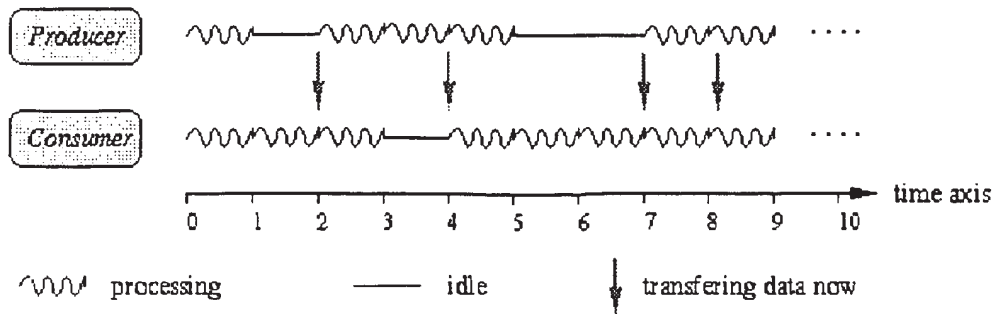
In today's demonstration we consider a typical problem of how to transfer data, one which regularly faces our second year software engineering students.

The Producer-Consumer Problem

Let's suppose we have two computers labelled *Producer* and *Consumer*. Producer spends its time grinding away processing its data, but every now and again it wishes to send the fruits of its labour to Consumer. Meanwhile Consumer grinds away processing data received from Producer.



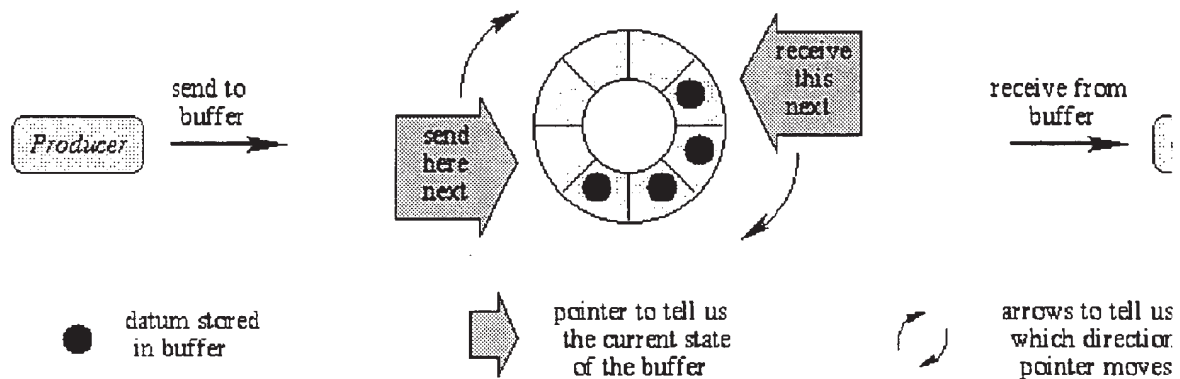
So, what is the problem here ? Producer sends data to Consumer, so what ? Well, Producer can only send each datum when Consumer is ready to receive it, and so the former has to wait doing nothing until the latter is ready to receive. Similarly, Consumer may have to wait for Producer to send a datum before it can continue with its own processing. All in all, as the following diagram shows, both Producer and Consumer may waste as much time do nothing as they actually spend processing.



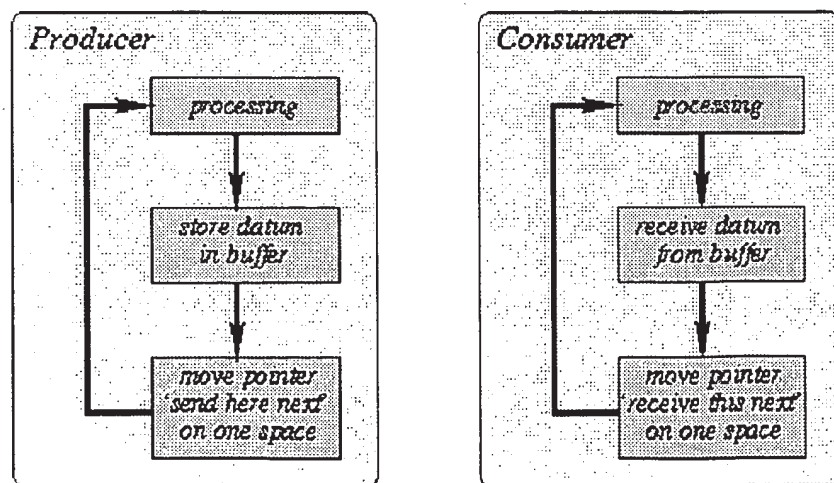
And so our problem is how to transfer data such that Producer and Consumer are never idle unnecessarily.

First pass at solving the problem

Any solution to this problem must first remove the obstacle of one computer having to wait for the other before it can send/receive. We do this by introducing a temporary storage area termed a *buffer* where Producer can send it's data until such time as Consumer is ready to receive it.



Now Producer and Consumer can each follow their own algorithm (i.e. sequence of steps) as follows, and quite independently of each other.

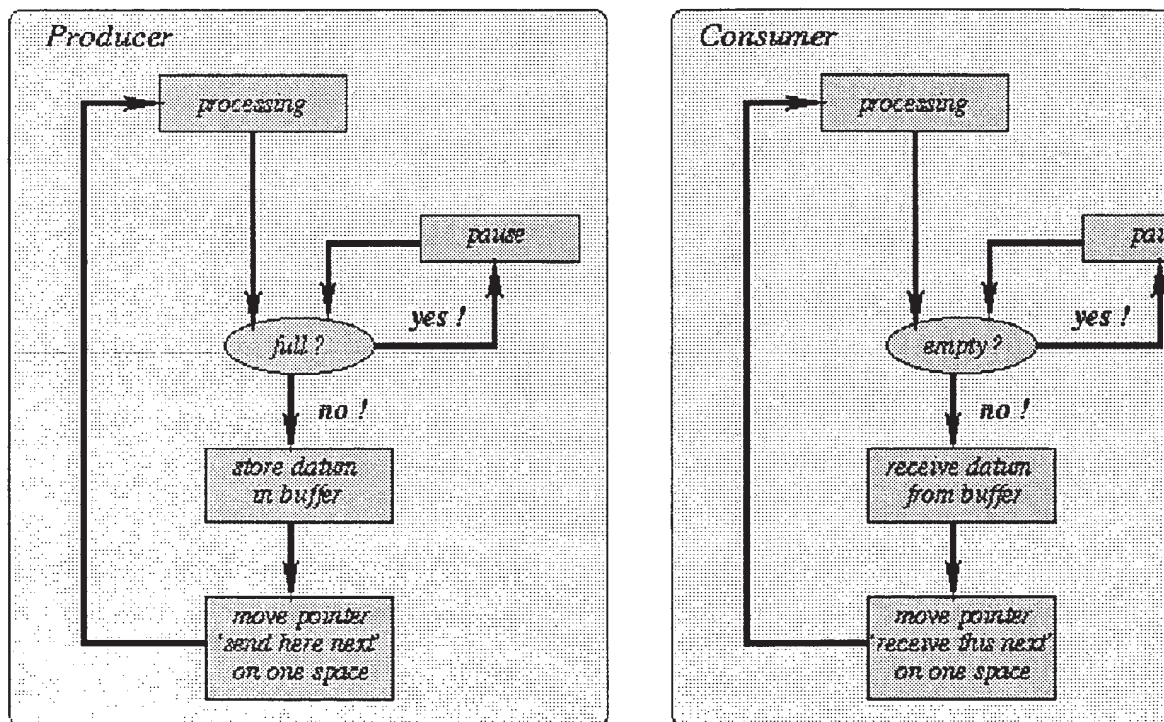


Note how we make good use of the buffer's spaces by recycling each one once its contents have

been received by Consumer. Well, I guess that might seem like the end of the story. Unfortunately the fact that Producer and Consumer can now operate independently of each other raises a difficulty often found in distributed systems. If two computers operate independently could they accidentally step on each other's toes ?

Second pass at solving the problem

One potential problem with our solution so far is that Producer might try to send a datum when the buffer is already full of data still waiting to be received by Consumer. Similarly, Consumer might be trying to receive data from an empty buffer, that is, when all the data sent so far by Producer has already been received. We have to find a way to prevent Producer sending to a full buffer, and of Consumer receiving from an empty buffer. We do this by insisting that Producer (resp. Consumer) pause for a moment whenever the buffer is full (resp. empty), which will hopefully give time for Consumer (resp. Producer) a chance to make space (resp. make a deposit) in the buffer. By adding this extra refinement to the algorithms for Producer and Consumer we get the following.



Final pass at solving the problem

But, how do we know when the buffer is full, and when it is empty. This we really do have to know in order to prevent the buffer from becoming corrupted. A simple answer would be to keep a tally of the number of unread data items currently in the buffer. Starting at zero, the tally is incremented by one every time a send is made, and decremented by one every time a receive is made. Whenever this number is zero the buffer must be empty, and so receiving is not allowed. When the number equals the overall number of spaces in the buffer then the latter must be full, and so sending is not allowed.

And finally, ...

This completes the design of our solution, and so it just remains to code it up in an appropriate programming language. If you want to see how that's done come to Warwick and be student in

http://www.dcs.warwick.ac.u...
computer science.

01/29/2002--page 4

DISTRIBUTED GAME ENVIRONMENT

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. Patent Application No. _____, entitled "BROADCASTING NETWORK," filed on July 31, 2000 (Attorney Docket No. 030048001 US); U.S. Patent Application No. _____, entitled "JOINING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048002 US); U.S. Patent Application No. _____, "LEAVING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048003 US); U.S. Patent Application No. _____, entitled "BROADCASTING ON A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048004 US); U.S. Patent Application No. _____, entitled "CONTACTING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048005 US); U.S. Patent Application No. _____, entitled "DISTRIBUTED AUCTION SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048006 US); U.S. Patent Application No. _____, entitled "AN INFORMATION DELIVERY SERVICE," filed on July 31, 2000 (Attorney Docket No. 030048007 US); U.S. Patent Application No. _____, entitled "DISTRIBUTED CONFERENCING SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048008 US); and U.S. Patent Application No. _____, entitled "DISTRIBUTED GAME ENVIRONMENT," filed on July 31, 2000 (Attorney Docket No. 030048009 US), the disclosures of which are incorporated herein by reference.

TECHNICAL FIELD

The described technology relates generally to a computer network and more particularly, to a broadcast channel for a subset of a computers of an underlying network.

25 BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network

[03004-8001/SL003733.107]

8009

-1-

7/31/00

protocols, and peer-to-peer middleware. Each of these communications techniques have their advantages and disadvantages, but none is particularly well suited to the simultaneous sharing of information among computers that are widely distributed. For example, collaborative processing applications, such as a network meeting programs, have a need to
5 distribute information in a timely manner to all participants who may be geographically distributed.

The point-to-point network protocols, such as UNIX pipes, TCP/IP, and UDP, allow processes on different computers to communicate via point-to-point connections. The interconnection of all participants using point-to-point connections, while theoretically
10 possible, does not scale well as a number of participants grows. For example, each participating process would need to manage its direct connections to all other participating processes. Programmers, however, find it very difficult to manage single connections, and management of multiple connections is much more complex. In addition, participating processes may be limited to the number of direct connections that they can support. This
15 limits the number of possible participants in the sharing of information.

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers,
20 and the common object request broker architecture ("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each
25 client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (*i.e.*,
30 the server) would prevent communications between any of the clients.

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network

protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (e.g., routers) to support the sharing of information efficiently.

5 The peer-to-peer middleware communications systems rely on a multicasting network protocol or a graph of point-to-point network protocols. Such peer-to-peer middleware is provided by the T.120 Internet standard, which is used in such products as Data Connection's D.C.-share and Microsoft's NetMeeting. These peer-to-peer middleware systems rely upon a user to assemble a point-to-point graph of the connections used for
10 sharing the information. Thus, it is neither suitable nor desirable to use peer-to-peer middleware systems when more than a small number of participants is desired. In addition, the underlying architecture of the T.120 Internet standard is a tree structure, which relies on the root node of the tree for reliability of the entire network. That is, each message must pass through the root node in order to be received by all participants.

15 It would be desirable to have a reliable communications network that is suitable for the simultaneous sharing of information among a large number of the processes that are widely distributed.

BRIEF DESCRIPTION OF THE DRAWINGS

20 Figure 1 illustrates a graph that is 4-regular and 4-connected which represents a broadcast channel.

 Figure 2 illustrates a graph representing 20 computers connected to a broadcast channel.

 Figures 3A and 3B illustrate the process of connecting a new computer Z to the broadcast channel.

25 Figure 4A illustrates the broadcast channel of Figure 1 with an added computer.

 Figure 4B illustrates the broadcast channel of Figure 4A with an added computer.

30 Figure 4C also illustrates the broadcast channel of Figure 4A with an added computer.

Figure 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner.

Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner.

5 Figure 5C illustrates the neighbors with empty ports condition.

Figure 5D illustrates two computers that are not neighbors who now have empty ports.

Figure 5E illustrates the neighbors with empty ports condition in the small regime.

10 Figure 5F illustrates the situation of Figure 5E when in the large regime.

Figure 6 is a block diagram illustrating components of a computer that is connected to a broadcast channel.

Figure 7 is a block diagram illustrating the sub-components of the broadcaster component in one embodiment.

15 Figure 8 is a flow diagram illustrating the processing of the connect routine in one embodiment.

Figure 9 is a flow diagram illustrating the processing of the seek portal computer routine in one embodiment.

20 Figure 10 is a flow diagram illustrating the processing of the contact process routine in one embodiment.

Figure 11 is a flow diagram illustrating the processing of the connect request routine in one embodiment.

Figure 12 is a flow diagram of the processing of the check for external call routine in one embodiment.

25 Figure 13 is a flow diagram of the processing of the achieve connection routine in one embodiment.

Figure 14 is a flow diagram illustrating the processing of the external dispatcher routine in one embodiment.

30 Figure 15 is a flow diagram illustrating the processing of the handle seeking connection call routine in one embodiment.

Figure 16 is a flow diagram illustrating processing of the handle connection request call routine in one embodiment.

Figure 17 is a flow diagram illustrating the processing of the add neighbor routine in one embodiment.

Figure 18 is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment.

5 Figure 19 is a flow diagram illustrating the processing of the handle edge proposal call routine.

Figure 20 is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment.

10 Figure 21 is a flow diagram illustrating the processing of the fill hole routine in one embodiment.

Figure 22 is a flow diagram illustrating the processing of the internal dispatcher routine in one embodiment.

Figure 23 is a flow diagram illustrating the processing of the handle broadcast message routine in one embodiment.

15 Figure 24 is a flow diagram illustrating the processing of the distribute broadcast message routine in one embodiment.

Figure 26 is a flow diagram illustrating the processing of the handle connection port search statement routine in one embodiment.

20 Figure 27 is a flow diagram illustrating the processing of the court neighbor routine in one embodiment.

Figure 28 is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment.

Figure 29 is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment.

25 Figure 30 is a flow diagram illustrating the processing of the broadcast routine in one embodiment.

Figure 31 is a flow diagram illustrating the processing of the acquire message routine in one embodiment.

30 Figure 32 is a flow diagram illustrating processing of the handle condition check message in one embodiment.

Figure 33 is a flow diagram illustrating processing of the handle condition repair statement routine in one embodiment.

Figure 34 is a flow diagram illustrating the processing of the handle condition double check routine.

DETAILED DESCRIPTION

A broadcast technique in which a broadcast channel overlays a point-to-point communications network is provided. The broadcasting of a message over the broadcast channel is effectively a multicast to those computers of the network that are currently connected to the broadcast channel. In one embodiment, the broadcast technique provides a logical broadcast channel to which host computers through their executing processes can be connected. Each computer that is connected to the broadcast channel can broadcast messages onto and receive messages off of the broadcast channel. Each computer that is connected to the broadcast channel receives all messages that are broadcast while it is connected. The logical broadcast channel is implemented using an underlying network system (*e.g.*, the Internet) that allows each computer connected to the underlying network system to send messages to each other connected computer using each computer's address. Thus, the broadcast technique effectively provides a broadcast channel using an underlying network system that sends messages on a point-to-point basis.

The broadcast technique overlays the underlying network system with a graph of point-to-point connections (*i.e.*, edges) between host computers (*i.e.*, nodes) through which the broadcast channel is implemented. In one embodiment, each computer is connected to four other computers, referred to as neighbors. (Actually, a process executing on a computer is connected to four other processes executing on this or four other computers.) To broadcast a message, the originating computer sends the message to each of its neighbors using its point-to-point connections. Each computer that receives the message then sends the message to its three other neighbors using the point-to-point connections. In this way, the message is propagated to each computer using the underlying network to effect the broadcasting of the message to each computer over a logical broadcast channel. A graph in which each node is connected to four other nodes is referred to as a 4-regular graph. The use of a 4-regular graph means that a computer would become disconnected from the broadcast channel only if all four of the connections to its neighbors fail. The graph used by the broadcast technique also has the property that it would take a failure of four computers to

divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

Figure 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (*i.e.*, the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from computer F to computer B. The maximum of the distances between the computers is the "diameter" of broadcast channel. The diameter of the broadcast channel represented by Figure 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. Figure 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1-12, 12-15, 15-18, and 18-3).

The broadcast technique includes (1) the connecting of computers to the broadcast channel (*i.e.*, composing the graph), (2) the broadcasting of messages over the broadcast channel (*i.e.*, broadcasting through the graph), and (3) the disconnecting of computers from the broadcast channel (*i.e.*, decomposing the graph) composing the graph.

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then

establishes a connection with four of the computers that are already connected to the broadcast channel. (This assumes that there are at least four computers already connected to the broadcast channel. When there are fewer than five computers connected, the broadcast channel cannot be a 4-regular graph. In such a case, the broadcast channel is considered to be in a "small regime." The broadcast technique for the small regime is described below in detail. When five or more computers are connected, the broadcast channel is considered to be in the "large regime." This description assumes that the broadcast channel is in the large regime, unless specified otherwise.) Thus, the process of connecting to the broadcast channel includes locating the broadcast channel, identifying the neighbors for the connecting computer, and then connecting to each identified neighbor. Each computer is aware of one or more "portal computers" through which that computer may locate the broadcast channel. A seeking computer locates the broadcast channel by contacting the portal computers until it finds one that is currently fully connected to the broadcast channel. The found portal computer then directs the identifying of four computers (*i.e.*, to be the seeking computer's neighbors) to which the seeking computer is to connect. Each of these four computers then cooperates with the seeking computer to effect the connecting of the seeking computer to the broadcast channel. A computer that has started the process of locating a portal computer, but does not yet have a neighbor, is in the "seeking connection state." A computer that is connected to at least one neighbor, but not yet four neighbors, is in the "partially connected state." A computer that is currently, or has been, previously connected to four neighbors is in the "fully connected state."

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. Figures 3A and 3B illustrate the process of a new computer Z connecting to the broadcast channel. Figure 3A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E

is established as indicated by Figure 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

5 Each computer connected to the broadcast channel allocates five communications ports for communicating with other computers. Four of the ports are referred to as "internal" ports because they are the ports through which the messages of the broadcast channels are sent. The connections between internal ports of neighbors are referred to as "internal" connections. Thus, the internal connections of the broadcast channel
10 form the 4-regular and 4-connected graph. The fifth port is referred to as an "external" port because it is used for sending non-broadcast messages between two computers. Neighbors can send non-broadcast messages either through their internal ports of their connection or through their external ports. A seeking computer uses external ports when locating a portal computer.

15 In one embodiment, the broadcast technique establishes the computer connections using the TCP/IP communications protocol, which is a point-to-point protocol, as the underlying network. The TCP/IP protocol provides for reliable and ordered delivery of messages between computers. The TCP/IP protocol provides each computer with a "port space" that is shared among all the processes that may execute on that computer. The ports
20 are identified by numbers from 0 to 65,535. The first 2056 ports are reserved for specific applications (*e.g.*, port 80 for HTTP messages). The remainder of the ports are user ports that are available to any process. In one embodiment, a set of port numbers can be reserved for use by the computer connected to the broadcast channel. In an alternative embodiment, the port numbers used are dynamically identified by each computer. Each computer
25 dynamically identifies an available port to be used as its call-in port. This call-in port is used to establish connections with the external port and the internal ports. Each computer that is connected to the broadcast channel can receive non-broadcast messages through its external port. A seeking computer tries "dialing" the port numbers of the portal computers until a portal computer "answers," a call on its call-in port. A portal computer answers when it is
30 connected to or attempting to connect to the broadcast channel and its call-in port is dialed. (In this description, a telephone metaphor is used to describe the connections.) When a computer receives a call on its call-in port, it transfers the call to another port. Thus, the

seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the diameter of the broadcast channel may increase when each seeking computer uses the same found portal computer and establishes a connection to the broadcast channel directly through that found portal computer. Conceptually, the graph becomes elongated in the direction of where the new nodes are added. Figures 4A-4C illustrate that possible problem. Figure 4A illustrates the broadcast channel of Figure 1 with an added computer. Computer J was connected to the broadcast channel by edge pinning edges C-D and E-H to computer J. The diameter of this broadcast channel is still two. Figure 4B illustrates the broadcast channel of Figure 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges E-J and B-C to computer K. The diameter of this broadcast channel is three, because the shortest path from computer G to computer K is through edges G-A, A-E, and E-K. Figure 4C also illustrates the broadcast channel of Figure 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges D-G and E-J to computer K. The diameter of this broadcast channel is, however, still two. Thus, the selection of neighbors impacts the diameter of the broadcast channel. To help minimize the diameter, the broadcast technique uses a random selection technique to identify the four neighbors of a computer in the seeking connection state. The random selection technique tends to distribute the connections to new seeking computers throughout the computers of the broadcast channel which may result in smaller overall diameters.

Broadcasting Through the Graph

As described above, each computer that is connected to the broadcast channel can broadcast messages onto the broadcast channel and does receive all messages that are broadcast on the broadcast channel. The computer that originates a message to be broadcast sends that message to each of its four neighbors using the internal connections. When a
5 computer receives a broadcast message from a neighbor, it sends the message to its three other neighbors. Each computer on the broadcast channel, except the originating computer, will thus receive a copy of each broadcast message from each of its four neighbors. Each computer, however, only sends the first copy of the message that it receives to its neighbors and disregards subsequently received copies. Thus, the total number of copies of a message
10 that is sent between the computers is $3N+1$, where N is the number of computers connected to the broadcast channel. Each computer sends three copies of the message, except for the originating computer, which sends four copies of the message.

The redundancy of the message sending helps to ensure the overall reliability
15 of the broadcast channel. Since each computer has four connections to the broadcast channel, if one computer fails during the broadcast of a message, its neighbors have three other connections through which they will receive copies of the broadcast message. Also, if the internal connection between two computers is slow, each computer has three other connections through which it may receive a copy of each message sooner.

Each computer that originates a message numbers its own messages
20 sequentially. Because of the dynamic nature of the broadcast channel and because there are many possible connection paths between computers, the messages may be received out of order. For example, the distance between an originating computer and a certain receiving computer may be four. After sending the first message, the originating computer and
25 receiving computer may become neighbors and thus the distance between them changes to one. The first message may have to travel a distance of four to reach the receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

When the broadcast channel is in a steady state (*i.e.*, no computers connecting
30 or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a

steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

One solution to this problem is to have each computer queue all the messages that it receives until it can send them in their proper order to its neighbors. This solution, however, may tend to slow down the propagation of messages through the computers of the broadcast channel. Another solution that may have less impact on the propagation speed is to queue messages only at computers who are neighbors of the newly connected computers. Each already connected neighbor would forward messages as it receives them to its other neighbors who are not newly connected, but not to the newly connected neighbor. The already connected neighbor would only forward messages from each originating computer to the newly connected computer when it can ensure that no gaps in the messages from that originating computer will occur. In one embodiment, the already connected neighbor may track the highest sequence number of the messages already received and forwarded on from each originating computer. The already connected computer will send only higher numbered messages from the originating computers to the newly connected computer. Once all lower numbered messages have been received from all originating computers, then the already connected computer can treat the newly connected computer as its other neighbors and simply forward each message as it is received. In another embodiment, each computer may queue messages and only forwards to the newly connected computer those messages as the gaps are filled in. For example, a computer might receive messages 4 and 5 and then receive message 3. In such a case, the already connected computer would forward queue messages 4 and 5. When message 3 is finally received, the already connected computer will send messages 3, 4, and 5 to the newly connected computer. If messages 4 and 5 were sent to the newly connected computer before message 3, then the newly connected computer would process messages 4 and 5 and disregard message 3. Because the already connected computer queues messages 4 and 5, the newly connected computer will be able to process message 3. It is possible that a newly connected computer will receive a set of messages from an originating computer through one neighbor and then receive another set of message from the

same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

5 Decomposing the Graph

 A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a computer disconnects in a planned manner, it sends a disconnect message to each of its four neighbors. The disconnect message includes a list that identifies the four neighbors of the disconnecting computer. When a neighbor receives the disconnect message, it tries to connect to one of the computers on the list. In one embodiment, the first computer in the list will try to connect to the second computer in the list, and the third computer in the list will try to connect to the fourth computer in the list. If a computer cannot connect (e.g., the first and second computers are already connected), then the computers may try connecting in various other combinations. If connections cannot be established, each computer broadcasts a message that it needs to establish a connection with another computer. When a computer with an available internal port receives the message, it can then establish a connection with the computer that broadcast the message. Figures 5A-5D illustrate the disconnecting of a computer from the broadcast channel. Figure 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner. When computer H decides to disconnect, it sends its list of neighbors to each of its neighbors (computers A, E, F and I) and then disconnects from each of its neighbors. When computers A and I receive the message they establish a connection between them as indicated by the dashed line, and similarly for computers E and F.

 When a computer disconnects in an unplanned manner, such as resulting from a power failure, the neighbors connected to the disconnected computer recognize the disconnection when each attempts to send its next message to the now disconnected computer. Each former neighbor of the disconnected computer recognizes that it is short one connection (i.e., it has a hole or empty port). When a connected computer detects that one of its neighbors is now disconnected, it broadcasts a port connection request on the broadcast channel, which indicates that it has one internal port that needs a connection. The port connection request identifies the call-in port of the requesting computer. When a connected

computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to

each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

It is possible that the two original neighbors with the condition may have the same set of neighbors. When the neighbor that receives the condition check message determines that the sets of neighbors are the same, it sends a condition double check message to one of its neighbors other than the neighbor who also has the condition. When the computer receives the condition double check message, it determines whether it has the same set of neighbors as the sending computer. If so, the broadcast channel is in the small regime and the condition is not a problem. If the set of neighbors are different, then the computer that received the condition double check message sends a condition check message to the original neighbors with the condition. The computer that receives that condition check message directs one of its neighbors to connect to one of the original neighbors with the condition by sending a condition repair message. Thus, one of the original neighbors with the condition will have its port filled.

Figure 5C illustrates the neighbors with empty ports condition. In this illustration, computer H disconnected in an unplanned manner, but computers F and I responded to the port connection request of the other and are now connected together. The other former neighbors of computer H, computers A and E, are already neighbors, which gives rise to the neighbors with empty ports condition. In this example, computer E received the port connection request from computer A, recognized the possible condition, and sent (since they are neighbors via the internal connection) a condition check message with a list of its neighbors to computer A. When computer A received the list, it recognized that computer E has a different set of neighbor (*i.e.*, the broadcast channel is in the large regime). Computer A selected computer D, which is a neighbor of computer E and sent it a condition repair request. When computer D received the condition repair request, it disconnected from one of its neighbors (other than computer E), which is computer G in this example. Computer D then connected to computer A. Figure 5D illustrates two computers that are not neighbors who now have empty ports. Computers E and G now have empty ports and are not currently neighbors. Therefore, computers E and G can connect to each other.

Figures 5E and 5F further illustrate the neighbors with empty ports condition. Figure 5E illustrates the neighbors with empty ports condition in the small regime. In this

example, if computer E disconnected in an unplanned manner, then each computer broadcasts a port connection request when it detects the disconnect. When computer A receives the port connection request from computer B, it detects the neighbors with empty ports condition and sends a condition check message to computer B. Computer B recognizes that it has the same set of neighbors (computer C and D) as computer A and then sends a condition double check message to computer C. Computer C recognizes that the broadcast channel is in the small regime because it also has the same set of neighbors as computers A and B, computer C may then broadcast a message indicating that the broadcast channel is in the small regime.

Figure 5F illustrates the situation of Figure 5E when in the large regime. As discussed above, computer C receives the condition double check message from computer B. In this case, computer C recognizes that the broadcast channel is in the large regime because it has a set of neighbors that is different from computer B. The edges extending up from computer C and D indicate connections to other computers. Computer C then sends a condition check message to computer B. When computer B receives the condition check message, it sends a condition repair message to one of the neighbors of computer C. The computer that receives the condition repair message disconnects from one of its neighbors, other than computer C, and tries to connect to computer B and the neighbor from which it disconnected tries to connect to computer A.

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is

connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

If many computers are at the same time seeking connection to a broadcast channel through a single portal computer, then the ports of the portal computer may be busy when called by seeking computers. The seeking computers would typically need to keep on redialing a busy port. The process of locating a call-in port may be significantly slowed by such redialing. In one embodiment, each seeking computer may each reorder the first few port numbers generated by the hashing algorithm. For example, each seeking computer could randomly reorder the first eight port numbers generated by the hashing algorithm. The random ordering could also be weighted where the first port number generated by the hashing algorithm would have a 50% chance of being first in the reordering, the second port number would have a 25% chance of being first in the reordering, and so on. Because the seeking computers would use different orderings, the likelihood of finding a busy port is reduced. For example, if the first eight port numbers are randomly selected, then it is

possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

Each computer that can connect to the broadcast channel has a list of one or more portal computers through which it can connect to the broadcast channel. In one embodiment, each computer has the same set of portal computers. A seeking computer locates a portal computer that is connected to the broadcast channel by successively dialing the ports of each portal computer in the order specified by an algorithm. A seeking computer could select the first portal computer and then dial all its ports until a call-in port of a computer that is fully connected to the broadcast channel is found. If no call-in port is found, then the seeking computer would select the next portal computer and repeat the process until a portal computer with such a call-in port is found. A problem with such a seeking technique is that all user ports of each portal computer are dialed until a portal computer fully connected to the broadcast channel is found. In an alternate embodiment, the seeking computer selects a port number according to the algorithm and then dials each portal computer at that port number. If no acceptable call-in port to the broadcast channel is found, then the seeking computer selects the next port number and repeats the process. Since the call-in ports are likely allocated at lower-ordered port numbers, the seeking computer first dials the port numbers that are most likely to be call-in ports of the broadcast channel. The seeking computers may have a maximum search depth, that is the number of ports that it will dial when seeking a portal computer that is fully connected. If the seeking computer exhausts its search depth, then either the broadcast channel has not yet been established or, if the seeking computer is also a portal computer, it can then establish the broadcast channel with itself as the first fully connected computer.

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight,