# Distributed Systems — Peer-to-Peer

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/ds
Autumn Term 2012

# Peer-to-Peer Systems

## Overview

- This section of the course will discuss peer-to-peer systems
- We will look at the motivations for a such a system
- The limitations of a such a system
- Characterstics of such systems and hence the suitable types of applications for peer-to-peer systems
- As well as how to provide middleware frameworks for creating peer-to-peer applications which have the additional difficulty that they must be application agnostic

# Peer-to-Peer Systems

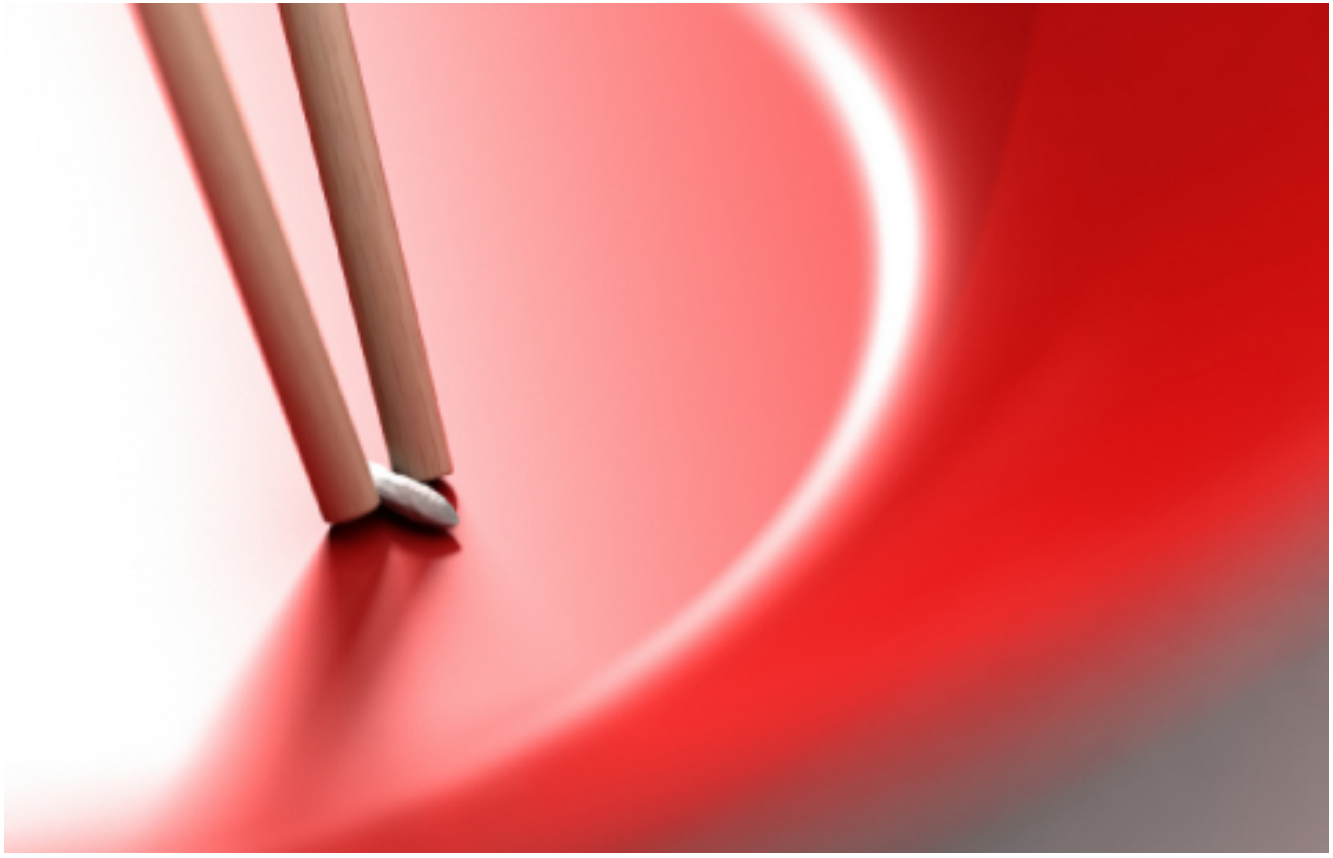
## Google's Daily Processing of Bytes

- Apparently Google (as of around 2009) processes around 24 petabytes of data every day
- This is quite a lot
- How much?

# Peer-to-Peer Systems

## Rice Bytes

- Let's imagine that a single byte is represented by a single grain of rice



# Peer-to-Peer Systems

## Rice Bytes

- A kilobyte, 1K or 1024 bytes then is a 1024 grains of rice, or about a bowl

# Peer-to-Peer Systems

## Rice Bytes

- A megabyte then, represented as rice, is a sack of rice:

## Rice Bytes

- Next up is 1024 megabytes, commonly referred to as a gigabyte
- This is represented as two large shipping containers full of rice
- 1 shipping unit = 1 TEU (twenty-foot equivalent unit)
- We could feed everyone in Edinburgh two bowls of rice

# Peer-to-Peer Systems

## Rice Bytes

- So what is a 1024 gigabytes?
- Less well known, but it is a terabyte
- With this many grains of rice we would require 2048 shipping containers
- It is also enough rice to feed a meal to everyone in the European Union (about 500 million people), twice


This particular ship has a capacity of 1618 TEU

# Peer-to-Peer Systems

## Rice Bytes

- The largest container ships are the Mærsk fleet
- Each can carry 15,500 TEU (containers)
- A petabyte is equivalent to 2097152 containers
- Hence we would need 135 of the largest ever container ship.
- Enough to feed everyone on the planet 146 bowls of rice or cover New York City with about a metre of rice

# Peer-to-Peer Systems

## Rice Bytes

- The largest container ships are the Mærsk fleet
- Each can carry 15,500 TEU (containers)
- A petabyte is equivalent to 2097152 containers
- Hence we would need 135 of the largest ever container ship.
- Enough to feed everyone on the planet 146 bowls of rice or cover New York City with about a metre of rice


# Peer-to-Peer Systems

- That's one petabyte, Google gets through 24 or so a day
- Or 1920 bowls of rice for every one of the 7 billion people on the planet today
- Or covering New York City to a depth of 24 metres in rice

thanks to: `http://noiseinmyhead.wordpress.com/2008/06/26/visualizing-huge-amounts-of-data/`

# Peer-to-Peer Systems

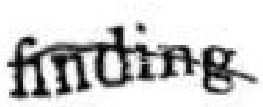
## Centralised Servers

- Providing a service via a single centralised named server is an obvious architecture
- It simplifies much of the design
- But it has an obvious flaw, as the number of clients grows so too does the work done by the centralised server
- Even if we had more computer capacity, we may be limited by the available physical bandwidth to that particular site

# Peer-to-Peer Systems

## A Plausible Solution

- Peer-to-peer systems arose from the realisation that users could contribute some of their own resources to the growing system
- Meaning that as the number of users grows, so too does the number of available resources
- Clay Shirky termed this: exploiting the resources "on the edge of the Internet"
- These resources can be:
  - storage
  - compute cycles
  - bandwidth
  - content
  - human presence

# Peer-to-Peer Systems

## Google Down

- In a very timely fashion Google was unreachable for around 3-5% of the Internet on Monday evening PST.
- Recall the Routing Information Protocol, it is essentially a trust based protocol
- If a particular router claims to be able to route packets to a particular network which it cannot, some other routers may believe
- If so they start sending packets to a network which will be unable to deliver them
- Hence some hosts, will find the target network unreachable

# Peer-to-Peer Systems

## Border Gateway Protocol

- The RIP is a highly simplified version of what is used throughout the Internet
- Often referred to as BGP or Border Gateway Protocol
- Being more complex allows it to be more robust, but at the same time "route leakage" can occur
- This is when the faulty route is leaked out, such that gateways and routers further afield start to route via the faulty route
- In this case, California couldn't reach Google (located in California) because of a faulty route originating from an ISP in Indonesia
- This was likely due to a "fat fingered" address than a malicious attempt to subvert Google

# Peer-to-Peer Systems

Common Features:

1. Their design ensures that each user contributes resources to the system
2. Although their resources may differ, all nodes have the same functionality, capabilities and responsibilities
3. Their correct operation does not depend on the existence of any centrally administered systems
4. They can be designed to offer a limited degree of anonymity to the providers and users of resources
5. A key issue for their efficient operation is the choice of an algorithm for the placement of data (resources) across many hosts and subsequent access to it in a manner that balances the workload and ensures availability without adding undue overheads

# Peer-to-Peer Systems

## Unreliability of Providers

- The owners of the computers sharing resources in a peer-to-peer system may be a variety of individuals and organisations
- None of them provide any level of service guarantee, in particular nodes join and leave the system *at will*
- Leading to unpredictable availability of any particular process/node
- Meaning that the provision of any particular resource should not depend upon the continued availability of any particular node
- Preparing for this requires redundancy in a way which may help against malicious attack or unpredicted outages
- The required redundancy may even help with performance
- As a last resort, we may simply have to put up with unavailability of certain resources

# Peer-to-Peer Systems

## Popular Uses

- These features mean that a corporations hoping to collect revenue from a service have shied from such systems
- It is difficult to make any kind of service level guarantees
- However peer-to-peer have been very popular for file-sharing systems mostly because such systems do not pretend to offer any particular level of service, they operate a strictly "maybe" policy
- In addition a relatively large level of service can be obtained from very little outlay
- Academics have therefore also been somewhat drawn to peer-to-peer systems

# Peer-to-Peer Systems

## Distributed Computation

- Peer-to-peer systems are generally associated with the sharing of data resources and the bandwidth required to access those shared data resources, but we noted other resources
- The famous *SETI@Home* project aims to use individuals' spare computing cycles to perform part of the larger computation of analysing received radio signals for intelligent communication
- *SETI@Home* is an interesting example as it does not require communication between individual nodes
- That is, each segment may be analysed in isolation
- A brand of computation that is termed "*embarrasingly parallelisable*"
- Utilising the Internet's vast array of computers for a broader range of tasks will depend upon the development of a distributed platform which supports communication between participating nodes

# Peer-to-Peer Systems

## Distributed Computation

- There is a further threat to the platform of distributed computing
- Climate Change
- When distributed computing first became popular it was seen as a very green use of otherwise idle (but switched on) computers
- Computers at the time used roughly the same amount of energy to remain switched and idle as when doing some calculation
- Hence using those idle computers to do anything remotely useful was seen as a great re-use of resources
- Today though, computers use much less energy when idle and hence running them at full power to perform a large computation is seen as a waste of energy unless that computation is somewhat important

# Peer-to-Peer Systems

## Three generations

- Although peer-to-peer systems have existed since at least the 1980s, they first really became popular when always-on broadband became generally available (start of this century)
- We can identify three generations of peer-to-peer systems:
  1. Napster music exchange — relied in part on a central server
  2. File sharing systems — with greater fault tolerance and no reliance on a central server, examples include:
     - Gnutella
     - DirectConnect
     - Kazaa
     - Emule
     - Bittorrent
     - FreeNet
  3. The emergence of middleware layers for peer-to-peer systems — making possible the application independent provision of resources

# Peer-to-Peer Systems

## Napster

- Napster was an early offering in peer-to-peer style systems
- Offering the ability for users to share data files it quickly became popular with those sharing music files
- However Napster was shut down as a result of:
  - People sharing copyrighted music
  - This lead the owners of the copyrighted material to instigate legal proceedings against the Napster service operators
  - This in turn caused the Napster service to be shut down

# Napster

## Napster's Modus Operandi

- Napster relied upon a central index of files available for download
- Each new peer that joined the network, communicated to the central service a list of all available files
- When a user had a request for a particular file the following steps where executed:
  1. A file location request is made by a user to the centrally managed Napster index
  2. The Napster server responds to the request with a list of peers who have the requested file available
  3. The user then requests that file from one of the list of peers
  4. The peer from which the file is requested then delivers the file directly to the requesting user, without central server intervention
  5. Finally, once the requested file is received by the user it informs the centrally managed Napster server such that the index of files may be updated
     - That is, the requesting user now has the particular file

# Napster

## Key point

- The indexing system was not distributed (though it was replicated)
- The distributed resources were both the available files
  - In terms of the fact that they are stored on peer computers and not any centrally managed machines
  - Additionally in that they originated from the users themselves
  - And finally the bandwidth available at each peer, since files are delivered straight from peer to peer without going via a central server

# Peer-to-Peer Systems

## Legal Proceedings

- Napster argued that they were not liable for the copyright infringement because they were not part of the copying process
- The argument ultimately failed as the index servers were viewed as an essential part of the copying process
- The index servers were at known network addresses, meaning that their owners could not retain anonymity
- Hence they could be targeted by lawsuits

# Peer-to-Peer Systems

## Napster Lessons and legacy

- Napster performed load balancing, directing user requests to users closer (in terms of network hops) to the requesting user
- Thus avoiding all users requesting a file from the same user
- Napster used a replicated, unified index of all available music files, this didn't represent a huge limitation since there was little requirement for the replicated indexes to be consistent
- But it could be a limitation for another application
- Napster also took advantage of the fact that music files are immutable data resources, they do not get updated
- No guarantees were made about the availability of any particular file. A user made a request which may or may not be satisfied

# Napster Lessons and legacy

- Napster then was ultimately shutdown
- But many derivative file-sharing networks live on
- Independence from any centrally managed server makes legal action far harder to pursue and ultimately less potent
- Whatever your views on the sharing of © material it is not particularly difficult to imagine "legitimate" uses
- Many people around the world are opressed in particular without right to the freedom of expression
- Many countries for example do not allow access to Facebook or Twitter
- During the "Arab Spring" the use of sites such as Twitter and Facebook are well known to have been crucial
- Both were blocked by several governments in an attempt to quash an uprising

# Peer-to-Peer Systems

## Peer-to-Peer Middleware

- With the third generation of peer-to-peer systems came about the development of middleware on top of which peer-to-peer systems could be built
- Developing middleware is more problematic than a single application because we cannot take advantage of any application specific assumptions
- Such as the file sharing assumption that there need be no guarantee of the availability of any particular file

# Peer-to-Peer Middleware

## Indexing

- Restricting ourselves for the moment to providing access to data resources, a key problem is the indexing of available files to hosts at which those files are available
- Napster, used a central server with a known address
- Gnutella and other second generation peer-to-peer file-sharing systems use a partioned and distributed index
- Both systems made the assumption that different users could have different results when requesting access to a specific resource

# Peer-to-Peer Middleware

## Functional Requirements

- The aim of peer-to-peer middleware is to simplify the construction of services implemented over widely distributed hosts
- Any node must therefore be able to locate and communicate with any individual resource which is made available
- The system must be able to cope with the arbitrary addition or removal of resources and hosts
- As with all middleware, peer-to-peer middleware (if it is to be widely adopted) must offer a simple/appropriate programming interface

# Peer-to-Peer Middleware

## Non-functional Requirements

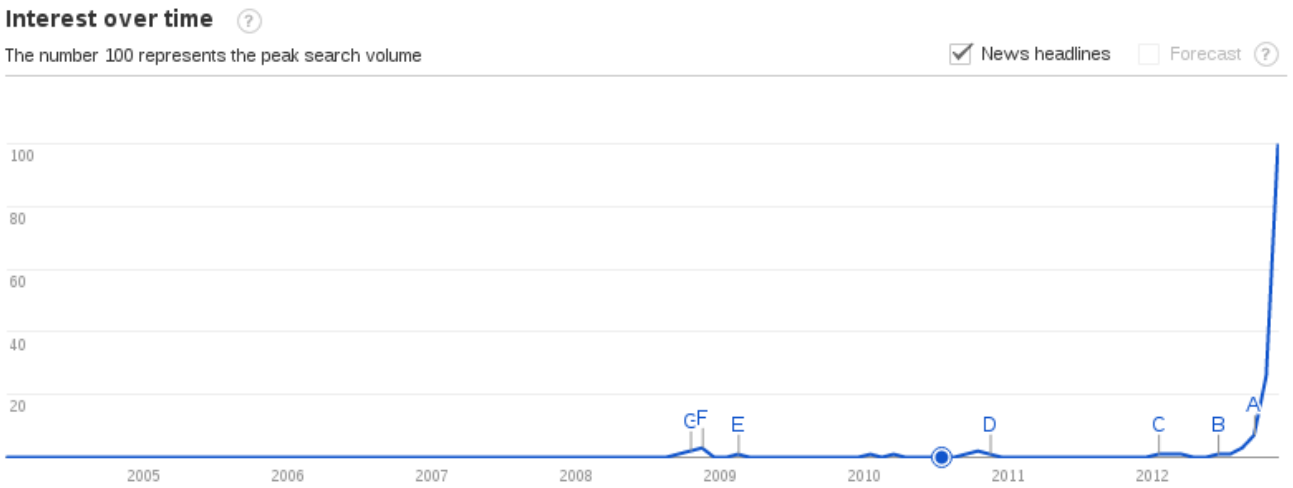
- Global Scalability — the very idea of peer-to-peer systems is to both cope with and exploit large numbers of users. Peer-to-peer systems must therefore be able to support applications that access millions of objects on hundreds of thousands of hosts
- A peer-to-peer system should be able to take advantage of the ability for service provision to grow dynamically as the number of users increase
- In the previous part of the course we saw how system virtualisation can aid a central service in dynamically adjusting service provision but for a peer-to-peer system this should not be necessary

# Peer-to-Peer Middleware

## Non-functional Requirements

- Load Balancing — The performance of any system exploiting large numbers of hosts, even if those hosts were co-located, depends upon being able to distribute the load across those hosts evenly.
- This can be achieved to some extent by randomly placing resources and replicating heavily used resources

Web searches for "Nate Silver"



# Peer-to-Peer Middleware

## Non-functional Requirements

- Optimisations for local interactions — The "network distance" between peers has a large impact on the latency of individual interactions. Additionally network traffic is highly impacted if there are many distant interactions
  - We saw an example of this for Napster, that attempted to return to a requesting user, provider hosts which were "network near" to the requesting host

# Peer-to-Peer Middleware

## Non-functional Requirements

- Accomodating highly adaptable host availability — Most peer-to-peer systems are constructed such that hosts are free to join or leave at any time. Some studies of peer-to-peer networks have shown large turnover in participating hosts. Re-distribution of load when hosts join and leave is a major technical challenge
  - Note that it may even be that all members interested in a particular resource leave, but that that resource should not disapear
  - Consider a peer-to-peer social network, say a peer-to-peer Facebook
  - A single user's profile must be retained even when not only that user has left but also all the friends of that particular user

# Peer-to-Peer Middleware

## Non-functional Requirements

- Security of Data — Particularly in an environment of heterogeneous trust.
  - File sharing systems do not by their very nature require much of security of data, the whole point is that data is shared
  - Consider again the peer-to-peer version of Facebook
  - A single user's profile must be stored on several machines, but should only be available to a group of authorised users (that user's friends)

# Peer-to-Peer Middleware

## Non-functional Requirements

- Anonymity and Deniability — Anonymity is a legitimate concern for many applications
  - In particular situations demanding a resistance to censorship.
  - "whistleblowing" on a company or group
- A related requirement is that hosts demand a root to deniability if they are to be used to store/forward data originating from other users. Otherwise the risk in involving oneself in a peer-to-peer network is high. Here the use of a large number of hosts can actually be an advantage. The key phrase is "plausible deniability"
- Key disclosure laws — some countries inforce that the user supply a key to law enforcement/government representatives for any encrypted data (or enforce mandatory decryption)
- In the UK at least three people have been prosecuted and convicted for refusing to supply decryption keys
- The defence is to "prove" that one does not possess the encryption key or that the data is random

# Peer-to-Peer Systems

## Obvious Solution

- Recall that we want a service such that: Any node is able to locate and communicate with any individual resource which is made available
- The obvious solution is to maintain a database at each node of all resource (objects) of interest
- This isn't going to work though for several reasons:
  1. It does not scale
  2. It involves a heavy amount of traffic to relay all updates to all nodes
  3. Not all nodes are always available, hence re-joining the network would have a heavy cost associated with it
- Knowledge of the locations of all objects must be partitioned and distributed throughout the network
- A high degree of replication is required to counteract the intermittent availability of hosts

# Peer-to-Peer Systems

## Telephone Trees

- Not so common now since we have convenient broadcast of messages via text or email
- The goal is to broadcast some message to a group of people,
  - generally these were the parents of a group of children
  - the message related to say the ETA back from some group excursion
- Each parent knew the phone numbers of up to four others
- When they received a call giving information, it was then their duty to inform the "branches" of which they knew
- This was, in a sense, a routing overlay, built upon the routing mechanism already in place for the telephone system
- Although of course in this case it was used for broadcasting rather than locating a resource

# Peer-to-Peer Systems

## GUIDs

- Peer-to-Peer systems usually store multiple copies of any given resource object as a redundancy guard against unavailability of a single copy
- Each object is associated with a GUID (globally unique identifier)
- Each person in the phone-tree did not need to know the names, addresses, or anything about those individuals to which they should forward the call
- They only required to know their GUID, which was in this case their phone number
- GUIDs should be opaque, that is, they reveal nothing about the object to which it refers or its location (see later)
  - In this sense they are nothing like a postal address
  - More *like* your mobile phone number

## GUIDS — small aside

- The Open Software Foundation recommends an algorithm for generating GUIDs
- V1 of this algorithm used, as a part of the GUID, the network card *MAC* address
- Meaning that the creator of a GUID (and hence a document to which it is attached) could be determined from the GUID alone
- This fact was used to David L. Smith the person who released the *Melissa* virus into the wild
- He was sentenced to 10 years (serving 20 months) and fined $5000
- V4 of the algorithm does not do this

## Routing Overlays

- A distributed algorithm known as a routing overlay takes responsibility for routing requests to some node which holds the object
- The object of interest may be placed at, and subsequently relocated at any node in the network
- It is termed an overlay since it implements in the client a routing algorithm that is quite separate from the routing of individual IP packets
- The routing overlay ensures that any node can access any object through a sequence of nodes, by exploiting the knowledge at each of them to locate the destination object

# Routing Overlays

## Main Tasks of the Routing Overlay

- Routing of Requests to Objects
  - A client wishing to perform some act upon a particular object must send that that request, with the GUID attached, through the routing overlay
- Insertion of Objects
  - A node wishing to insert a new object, must compute a new GUID for that object and announce it to that routing overlay such that that object is available to all nodes
- Deletion of Objects
  - When an object is deleted the routing overlay must make it unavailable for other clients
- Node addition and removal
  - Nodes may join and leave the service at will. The routing overlay must organise for new nodes to take over some of the responsibilities of other (hopefully nearby) nodes
  - When a node leaves, the routing overlay must distribute its responsibility to remaining nodes

# Overlay Routing

## Distributed Hash Tables

- A distributed hash table has three operations:
  1. *put*(*GUID*, *data*): stores data at all nodes responsible for the object identified by GUID
  2. *remove*(*GUID*): deletes all references to GUID and the associated data
  3. *get*(*GUID*) : retrieves the data associated with GUID from one of the nodes responsible for it
- Note then that operations may be subject to mutual-exclusion style race conditions
- A count of something for example involves first retrieving the current count and storing the incremented count. These two operations could clearly be interleaved by two concurrent processes

## Distributed Object Location and Routing

- DOLR has the following operations:
  1. *publish*(*GUID*): Makes the node performing the *publish* the host for the object corresponding to *GUID*. The *GUID* should be computed from the object (or a part of it).
  2. *unpublish*(*GUID*): Makes the object corresponding to *GUID* unavailable.
  3. *sendToObj*(*msg*, *GUID*, [*n*]): Sends a message to the target object. This could be a request to update the object, or more likely, a request to open a connection in order to transfer the data associated with the object.
     - [*n*] is optional and specifies the number of replicas that the delivery of the same message should reach

# Overlay Routing

## Replication

- In order that an object remains available across node addition and removal, storage of an object must occur at more than one node
- For a Distributed Hash Table, some replication factor $r$ is chosen (an appropriate choice gives a very high probability of continuous availability)
  - The object is then replicated at $r$ nodes which are the $r$ nodes numerically closest to the host node
- For the Distributed Object Location and Routing protocol, locations for the replicas of data objects are decided outwith the routing layer.
  - The DOLR layer is notified of these *host address* of each replica using the *publish* operation

# Peer-to-Peer Systems

## Readable Object Identifiers

- GUIDs, nice though they are, are not human readable
- Client applications must therefore obtain the GUIDs for resources using some human-readable name or search request
- Ideally, such a lookup is also stored in a peer-to-peer manner
- This avoids a centralised service a la Napster and the associated disadvantages of such a centralised service
- Bittorrent is an interesting example, it uses individual web pages to publish "stub" files
  - The stub file includes the object's GUID and:
  - The URL of a *tracker* which holds an up-to-date list of network addresses willing to provide the requested object
  - Note that it essentially uses existing search engines as the search facilities
  - Websites with particular object "stub" files may be "attacked", but:
    - There may be many of them
    - Each web site may only host a small number, perhaps only a single, stub file

# Peer-to-Peer Systems

## Pastry

- Implements a Distributed Hash Table
- Can be used for any application for which objects are stored and retrieved
- generally more useful if the objects are immutable or updated rarely
- Squirrel is an application built upon Pastry, it is a peer-to-peer web-cache system
- This works well as although the objects may be updated it is not crucial that all replicas are consistent

# Pastry

## Pastry Routing Overlay

- In Pastry each object and node is given an opaque 128-bit GUID
- In a network with $N$ participating nodes the Pastry routing algorithm will deliver a message to an object or node within $\mathcal{O}(logN)$ steps
- If the GUID identifies a currently inactive node then the message is delivered to the node with a GUID numerically closest to the target GUID
- Each step along the route involves the use of an underlying transport protocol, usually UDP.
- Each such step, transfers the message from the current node to a node which is numerically closer to its destination
- Closer here though is in the entirely artificial GUID space and may in fact involve routing the message geographically more distant to the target node than the current node

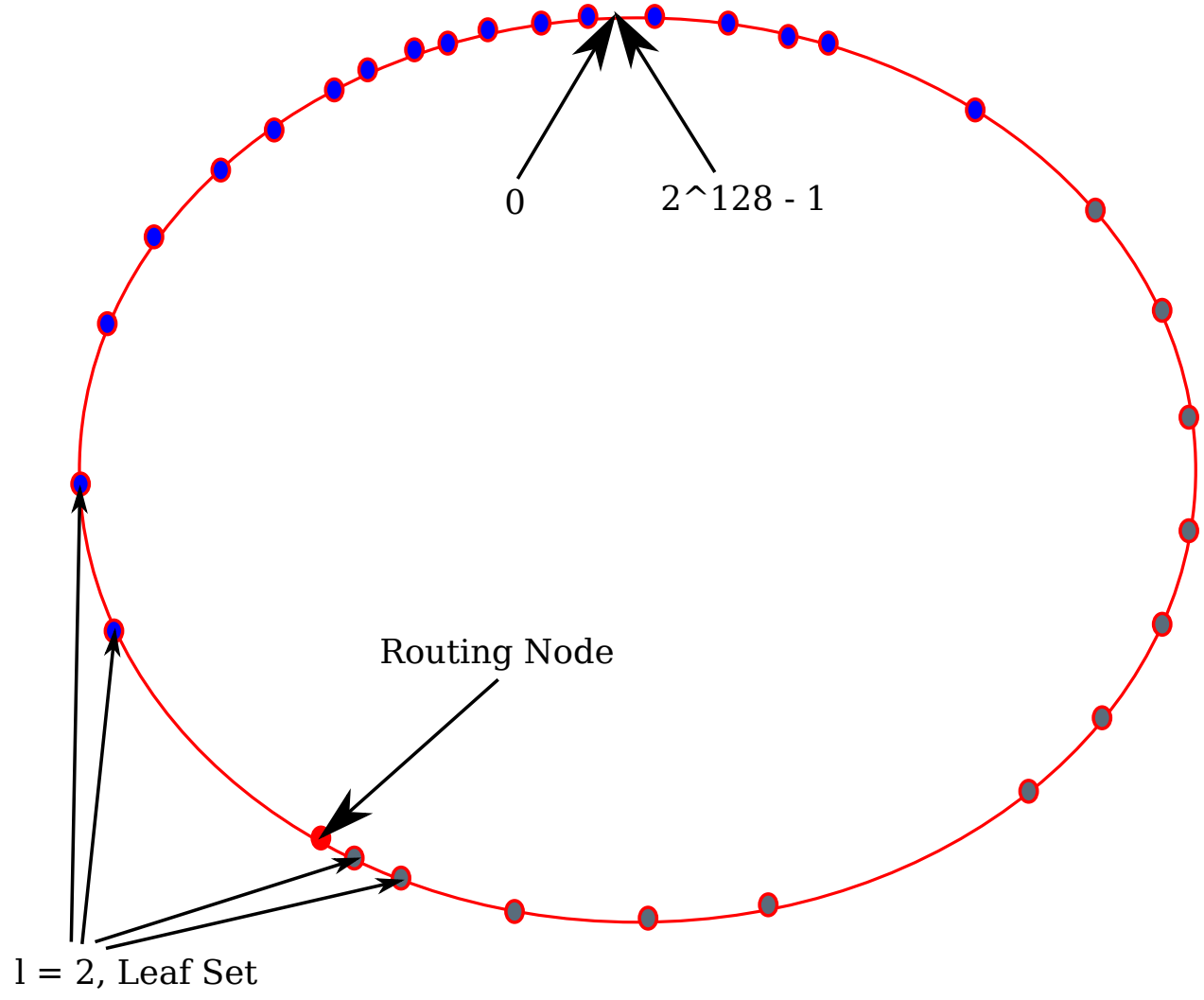
# Peer-to-Peer Systems

## Routing Overlay — Ring based

- Each node stores a vector $L$ of size $2 \times l$ containing the GUIDs and IP addresses of nodes whose GUIDs are numerically closest on each side: $l$ nodes above and $l$ nodes below
- The vector $L$ is known as the leaf set, and leaf sets are updated when nodes join or leave the network
- The GUID space is treated as circular, so GUID 0's lower neighbour is $2^{128} - 1$ and vice versa
- Any node with GUID $D$ upon receiving a message for $D'$:
  - If $D'$ is in $L$ then $M$ can be directly forwarded to the target node
  - Otherwise $M$ is forwarded to the GUID in $L$ numerically closest to $D'$. Which will be either the left most or the right most node in $L$
    - It is the right most, if $D' > D$ and $D' - D < \frac{2^{128}-1}{2}$ or $D > D'$ and $D - D' > \frac{2^{128}-1}{2}$
    - And the left most node otherwise
- To deliver a message we require at most $\frac{N}{2 \times l}$ hops

# Routing Overlay

## Ring-Based



## Routing Overlay — Using Routing Tables

- Each node maintains a tree-structured routing table giving GUIDs and IP addresses for a set of nodes spread throughout the entire range $0 \ldots 2^{128} - 1$
- However the routing table for node with GUID $D$ will have an increased density of coverage for GUIDs which are numerically close to $D$

## GUID Routing Table

The routing table for a node with GUID $90B\ldots$

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C |
| | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ | | $n_A$ | $n_B$ | $n_C$ |
| 1 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C |
| | | $n_{91}$ | $n_{92}$ | $n_{93}$ | $n_{94}$ | $n_{95}$ | $n_{96}$ | $n_{97}$ | $n_{98}$ | $n_{99}$ | $n_{9A}$ | $n_{9B}$ | $n_{9C}$ |
| 2 | 900 | 901 | 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 90A | 90B | 90C |
| | $n_{900}$ | $n_{901}$ | $n_{902}$ | $n_{903}$ | $n_{904}$ | $n_{905}$ | $n_{906}$ | $n_{907}$ | $n_{908}$ | $n_{909}$ | $n_{90A}$ | | $n_{90C}$ |

$\ldots$ the table has as many rows as there are hexadecimal digits in a 128 bit number, 32

- 128 bit GUIDs are examined as a string of 32 hexadecimal digits
- Each row has 15 entries (curtailed here for space)
- One for each value that does not match the current node's prefix
- The entry in each cell is the IP address of a node with a GUID with the prefix corresponding to the row and column

# Pastry

## The Pastry Routing Algorithm

- To handle a message $M$ addressed to GUID $D$ at node $A$, where $R[p, i]$ is the element at column $i$, row $p$ of the routing table at node $A$:
  1. If$(L_{-l} < D < L_l)$
  2. Forward $M$ to the element $L_i$ of the leaf set with the GUID closest to $D$ or the current node $A$
  3. else
  4. Find $p$, the length of the longest common prefix of $D$ and $A$, and $i$, the $(p+1)^{th}$ hexadecimal digit of $D$
  5. If $(R[p, i] \neq null)$
  6. Forward $M$ to $R[p, i]$
  7. else
  8. Forward $M$ to any node in $L$ (or $R$) with a common prefix of length $p$ but a GUID that is numerically closer
- The lines in grey implement the previous ring-based algorithm, hence we can be sure that the algorithm will succeed in routing each message

# Pastry Routing Algorithm

- Each table must have the property that:
  - The GUID of the node addressed in $R[p, i]$ has a common prefix with the target GUID $D$ of length at least $p + 1$
  - Provided of course that $D[p] = i$
  - Another way of saying this is, should we have the cell: $\left| {}^{16A2C}_{\ \ n} \right|$
  - Then $n$ addresses a node with a GUID with the prefix $16A2C$
  - Note that we would not have such a cell if the *current* node had the prefix $16A2C$
  - Hence each time a message is forwarded it is forwarded to a node with a GUID that has longer matching prefix than the current node, so eventually it must be forwarded to the correct node

| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ | | $n_A$ | $n_B$ | $n_C$ |
| 1 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C |
| | | $n_{91}$ | $n_{92}$ | $n_{93}$ | $n_{94}$ | $n_{95}$ | $n_{96}$ | $n_{97}$ | $n_{98}$ | $n_{99}$ | $n_{9A}$ | $n_{9B}$ | $n_{9C}$ |
| 2 | 900 | 901 | 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 90A | 90B | 90C |
| | $n_{900}$ | $n_{901}$ | $n_{902}$ | $n_{903}$ | $n_{904}$ | $n_{905}$ | $n_{906}$ | $n_{907}$ | $n_{908}$ | $n_{909}$ | $n_{90A}$ | | $n_{90C}$ |

. . . the table has as many rows as there are hexadecimal digits in a 128 bit number, 32

# Pastry Overlay Routing

## Host Integration

- When a host joins the network it must follow a specific protocol to obtain its table and leaf nodes as well as updating others
- The new node first computes a suitable GUID for itself
- The joining node should have the address of at least one existing node, it contacts this (or finds a nearer neighbour, where nearer is in reference to actual network distance)
- Suppose our new node has GUID $X$ and its first contact has GUID $A$. The node sends a *join* request message to $A$ giving $X$ as its destination GUID
- The node $A$ then forwards this request message to the node with the numerically closest GUID to $X$, let's call it $Z$
- Of course $A$ does not in general know what that node is, it simply forwards on the *join* message as though routing to node $X$

# Host Integration

## Building the Routing table for $X$

- The key point is that a node ($Z$) must be able to tell that it is the currently closest (numerically) GUID to $X$
- It can know this due to its own leaf set
- As the *join* message is forwarded (ultimately to $Z$), the forwarding nodes help build up the routing table of $X$
- Note that the first row of $X$ does not really depend upon the GUID $X$, so it can simply copy the first row of $A$.
  - It must update it slightly since $X$ and $A$ do not necessarily share the same first digit
  - In place $R_A[0, i]$ where $i = A[0]$ is the first digit of $A$, there will be no address, so in slot for $X$ we can simply place the address of $A$
  - Additionally $R_X[0, j]$ where $j = X[0]$ is the first digit of $X$ can be left empty even though $R_A[0, j]$ may not be

# Host Integration

## Concrete example of $X$ and $A$

- Suppose the GUID of $X$ is number $1 (0000 \ldots 1)$
- The GUID of $A$ is number $2^{127} (1000 \ldots 0)$
- The first row of $A$ in positions $2 \ldots F$ are perfectly valid
- The value that $A$ has for prefix 0 is worthless to $X$, but that's okay because in that position $X$ will have no address (it's the red entry in the first row for $X$ because it is the prefix of $X$)
- $A$ has no entry in column 1 (it's $A$'s red entry in the first row), but that's okay because we know a good address to fill in that column in $X$'s first row, the value is the address of $A$.

# Host Integration

## Routing of the Join message

- The second row of $A$'s table though is probably not relevant
- During its travels to the node numerically closest to $X$, $(Z)$, the join message passes through some nodes $B, C \ldots Y$
- Each node $B, C \ldots Y$ through which the join message passes, transmit relevant parts of their routing tables and leaf sets to $D$
- Because of the routing algorithm the second row of $B's$ table will be relevant for $X$, so it simply sends $X$ its second row, and also forwards the message on to $C$
- Now the third row of $C$ should be applicable for $X$ since it shares the same prefix of at least length 2.
- In fact $C$ may have been in row $n$ of $B$'s table and hence can send $X$ rows $2 \ldots n$
- When the message finally arrives at $Z$, we should have built up most of $X$'s new routing table, and all we require is a good leaf set

# Host Integration

## Routing of the Join message

- $Z$ is the numerically closest GUID to $X$
- Suppose $X > Z$:
  - The left hand side of $X$'s leaf set is the left hand side of $Z$'s but $Z$ itself
  - The right hand side is exactly the right hand side of $Z$'s original leaf set
  - $Z$ however should update the right hand side of its leaf set to include $X$ as the closest and optionally remove the right most node from the leaf set.
- Finally then once $X$ has received and built up its own routing table and leaf nodes, it sends this information to all the nodes in its leaf node set and routing table such that they may update their accounts appropriately
- Incorporating this new node into the network requires the transmission of $\mathcal{O}(logN)$ messages

# Pastry Overlay Routing

## Host Removal

- A host may fail or leave at any time
- When this happens we must repair the routing tables and leaf sets so as not to contain the departed node
- We will assume that neighbouring nodes can detect a failed node via periodic polling and consider mostly the case of a node which departs intentionally
- Assume either way that a node $D$ detects, or is alerted by the departing node itself, that node $X$ has left the network
- Node $D$, looks for a close node $L'$ in its own leaf set and requests a copy of the leaf set of $L'$
- The leaf set which $L'$ sends $D$ should overlap that of $D$ and in particular contain a node suitable to replace that of $X$
- Other neighbouring nodes are then informed of the failure and they perform a similar procedure

# Pastry Overlay Routing

## Fault Tolerance

- Nodes may gracefully leave the system, but they may also fail, in a peer-to-peer system this could represent the user switching off or killing the process
- Failed nodes are detected through a system of "heartbeat" messages sent by non-failed nodes to their leaf sets
- However, failed node notification will not propagate through the network quick enough to eliminate routing failures
- If the application in question requires reliable delivery of messages then a reliable protocol must be built upon the routing overlay
- Recall at the start of the course we discussed reliable (TCP) communication and unreliable (UDP) communication
- One reason to use unreliable communication is that the application built ontop of the communication may be required to perform its own ommission/error detection/correction
- This is one such example

# Pastry Overlay Routing

## Fault Tolerance

- Where such a *re-try* mechanism is used it should allow the *Pastry* routing overlay time to adapt to an error
- However, as it stands this may not overcome all errors and certainly will not help in the presence of a malicious node
- To overcome this, an element of randomness is introduced into the routing algorithm
- To forward a message a node $P$, might choose not to immediately send it it to the node in $P$'s routing table with the longest matching prefix, but instead, with a small probability, send to a node higher up the routing table.

| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ | | $n_A$ | $n_B$ | $n_C$ |
| 1 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C |
| | | $n_{91}$ | $n_{92}$ | $n_{93}$ | $n_{94}$ | $n_{95}$ | $n_{96}$ | $n_{97}$ | $n_{98}$ | $n_{99}$ | $n_{9A}$ | $n_{9B}$ | $n_{9C}$ |
| 2 | 900 | 901 | 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 90A | 90B | 90C |
| | $n_{900}$ | $n_{901}$ | $n_{902}$ | $n_{903}$ | $n_{904}$ | $n_{905}$ | $n_{906}$ | $n_{907}$ | $n_{908}$ | $n_{909}$ | $n_{90A}$ | | $n_{90C}$ |

. . . the table has as many rows as there are hexadecimal digits in a 128 bit number, 32

# Pastry Overlay Routing

## Locality

- The routing table has an address associated for each possible digit in the $i^{th}$ position which does not match the current node's $i^{th}$ digit
- Each such address has a GUID with a prefix of length $i-1$ which matches the current node's
- In a well populated overlay, and in particular in the early rows of the table, there will be many such choices
- Each choice is made based on a metric which measures network locality
- Usually IP hops, or round-trip time
- This cannot guarantee optimal routings but has been shown in simulations to produce routes that are only 30-50% longer
- It also helps route around failed nodes which have large round-trip times

# Peer-to-Peer Systems

## Tapestry

- *Tapestry* is similar in goals to *Pastry*
- The *Tapestry* infrastructure uses a distributed hash table routing mechanism similar to the one described for *Pastry*
- However, the exposed *Tapestry* API is that of a DOLR (Distributed Object Location and Routing) interface
- Recall: DOLR has the following operations:
  1. *publish*(*GUID*): Makes the node performing the *publish* the host for the object corresponding to *GUID*. The *GUID* should be computed from the object (or a part of it).
  2. *unpublish*(*GUID*): Makes the object corresponding to *GUID* unavailable.
  3. *sendToObj*(*msg*, *GUID*, [*n*]): Sends a message to the target object. This could be a request to update the object, a request to open a connection in order to transfer the data associated with the object.
     - [*n*] is optional and specifies the number of replicas that the delivery of the same message should reach

# Peer-to-Peer Systems

## Tapestry

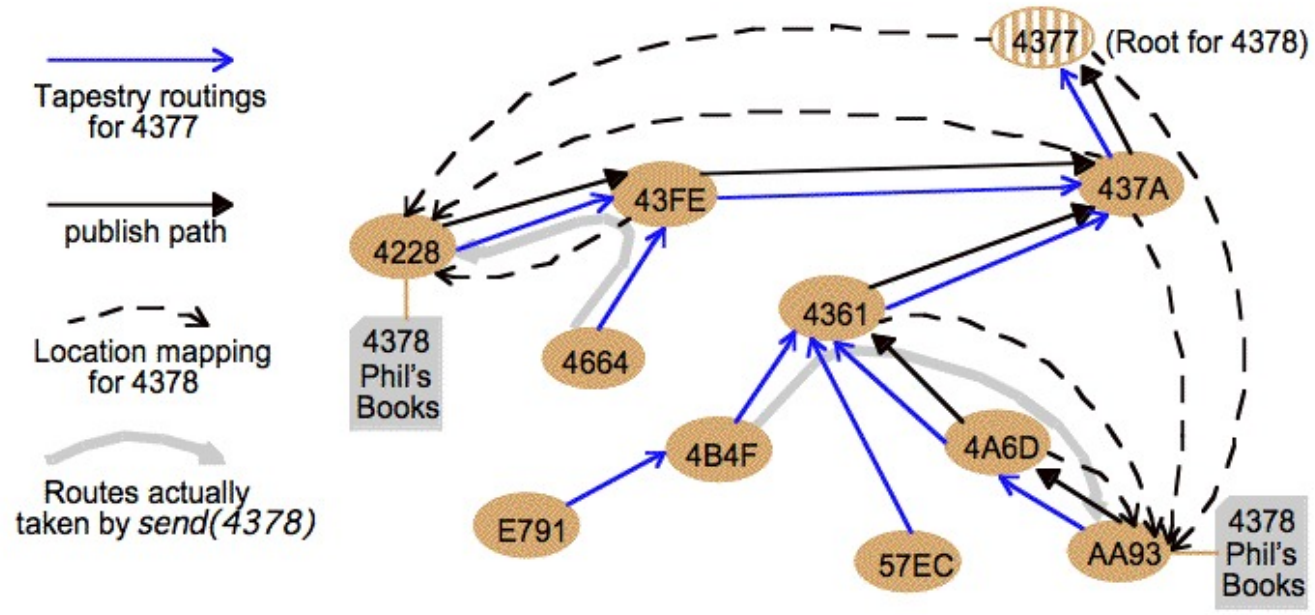
- Because replication is handled by the application rather than *Tapestry* itself, this gives applications additional flexibility in how to handle replication
- For example a file-sharing system may not need to explicitly handle replication since it done implicitly whenever a user copies an existing resource
- It is possible that absolutely no replication (of at least some resources) is necessary or desired
  - For example an online game could operate with each player hosting their own current state
  - When the player leaves, the state need not persist
  - Though the player's account may persist, this would be an example where some, but not all of resources are replicated

# Tapestry

- Each object and routing node has a 160-bit identifier (GUID) associated with it
- In addition each (published) object is associated with exactly one "*root node*"
- The root node maintains a table mapping object GUIDs to the addresses of all replicas
- The root node will be the node with a GUID numerically closest to the GUID associated with the object
- When a node invokes $publish(GUID)$ the message is routed to the object's associated root node
- When a $sendToObj(GUID, msg, [n])$ message is invoked that too is routed to the root node of the object
  - The root node may then choose how many and which replicas to send that message to
  - The decision obviously being application dependent

## Tapestry Routing

Figure 10.10: Tapestry routing From [Zhao et al. 2004]




Replicas of the file *Phil's Books* (G=4378) are hosted at nodes 4228 and AA93. Node 4377 is the root node for object 4378. The Tapestry routings shown are some of the entries in routing tables. The publish paths show routes followed by the publish messages laying down cached location mappings for object 4378. The location mappings are subsequently used to route messages sent to 4378.

# Peer-to-Peer Systems

## Structured or Unstructured

- Structured peer-to-peer networks have a specific distributed data structure maintaining the routing overlay
- The structure imposed means that the peer-to-peer networks are efficient, offering some bound on, say, the number of messages required to route a message to an object
- Pastry for example relied upon the logical ring of GUID ids, and the routing tables made up distributed 'trees'
- However this is paid for in the cost of maintaining the distributed data structure underneath the peer-to-peer network
- An alternative is an *unstructured* peer-to-peer network

# Peer-to-Peer Systems

## Unstructured

- An unstructured peer-to-peer network does not rely on any distributed data structure
- Instead it relies upon an ad-hoc system of adding peers as they become available
- Each node joins the network by following some simple, local rules.
- A joining node must establish connectivity with a set of 'neighbours'
- It knows that the neighbours will also be connected to their own neighbours and so on
- Connectivity to everyone follows from a 'Kevin Bacon' style arrangement, except that there is no special node

# Unstructured Peer-to-Peer

## Locating an object

- In an unstructured peer-to-peer network then, it is straightforward and inexpensive to join and leave a network
- However locating an object must be done by searching the resulting "mish-mash" of connections
- This approach then cannot guarantee to locate any specific object
- It is also possible that excessive amounts of traffic are used in locating and using objects
- Still, unstructured peer-to-peer networks have been shown to work
- In fact they are the dominant paradigm used in the Internet today

# Peer-to-Peer Systems

## Unstructured dominance

- 
  - Gnutella
  - Limewire
  - Freenet
  - Bittorrent
- All examples of unstructured peer-to-peer networks
- Many studies have estimated the overall proportion of Internet traffic which is peer-to-peer
- They vary widely in their estimates from some 20% to over 70%
- Safe to say it is a significant proportion, it's hard to say what is taken up with unnecessary transfer of data

# Unstructured peer-to-peer systems

## Unnecessary Data Transfer

- A variety of reasons, including inefficiency of the peer-to-peer system in question which may not be satisfying requests, dropping messages, or simply not pairing up providers with consumers in a network-efficient manner
- We may also get a lot of dropped connections because peers may leave at any time — file splitting is used to mitigate this
- Broken files, incorrectly labelled files etc
- Due to the uncertainty of availability many users "download now, consider later"
- Content may not be offered in the size desired, eg a whole album as opposed to single song which is desired

# Structured vs Unstructured

## Comparison

- Structured
  - Advantages
    1. Guaranteed to locate (existing) objects
    2. Relatively low message overhead
  - Disdvantages
    1. Need to maintain complex overlay structures
    2. Slow to adapt to highly dynamic networks
    3. Software is difficult to upgrade if it updates the distributed data structures used
- Unstructured
  - Advantages
    1. Self-organising and naturally resilient to node failure
    2. Different versions of software can often interoperate with little engineering effort
  - Disdvantages
    1. Offers no guarantees on locating objects even if they exist
    2. Can generate large amounts of messaging overhead

# Unstructured Peer-to-Peer

## Searching

- When file-sharing, a major problem is the location of desirable files
- We will stick to the problem of file-sharing but the same problem exists for many other similar applications
- Whether we are using a structured or unstructured peer-to-peer network we may still require to do some search to find an appropriate GUID
- The search strategies we look at now are applicable in a number of places, but we will specialise the case to search for a file in an unstructured peer-to-peer network

# Peer-to-Peer Systems

## File Searching

- The problem of searching for a particular file (or one of a set which is appropriate) becomes the problem of searching the entire network
- Naïvely done this could flood the network with many search requests
- A simple strategy is that a search request is sent to the nearest neighbours, each of whom respond with success or forward the search on to their neighbours
- Similar to IP multicast, each such search request has a time-to-live variable which is decremented each time the request is forwarded
- The approach though does not scale well

# Peer-to-Peer Systems

## Improvements

- Expanded Ring Search
  - If there is an effective replication strategy in place, many searches may complete successfully locally
  - This is particularly true of file-sharing networks where the most popular files are those which are searched for the most often
  - Expanded ring search does the same as the naïve version but starts with a very small time-to-live variable
  - If that search fails, it tries again with a larger time-to-live variable
  - and so on, up to some limit

## Improvements

- Random Walks
  - A search agent can be set off in search of the desired file
  - The agent is of course not an actual agent but simply a message
  - When the message arrives at a node, the successfully found file can be sent directly back to the originator of the random walk agent
  - If not, it is forwarded to one other peer, the choice of peer is made randomly
  - A peer wishing to search may set off several random "agents" concurrently
  - Again they are generally equipped with a time-to-live counter

# Peer-to-Peer Systems

## Improvements

- Gossiping
  - A node sends a request to a neighbour with a given probability
  - Hence a request spreads probabilistically through the network
  - The *Gossiping* name alludes to the way in which a search spreads through the network as a rumour spreads through social networks
  - Sometimes these are called *epidemic protocols*, because the (in the case) search spreads through the network like a virus

# Peer-to-Peer Systems

## Improvements

- Ultra-Peers
  - In a pure peer-to-peer network all peers are treated equally
  - An ultra-peers system makes the observation that we may treat peers as equals but that does not reflect reality
  - A few selected peers are designated *ultra-peers*, generally because they have extra resources and some commitment to extended availability within the peer-to-peer system
  - These ultra-peers are heavily connected with each other, and ordinary peers connect themselves to one or more ultra-peers
  - This can offer dramatic improvements in terms of the number of hops required for exhaustive search
  - The ultra-peers are the Kevin Bacons of the peer-to-peer system

# Peer-to-Peer Systems

## Query Routing Protocol

- In this system peers exchange information about the files/resources they have available
- For example each peer may gather together a set of words in the file names of their available files.
- These words are then sent to the associated ultra-peer
- The ultra-peer collates all these into a single table of available 'words' and exchanges this information with its neighbouring ultra-peers
- So when a (text based) search query is made each ultra-peer knows which search paths are likely to obtain positive results

# Peer-to-Peer Systems

## Peerson

- Peerson (`www.peerson.net`) is a distributed peer-to-peer social network akin to Facebook
- Encryption is utilised heavily in order to provide security of user data
- This is in contrast to centralised servers which may encrypt stored data, but then the keys are stored in the same place
- In Peerson encryption keys are required to access any files (parts of a user's profile)
- The user has control over who may obtain those keys

# Peer-to-Peer Systems

## Summary

- We began with looking at the motivations behind the development of peer-to-peer systems
  - Break the reliance of the system on a central server which may be vulnerable from attack, both technical and bureaucratic
  - Utilising the resources of those using the server such that capacity grows with the number of users
  - Providing anonymity to content providers
- The now defunct pioneering system *Napster*
  - Napster relied on a central server, but that server hosted no content, bandwidth to the central server was limited as well because no content was therefore downloaded from the central server
  - Instead the central server was merely used by remote peers to locate content and setup independent connections between peers
  - Ultimately though the reliance on a central server proved enough fodder for the entertainment industry's lawyers and Napster was shutdown

# Peer-to-Peer Systems

## Summary

- Napster however proved the feasibility of the concept and several services grew into the space left behind by Napster
- Such services do not rely on any single central server and have so far proved resilient to legal attacks
- However we focused our attention on efforts to provide a generic framework for building peer-to-peer applications
- Such frameworks currently focus on providing a distributed hash table, storing objects and replicas at multiple peers for later retrieval
- Distributed Object Location and Routing systems are an extension providing a more convenient API, in particular for objects which may be updated

# Peer-to-Peer Systems

## Summary

- In general though peer-to-peer systems have and continue to be used mostly for file sharing
- In particular the sharing of immutable files such as music files and video files
- Objects tend to be visited exactly once by a user and hence unstructured networks flourish as the additional structure provided by a distributed data structure cannot be put to great use
- The low cost and dynamic service provision mean that they are continued ot be offered by those with small budgets
- Large corporations such as Microsoft, Apple, Google, Facebook and Twitter are yet to embrace peer-to-peer applications

# Any Questions

Any Questions?